Capitolo 1

PROGRAMMAZIONE DI SISTEMI

1.1 Introduzione

Prima di immergerci nello studio dei sistemi operativi, cerchiamo di convincere il lettore che vi sono validi motivi per affrontare tale studio. La nostra tesi è che esistono vari tipi di programmazione e che, soltanto conoscendo bene i sistema operativi, risulta possibile scrivere programmi di un certo tipo.

In effetti, oltre alla programmazione più semplice che si impara in poche settimane, esistono altri tipi di programmazione che richiedono anni di duro impegno per poterli dominare correttamente.

Questo fatto è evidenziato nel mondo produttivo da apposite qualificazioni professionali: il "programmatore" è nella gerarchia aziendale qualcosa di assai diverso dal "programmatore di sistemi".

Per i nostri scopi, sorvoleremo sul fatto che esistono vari tipi di programmazione di sistema e distingueremo soltanto tre livelli di programmazione:

1. indipendente dall'hardware e dal sistema operativo
2. dipendente dal sistema operativo ma indipendente dall'hardware
3. dipendente dal sistema operativo e dall'hardware

Vediamo in cosa differiscono tali tipi di programmazione.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

Programmi indipendenti dall'hardware e dal sistema operativo

Il primo livello è ovviamente quello che garantisce la massima portabilità dei programmi tra piattaforme hardware e software di tipo diverso. Le specifiche di molti linguaggi ad alto livello quali Cobol, C o C++ sono state rigidamente definite da appositi comitati di standardizzazione. In conseguenza, risulta possibile scrivere programmi che possono essere compilati ed eseguiti correttamente su calcolatori di tipo diverso e/o su calcolatori che usano sistemi operativi di tipo diverso.

Un altro classico esempio di linguaggio che offre una ottima portabilità dei programmi è il linguaggio Java oggi molto usato nel mondo Internet.

In effetti molte delle pagine che si possono caricare da un sito remoto Internet includono non solo informazioni grafiche nel formato html ma anche programmi scritti in Java. Quando una di tali pagine viene caricata da un utente sulla propria macchina, il programma scritto nel linguaggio sorgente Java viene copiato dalla macchina remota ed eseguito sulla macchina locale. Per ragioni di sicurezza nonché di portabilità, vengono trasferiti soltanto programmi scritti in linguaggio sorgente. Una volta giunti a destinazione, tali programmi sono interpretati dalla "macchina virtuale Java", ossia dal calcolatore dotato di un interprete Java.

Programmi dipendenti dal sistema operativo ma indipendente dall'hardware

Per quanto versatili siano i linguaggi di programmazione sviluppati negli ultimi anni, essi non consentono di accedere direttamente ad informazioni gestite dal sistema operativo. Ciò viene fatto di proposito per rendere il linguaggio di programmazione indipendente dal sistema operativo.

Vi sono tutttavia casi in cui il programmatore deve potere avere accesso ad informazioni gestite dal sistema operativo, ad esempio per leggere informazioni riguardanti file, quali ad esempio la data dell'ultima modifica oppure la lunghezza di un file espressa in byte.

Quando ciò si verifica, è necessario fare uso di apposite funzioni di interfaccia offerte dal sistema operativo. Vedremo più avanti in questo capitolo come ciò avviene. A questo punto, vogliamo soltanto evidenziare come questo tipo di programmazione chiamata *programmazione di sistema* consente di sviluppare programmi più sofisticati a scapito però della portabilità: un programma

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

di sistema potrà essere ricompilato ed eseguito correttamente su macchine diverse, purché esse usino tutte lo stesso sistema operativo (Windows NT, Linux, AIX, ecc.)

Programmi dipendenti dal sistema operativo e dall'hardware

Esiste un terzo livello di programmazione (riservato a pochi specialisti) a cui è necessario ricorrere quando la programmazione di sistema non è sufficiente. Ciò avviene quando si richiede di scrivere un nuovo sistema operativo, oppure quando si richiede di ampliare le funzionalità di un sistema operativo esistente.

In tale caso, ovviamente, non è possibile avvalersi di apposite funzioni di interfaccia ma, al contrario, viene richiesto al programmatore di realizzarne delle nuove.

Un caso classico è quello della realizzazione dei cosiddetti "driver di I/O" per nuove periferiche. Quando viene immessa sul mercato una nuova apparecchiatura periferica (masterizzatore, sound card ,ecc.), è necessario aggiungere al sistema operativo prescelto un nuovo driver per la periferica in modo che essa possa essere "riconosciuta" dal sistema operativo.

Come vedremo alla fine del corso, i driver di I/O sono dei programmi inclusi nel sistema operativo che interagiscono con la scheda hardware di interfaccia del dispositivo di I/O. In questo senso, sono programmi aventi un grado di portabilità praticamente nullo, in quanto possono essere eseguiti soltanto su piattaforme hardware dello stesso tipo che fanno uso dello stesso sistema operativo.

1.2 Le librerie di programmi

Per semplicità, introdurremo la nozione di libreria di programmi facendo riferimento al linguaggio C, fermo restando che ogni linguaggio di programmazione include le proprie librerie di programmi.

Ogni linguaggio di programmazione offre la possibilità di segmentare il programma sottoprogrammi più semplici. Nel linguaggio C, il programma è composto da una o più *funzioni*. La funzione principale, quella che viene eseguita per prima quando il programma viene eseguito, è chiamata `main()`. Altre funzioni possono essere introdotte a piacere dal programmatore.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

Per semplificare il lavoro del programmatore, i produttori di software hanno messo a punto collezioni di funzioni di uso comune raggruppate in *librerie*[1] specializzate.

Dal punto di vista della portabilità dei programmi, è necessario distinguere due tipi di librerie:

- librerie standardizzate dal linguaggio di programmazione
- librerie non standardizzate dal linguaggio di programmazione

La differenza tra i due tipi di librerie è cruciale dal punto di vista della portabilità dei programmi: un programma mantiene la portabilità tra sistemi basati su piattaforme hardware e software diverse se fa uso di funzioni appartenenti a librerie del primo tipo. Vice versa, non è più garantita la portabilità dei programmi che fanno uso di funzioni incluse in librerie del secondo tipo.

Librerie standardizzate

Nel linguaggio C, per potere fare uso di una funzione standard, ossia appartenente ad una libreria standardizzata, è necessario premettere nel programma una apposita frase di tipo:

```
#include <xxx.h>
```

ossia un *header file* che descrive i prototipi di tutte le funzioni incluse nella libreria (o nelle librerie) associate all'identificatore `xxx`. Se, ad esempio, si intende usare la ben nota funzione `fwrite()` per scrivere caratteri di testo, è necessario premettere nel programma la frase:

```
#include <stdio.h>
```

che descrive i prototipi di tutte le funzioni dello "stream I/O".

Si noti come l'uso delle `#include` lascia all'implementatore del sistema operativo totale libertà riguardo al modo di realizzare le librerie. Quello che viene standardizzato è il prototipo della funzione e il nome dello header file. In

[1] Il termine librerie è un anglicismo entrato nel vocabolario comune. In italiano, sarebbe più corretto parlare di biblioteche.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

Funzione	#include	Descrizione
`fopen()`, `fprintf()`, ecc.	`<stdio.h>`	stream I/O
`malloc()`, `free()`, ecc.	`<stdlib.h>`	gestione memoria
`isalnum()`, `isdigit()`, ecc.	`<ctype.h>`	elaborazione caratteri
`strcat()`, `strcmp()`, ecc.	`<string.h>`	elaborazione stringhe
`memcmp()`, `memcpy()`, ecc.	`<string.h>`	elaborazione aree di memoria
`abs()`, `rand()`, ecc.	`<math.h>`	funzioni matematiche
`gmtime()`, `clock()`, ecc.	`<time.h>`	funzioni di temporizzazione
`execl()`, `exit()`, ecc.	`<stdlib.h>`	funzioni di controllo
`signal()raise()`, ecc.	`<signal.h>`	gestione segnali

Tabella 1.1: Funzioni di librerie standardizzate.

Linux, lo header file `stdio.h` è contenuto nel file `/usr/include/stdio.h`[2]. Se apriamo tale file tramite un editor di testo, ritroviamo in corrispondenza alla `fwrite()` una dichiarazione del tipo:

```
/* Write chunks of generic data to STREAM.  */
extern size_t fwrite __P ((__const void *__restrict __ptr,
                           size_t __size, size_t __n,
                           FILE *__restrict __s));
```

che specifica i ben noti 4 parametri:

1. un puntatore al primo elemento da scrivere
2. la dimensione in caratteri di un singolo elemento
3. il numero di elementi da scrivere
4. un puntatore allo stream di output

Per comodità, abbiamo indicato nella tabella 1.1 le principali funzioni standardizzate del linguaggio C ed il tipo di dichiarazione `#include` che bisogna premettere nel programma per poterle usare.

[2]In realtà, la collocazione degli header file può variare a seconda della distribuzione utilizzata. Negli esempi che seguono, faremo riferimento alla distribuzione Slackware.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

Librerie non standardizzate

Esistono numerose interessanti librerie non standardizzate e non possiamo pretendere di elencarle tutte. Un semplice esempio sarà però sufficiente ad illustrare le potenzialità d'uso di talunne librerie.

Supponiamo di volere realizzare una interfaccia a menu tramite un programma C. Le funzioi dello stream I/O non consentono di posizionare il cursore sullo schermo né di leggere caratteri da tastiera, se non dopo avere premuto il tasto di INVIO. Per ovviare alle suddette limitazioni, sono molte usate le librerie `ncurse` e `SLang`. Illustreremo alcune delle funzioni di quest'ultima libreria in una prossima esercitazione.

Ancora una volta, scrivere un programma in ANSI che faccia uso di librerie non standardizzate ne limita la portabilità: il programma sarà senz'altro compilato correttamente su una piattaforma hardware/software diversa ma non è detto che nel nuovo sistema sia presente la libreria non standard richiesta. In teoria, dovrebbe essere incluso il sorgente della libreria oltre a quello del programma per aver la certezza della portabilità.

1.3 Realizzazione di funzioni di libreria: il ruolo delle API

Esaminando con attenzione la tabella 1.1, il lettore osserverà che alcune delle funzioni di libreria standard possono tranquillamente essere programmate in C mentre altre non possono esserlo in quanto necessitano di informazioni non disponibili all'interno di tale linguaggio.

Le funzioni aritmetiche, ad esempio, oppure quelle che elaborano caratteri, possono essere realizzate interamente in C. Ma come realizzare in C una funzione come `gmtime()` che restituisce la data e l'ora corrente, oppure una funzione come `exit()` che causa la terminazione immediata del programma?

La risposta alla suddetta domanda è semplice: numerose funzioni di libreria sono funzioni speciali chiamate *API*, un acronimo per Application Programming Interface.

Le API non sono altro che funzioni di interfaccia tra il sistema operativo e l'utente che facilitano l'uso da parte del programmatore dei servizi offerti dal sistema operativo. Diamo un paio di esempi per chiarire tale concetto.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

Le API `malloc()` e `free()` consentono di allocare e deallocare aree di memoria. Esse effettuano quindi richieste al sistema operativo, rispettivamente, per allocare e de-allocare memoria. Vedremo in seguito come ciò viene effettivamente realizzato.

Le API dello stream I/O come `fopen()`, `fprint()`, ecc. effettuano quindi richieste al sistema operativo per avviare od effettuare operazioni di I/O, oltre ad offrire uno schema di bufferizzazione dei dispositivi di I/O coinvolti (stream).

Torniamo ancora una volta, l'ultima, sul discorso della portabilità dei programmi. Le API dipendono dal particolare sistema operativo utilizzato: le API di Windows XP non hanno niente a che vedere con quelle di Linux. Ciò nonostante, la definizione delle funzioni di libreria C per i due suddetti sistemi operativi è la stessa. La portabilità di programmi C è quindi realizzata in pieno, pur lasciando al realizzatore del compilatore C la libertà di implementare le funzioni di libreria facendo uso delle API più adatte tra quelle offerte dal sistema operativo.

1.4 La programmazione di sistemi: una anteprima

Le lezioni ed esercitazioni di questo corso anno l'intento di illustrare i rudimenti della programmazione di sistema facendo riferimento al linguaggio C ed al sistema operativo Unix (in particolare Linux). Come potete notare dal calendario delle lezioni, le tre principali aree considerate saranno la programmazione di I/O, la programmazione concorrente e la gestione di terminali alfanumerici.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

Capitolo 2

EVOLUZIONE DEI SISTEMI OPERATIVI

2.1 Introduzione

Lo studio del software di base in generale e dei sistemi operativi in particolare si rivela complesso in quanto doppiamente condizionato, sia dalla architettura (hardware) del calcolatore per cui è progettato tale software, sia dalle diverse esigenze dell'utenza a cui il sistema finale (hardware più software) è destinato.

L'approccio seguito in questa dispensa è quello di scindere, per quanto possibile, le esigenze dell'utenza, e quindi le caratteristiche funzionali del software di base, dalle tecniche di realizzazione. A tale scopo, si introducono alcuni concetti preliminari attinenti all'area del software di base. Successivamente, si illustrano, prescindendo dai criteri realizzativi, le funzionalità di varie classi di sistemi operativi mettendo in evidenza, allo stesso tempo, l'evoluzione storica che tali sistemi hanno subito in funzione delle innovazioni tecnologiche conseguite.

2.2 Software di base

A differenza di altre macchine, il calcolatore, considerato come congerie di circuiti hardware, non è un oggetto facilmente utilizzabile di per sè stesso: esso è in grado di eseguire istruzioni e quindi programmi, purché tali programmi siano scritti in linguaggio macchina binario, l'unico riconosciuto dal

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

calcolatore. Inoltre, quando un calcolatore inizia ad operare in seguito all'accensione, esso cerca in una area di memoria (generalmente, su una traccia di disco prefissata) le istruzioni da caricare in memoria e da eseguire per prime. Come è ovvio, questa inizializzazione della macchina, chiamata bootstrapping, sarà effettuata con successo, posto che la traccia di disco letta contenga effettivamente un programma scritto in linguaggio macchina.

Si desume da quanto appena detto che è praticamente impossibile per l'utente utilizzare direttamente l'hardware di un calcolatore, anche accettando di programmare in linguaggio binario, a meno di non disporre di strumenti che gli consentano, da un lato, di immettere programmi nella macchina e, dall'altro, di estrarre dalla memoria verso l'esterno i risultati dei calcoli eseguiti.

Fortunatamente, già negli anni '40 i progettisti delle prime macchine scoprirono che era conveniente sviluppare programmi in maniera modulare introducendo sottoprogrammi (subroutine) di uso comune e richiamandoli dal programma principale: era nato il software di base inteso come corredo di programmi standard accessibili all'utente con lo scopo di facilitare l'interazione uomo/macchina.

Da allora, il software di base è evoluto di pari passo all'hardware con delle marcate interdipendenze: sviluppi nel campo delle architetture e delle apparecchiature periferiche rendono necessario lo sviluppo di nuovo software di base; allo stesso tempo, tecniche di programmazione collaudate condizionano la progettazione dei più recenti microprocessori.

Anche se non esistono definizioni rigorose, si indica come software di base tutti i programmi che possono essere utilizzati da utenti diversi e che consentono di sviluppare più agevolmente altri programmi. A differenza dei programmi applicativi che, come il nome indica, servono a specializzare il calcolatore per risolvere problemi legati al mondo della produzione e del lavoro, i programmi nell'area del software di base si collocano a un livello intermedio tra l'hardware e i programmi applicativi. Un programma di gestione magazzino sviluppato per una azienda è un esempio di programma applicativo; viceversa, un programma flessibile che consente, partendo da uno schema generale, di costruire un programma di gestione magazzino, fornendo informazioni specifiche sul modo in cui l'azienda registra i prodotti immagazzinati, può essere considerato software di base.

Si indicano, infine, con il nome di sistemi operativi corredi di programmi

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

messi a punto per calcolatori i quali costituiscono in qualche modo il livello di software di base più vicino all'hardware.

Anche in questo caso, l'area di demarcazione tra sistema operativo e software di base risulta sfumata: un sistema di archiviazione (file system) è considerato come parte essenziale di un sistema operativo; un compilatore può esserlo o meno a seconda dalla presenza o assenza di un supporto di esecuzione (run-time environment) per il linguaggio di programmazione; un programma di ordinamento (sort) di record per chiavi crescenti, infine, è generalmente considerato come programma di utilità (software di base) ma non come parte del sistema operativo.

2.3 Classificazione dei sistemi operativi

Risulta opportuno, prima di procedere oltre nello studio dei sistemi operativi, dare una schematica classificazione di tali sistemi che metta in evidenza le caratteristiche dell'utenza che essi devono soddisfare, più che le tecniche realizzative hardware e software. A tale scopo, si distinguono cinque tipi di sistemi operativi chiamati, rispettivamente, dedicati, a lotti, interattivi, transazionali e per il controllo di processi.

I sistemi operativi dei primi tre tipi offrono all'utente la possibilità di scrivere, mettere a punto ed eseguire programmi arbitrari scritti in uno dei linguaggi di programmazione sopportati dal sistema. L'utente tipico di tali sistemi che saranno descritti nel prossimo paragrafo è quindi il programmatore che accede al calcolatore tramite un terminale. Il dialogo utente/sistema si svolge inviando comandi da terminale e ricevendo in risposta messaggi audiovisivi dal sistema. Come si vedrà in seguito, le funzioni del sistema operativo nell'assistere il programmatore vanno ben oltre la semplice compilazione ed esecuzione controllata di programmi per cui si parla comunemente di ambiente di sviluppo offerto dal sistema operativo, intendendo con tale termine l'insieme di servizi offerti che consentono una maggiore facilità nella progettazione, nella stesura e nella verifica di nuovo software. Nell'ambito dei sistemi operativi che offrono ambienti di sviluppo, è doveroso citare anche i cosiddetti sistemi di sviluppo, ossia una classe di sistemi operativi specializzati che consentono di sviluppare in modo agevole programmi da trasferire successivamente nelle memorie ROM di microprocessori.

I sistemi transazionali sono invece dei sistemi operativi disegnati attorno

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

ad una specifica applicazione: l'utente non è più libero di "inventare" nuovi programmi per poi richiederne l'esecuzione ma può soltanto scegliere da un menù prefissato quale funzione intende eseguire; fatto ciò, egli deve in generale immettere i dati necessari per calcolare la funzione. Esempi tipici di sistemi transazionali sono i sistemi di prenotazione posti o i sistemi per la movimentazione dei conti correnti bancari. Le interazioni dell'utente con il sistema operativo prendono il nome di transazioni (l'annullamento di una prenotazione oppure un versamento in conto corrente sono esempi di transazioni).

Nei casi esaminati in precedenza, si è supposto che il sistema operativo svolga una funzione di interfaccia tra l'hardware del calcolatore e un operatore al terminale. Nel caso dei sistemi per il controllo di processi, tale ipotesi viene in parte o del tutto a cadere: l'obiettivo principale di tali sistemi è quello di acquisire periodicamente dati provenienti da sensori e altre apparecchiature remote, di analizzare tali dati e di emettere quindi, in seguito all'analisi, opportuni segnali ad apparecchiature di uscita. I sistemi di controllo si dividono a loro volta in due categorie:

- sistemi ad anello aperto: esiste un operatore che analizza i segnali ricevuti dall'esterno in tempo reale e prende le opporune decisioni.
- sistemi ad anello chiuso: non è previsto alcun intervento umano; i segnali ricevuti dall'esterno sono elaborati e danno luogo a comandi inviati ai vari dispositivi collegati.

Un sistema di controllo di traffico aereo è ad anello aperto in quanto il controllore di volo recepisce le immagini inviate dai radar sullo schermo e comunica via radio ai piloti le opportune istruzioni.
I sistemi di controllo per centrali telefoniche elettroniche ed i sistemi di pilotaggio automatico sono ad anllo chiuso in quanto non è previsto l'intervento di operatori umani.
In un certo senso, i sistemi per il controllo di processi si possono considerare sistemi transazionali con interfaccia sintetica e quindi, in generale, con vincoli più stringenti sui tempi di risposta agli eventi esterni.

Concludendo, si ricorda che la classificazione introdotta prescinde dalle caratteristiche hardware del sistema: un home computer con un interprete Basic microprogrammmato include un sistema operativo dedicato che offre all'utente un rudimentale ambiente di sviluppo; viceversa, un mainframe con centinaia di terminali collegati può essere utilizzato prevalentemente come sistema

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

transazionale. I collegamenti tra l'architettura del calcolatore e la struttura del sistema operativo saranno invece esaminati in capitoli successivi.

2.4 Evoluzione storica

Per mettere in evidenza la stretta interdipendenza tra l'hardware dei calcolatori e i sistemi operativi, è interessante passare brevemente in rassegna i diversi tipi di colloquio uomo-macchina che sono stati messi a punto dagli albori dell'informatica a oggi, con particolare riferimento ai sistemi dedicati, batch e interattivi.

2.4.1 Sistemi dedicati (prima versione)

La prima fase che si protrae fino all'inizio degli anni '50 può essere definita come quella dei sistema dedicati. I vari utenti dispongono a turno dell'intero sistema: durante l'intervallo di tempo concesso (nel lontano 1950, il tempo macchina era una risorsa pregiata concessa a pochi utenti privilegiati per brevi periodi), il programmatore carica programmi in memoria, li fa eseguire e ne controlla l'esecuzione. Per richiamare le poche funzioni del sistema operativo, invia appositi comandi impostando interruttori e commutatori della console (la tastiera non era ancora stata introdotta) mentre, per controllare l'esecuzione del programma, legge i valori di alcuni registri rappresentati sul pannello tramite indicatori luminosi.

Il calcolatore Universal Electronic Computer costruito dall'università di Manchester utilizzava un lettore/perforatore di nastro con caratteri di 5 bit per leggere programmi dall'esterno. Non esisteva linguaggio assemblativo per cui la programmazione era effettuata in una alfabeto a 32 simboli e i caratteri di tale alfabeto erano trasferiti come gruppi di 5 bit in memoria. Come è stato accennato in precedenza, il caricamento di sottoprogrammi si effettuava in modo statico a partire da un indirizzo di base prefissato: si distingueva la "programmazione in grande" che consisteva nel calcolare i vari indirizzi di base dei sottoprogrammi tenendo conto delle loro lunghezze dalla "programmazione in piccolo" che consisteva invece nel mettere a punto un singolo sottoprogramma. Questa seconda attività era generalmente effettuata direttamente durante l'esecuzione del programma: appositi pulsanti e commutatori della console consentivano di fermare l'esecuzione del programma o di rallentarla in modo da consentire al programmatore di leggere i valori

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

assunti dai registri interni della macchina. Il sistema operativo di tale calcolatore consisteva essenzialmente in alcuni programmi per il trasferimento di blocchi di dati tra la memoria principale a quella ausiliaria costituita da un tamburo magnetico.

2.4.2 Gestione a lotti

La gestione appena descritta non era soddisfacente; il calcolatore considerato come macchina costosa e sempre più potente doveva essere sfruttato meglio: in particolare i periodi in cui l'utente bloccava la macchina per meglio controllare l'esecuzione del suo programma portavano ad una scarsa percentuale di utilizzazione della CPU. La risposta a tale problema venne fornita da un lato dalla introduzione di lettori di schede perforate e dall'altro da un primo affinamento del sistema operativo: nacque cosí all'inizio degli anni '50 una modalità di colloquio chiamata gestione a lotti (batch processing) che inibiva in pratica all'utente l'uso della console. Le principali caratteristiche di tale forma di colloquio possono cosí essere sintetizzate.

- Le richieste degli utenti sono costituite da pacchi di schede perforate; ogni pacco, chiamato lavoro o *job* inizia con una scheda di identificazione dell'utente e consiste in una sequenza di passi o *step*.

  Ogni step rappresenta una richiesta svolta dall'utente al sistema operativo per ottenere un servizio elementare generalmente associato alla esecuzione di un programma. Esempi di step sono la compilazione di un programma oppure l'esecuzione di un programma già compilato in precedenza. La scheda di identificazione del lavoro e le altre schede che servono a definire i vari step sono chiamate schede di controllo. Anche i programmi e gli eventuali dati devono apparire nel lavoro che consiste quindi in una alternanza di schede di controllo e di schede dati; queste ultime possono contenere un programma da compilare oppure delle informazioni alfanumeriche che saranno usate durante l'esecuzione del programma.

- Eseguito un lavoro, l'utente ottiene dal sistema operativo un tabulato dove sono stampati consecutivamente i risultati prodotti dai vari step eseguiti. Poiché non è più possibile accedere alla console, il sistema operativo deve registrare in tale tabulato, oltre agli eventuali risultati,

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

tutti i messaggi e diagnostici che si sono verificati durante l'esecuzione del lavoro.

- Il sistema operativo include un programma di controllo chiamato batch monitor che svolge le seguenti funzioni:
  - legge da lettore di schede il prossimo lavoro; se il lettore di schede è vuoto, rimane in attesa di un segnale dell'operatore che indica che sono state collocate altre schede sul lettore;
  - esegue successivamente gli step del lavoro letto e provvede, ad ogni step, a caricare da nastro magnetico o da disco l'apposito programma per poi passargli il controllo;
  - al termine dell'esecuzione di un step, inizia lo step successivo, oppure, se lo step eseguito era l'ultimo, inizia la decodifica della scheda di identificazione del job successivo e quindi l'esecuzione del primo step del nuovo job.

Usando lo schema appena descritto, ogni step di un job consiste nel leggere schede, eseguire istruzioni e stampare risultati. Poiché il lettore di schede e la stampante sono lente rispetto alla CPU, si verifica frequentemente, durante l'esecuzione di uno step, che la CPU rimanga bloccata in attesa di segnali del tipo "fine lettura schede" o "fine scrittura riga di stampa". Furono quindi introdotti perfezionamenti nella gestione a lotti, resi possibili dalla introduzione di nuove unità periferiche e di calcolatori dotati di più processori autonomi, per rimuovere tale collo di bottiglia.

In una prima fase, con l'introduzione di lettori di nastro magnetico aventi un elevato tasso di trasferimento, furono introdotte architetture che prevedevano sistemi ausiliari per effettuare la conversione scheda-nastro e quella nastro-scheda. Il calcolatore principale non interagiva più con periferiche lente ma solo con unità a nastro mentre i calcolatori ausiliari svolgevano in parallelo le attività di conversione.

Anche se serviva a rimuovere il collo di bottiglia citato in precedenza, l'uso di elaboratori ausiliari poneva problemi di gestione: il numero di nastri da montare e smontare era notevole e gli errori degli operatori frequenti.

Il perfezionamento successivo si ebbe verso la fine degli anni '50 quando apparvero i primi calcolatori con canali di I/O autonomi nonché le prime

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

memorie a disco di grande capacit: in pratica, venne ripresa l'idea precedente di fare interagire la CPU prevalentemente con unità periferiche veloci, in questo caso i dischi. Tuttavia, grazie ai canali di I/O integrati nel calcolatore, non era più necessario ricorrere a calcolatori ausiliari in quanto tali canali sono dei processori specializzati in grado leggere schede, stampare dati o interagire coi dischi, interferendo in modo ridotto con le attività della CPU (esistono in realtà interferenze dovute agli accessi simultanei alla memoria da parte dei vari processori e al fatto che la sincronizzazione canale di I/O-CPU avviene tramite interruzioni che richiedono, a loro volta, l'esecuzione sulla CPU di brevi programmi di gestione; comunque, tali interferenze possono essere alquanto contenute in un sistema ben dimensionato). La soluzione tuttora adottata ha portato all'uso di dischi di spooling (Simultaneous Peripheral Operations On Line): il disco di spooling di ingresso contiene "schede virtuali", ossia schede lette dal lettore e contenenti job non ancora eseguiti, quello di spooling di uscita contiene invece "tabulati virtuali", ossia tabulati già prodotti relativi a job eseguiti ma non ancora stampati. I canali di I/O provvedono a riempire o svuotare i dischi di spooling e a gestire i trasferimenti tra memoria e dischi mentre la CPU può essere quasi interamente dedicata ad eseguire programmi degli utenti o del sistema operativo.

Verso la fine degli anni '60 la gestione a lotti subisce, grazie ai progressi conseguiti nel campo dell'hardware e delle telecomunicazioni, una importante e radicale evoluzione consistente nell'eliminare l'ormai antiquato lettore di schede, consentendo agli utenti di sottomettere job da terminali remoti.

L'introduzione di circuiti integrati e la conseguente diminuzione dei costi dei componenti hardware fa sí che è ormai possibile servirsi di "terminali intelligenti", sia per editare un job da inviare successivamente ad un mainframe perché venga eseguito con le modalità della gestione a lotti, sia per visualizzare, archiviare o stampare i risultati del job eseguito. Lo sviluppo di reti di trasmissione dati ha inoltre consentito di distribuire i punti di accesso al grande calcolatore, ampliando in questo modo l'utenza potenziale.

Attualmente, emergono due aree di applicazioni in cui la gestione a lotti si rivela più conveniente di quella interattiva (vedi sottoparagrafo successivo): in primo luogo, negli ambienti di produzione in cui sono utilizzate in modo periodico, e quindi prevedibile, procedure complesse che impegnano il calcolatore per delle ore consecutive, la gestione a lotti, oltre a consentire di sfruttare appieno la potenza di calcolo del sistema, consente anche di stabilire un calendario di esecuzione (schedule) dei job e dà quindi al gestore del

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

centro di calcolo la certezza che un determinato job sarà eseguito entro una data prefissata.

In secondo luogo, la gestione a lotti rimane la più adatta ogni qualvolta la risorsa di calcolo ha un costo molto elevato: questo si verifica per i supercalcolatori utilizzati per eseguire calcoli particolarmente impegnativi. In tale caso, il costo di simili macchine suggerisce di ottimizzarne l'uso facendo in modo che i programmi degli utenti siano già messi a punto su macchine meno costose e ricorrendo alla gestione a lotti per sfruttare appieno la potenza del supercalcolatore.

2.4.3 Sistemi interattivi

Per quanto efficiente, la gestione a lotti non è la più adatta in ambienti in cui lo sviluppo e la messa a punto di programmi sono considerate attività prioritarie: in questo caso, infatti, il dovere attendere che termini il job precedente prima di potere effettuare una nuova prova limita notevolmente la produttività dei programmatori. Per ovviare a tale inconveniente, fu sviluppato nel 1963 presso il Massachussetts Institute of Technology (MIT) il sistema operativo Compatible Time Sharing System (CTSS) basato su un elaboratore IBM 7090 con un banco aggiuntivo di memoria che può essere considerato il primo dei sistemi operativi di tipo interattivo.

L'idea di base dei sistemi interattivi, anche chiamati a partizione di tempo (in inglese, time-sharing), è quella di assicurare, ad ogni utente collegato tramite terminale, una frazione garantita delle risorse del calcolatore, a prescindere dalle richieste degli altri utenti collegati. Questo si ottiene tramite un multiplamento nel tempo della CPU tra i vari programmi associati ai terminali attivi: ogni programma ha diritto a impegnare la CPU per un quanto di tempo (tipicamente, si usano quanti di poche centinaia di millisecondi, anche se sarebbe più corretto misurare il quanto in termini di numero di istruzioni eseguibili durante un quanto). Se il programma non termina entro il quanto prefissato, esso viene posto in lista d'attesa ed un nuovo programma riceve il quanto successivo. Se vi sono N terminali attivi, ogni utente ha la certezza di ottenere un quanto di tempo ogni N, qualunque siano i tempi di esecuzione degli altri programmi.

A prima vista, si potrebbe pensare che un sistema interattivo appaia agli N utenti collegati come un calcolatore N volte più lento di quello reale; in pratica, le prestazioni sono molto migliori poiché, durante l'attività di messa

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

a punto di programmi, l'utente alterna fasi di editing del programmi nonché fasi di analisi dei risultati e ripensamenti (think time) a fasi di compilazione e esecuzione per cui una buona parte delle richieste degli utenti (editing, presentazione dei risultati) può essere esaudita prevalentemente dai canali di I/O con scarso impegno della CPU.

I sistemi interattivi hanno avuto un notevole impatto nello sviluppo dei sistemi operativi: è stato necessario predisporre schemi hardware e software per proteggere i programmi e gli archivi degli utenti attivi nel sistema da danni causati da programmi non affidabili (o indiscreti?) di altri utenti. Allo stesso tempo, la gestione rigida della CPU basata sull'assegnazione di quanti di tempo di lunghezza prefissata è evoluta per tenere conto del fatto che il sistema operativo non è in grado di conoscere a priori quanto tempo di CPU richiederà un programma. Per questo motivo, si preferisce usare una strategia flessibile: quando un programma inizia ad eseguire, esso è considerato di tipo interattivo, se dopo alcuni quanti di tempo, esso non ha ancora terminato, la sua priorità viene diminuita; infine, se oltrepassa un numero prefissato di quanti, esso viene declassato a lavoro di tipo differito e, in pratica, sarà eseguito come in un sistema di gestione a lotti assieme ad altri lavori dello stesso tipo quando non vi sono più richieste di utenti ai terminali.

2.4.4 Sistemi dedicati (seconda versione)

All'inizio degli anni '80, appaiono i primi personal computer e con essi si verifica un rifiorire dei sistemi operativi dedicati che erano scomparsi dallo scenario informatico subito dopo l'avvento dei primi sistemi di gestione a lotti. In effetti, l'evoluzione tecnologica consente di produrre oggi a costi contenuti dei calcolatori che, pure avendo un ingombro limitato, hanno una potenza di calcolo paragonabile a quella dei mainframe degli anni '70. In simili condizioni, ha di nuovo senso impostare un sistema operativo dedicato che serva un unico posto di lavoro poiché l'efficiente utilizzazione del calcolatore cessa di essere un obiettivo prioritario. Paradossalmente, questa nuova generazione di sistemi operativi offre all'utente una interfaccia che risulta per alcuni aspetti superiore a quella offerta da sistemi interattivi sviluppati per mainframe. In particolare, la disponibilità di uno schermo grafico ad alta risoluzione accoppiato all'uso di microprocessori sempre più potenti ha reso possibile la realizzazione di software grafico interattivo che offre affascinanti

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

possibilità di utilizzazione. Il sistema operativo MacOS della Apple è uno dei capostipiti più noti di questa nuova generazione di sistemi operativi dedicati.

2.5 Sviluppi futuri

E' sempre difficile fare previsioni attendibili in settori in rapida evoluzione come è quello informatico. Tuttavia, molti pensano che nel prossimo decennio si consoliderà l'affermazione dei personal computer e che, in conseguenza, i maggiori sforzi dei progettisti saranno rivolti alla progettazione di sistemi operativi per tale classe di macchine.

Probabilmente, i risultati più interessanti si conseguiranno nel settore dei sistemi dedicati in rete. Con tale termine, si intende un sistema dedicato che consente non soltanto di utilizzare il personal localmente ma anche di mantenerlo collegato in rete (ad esempio, una rete pubblica di trasmissione dati quale Internet). In questo modo, l'utente può operare localmente mentre, simultaneamente e con un minimo di interferenza, il sistema operativo provvede a inviare messaggi già preparati in precedenza a nodi remoti e, allo stesso tempo, riceve e memorizza su disco messaggi inviati da altri nodi della rete al personal. Un altro promettente settore riguarda il collegamento tramite linea dedicata veloce di un elevato numero di personal computer. Disponendo di una linea veloce di qualche Gigabyte/sec e delle opportune schede di interfaccia mediante le quali collegare i personal computer alla linea, l'intero sistema risulta assimilabile ad un calcolatore parallelo con un rapporto prestazioni/costo molto interessante, tenuto conto delle economie di scala realizzate operando nel settore dei personal computer.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

Capitolo 3

SOFTWARE DI BASE

3.1 Introduzione

Tecnicamente parlando, un sistema operativo consiste essenzialmente in quattro moduli principali:

- un programma di inizializzazione che viene eseguito quando viene avviato l'elaboratore;
- un NUCLEO che esegue *chiamate di sistema*, ossia particolari API mediante le quali i programmi possono richiedere uno dei vari servizi offerti dal NUCLEO (tratteremo questo argomento nella sezione 6.4);
- un sistema d'archiviazione (file system);
- un'interfaccia che consente all'utente di esprimere le proprie richieste.

In effetti, i testi di sistemi operativi più accreditati si limitano a trattare i suddetti argomenti. Dato il carattere introduttivo di queste dispense, riteniamo tuttavia opportuno studiare i sistemi operativi seguendo un approccio "top down", per cui iniziamo col descrivere le caratteristiche principali di alcuni importanti programmi di utilità (viene anche usato il termine software di base). Come vedremo nella prossime pagine, tali programmi risultano indispensabili ai programmatori, agli utenti generici e agli amministratori di sistema.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

3.2 Messa a punto di programmi

Iniziamo col parlare degli strumenti che storicamente sono stati quelli introdotti per primi ed il cui obiettivo è quello di facilitare l'utente nel realizzare programmi.

3.2.1 Assemblatori, compilatori e interpreti

E' stato ribadito, a proposito del software di base, che l'unico linguaggio riconosciuto dal calcolatore è il linguaggio macchina binario. Poiché è estremamente disagevole scrivere programmi servendosi di tale linguaggio, sono stati sviluppati programmi traduttori il cui compito è quello di trasformare programmi scritti in uno dei vari linguaggi di programmazione esistenti (il cosiddetto *linguaggio sorgente*) in programmi scritti in linguaggio macchina.

Il processo di traduzione ha caratteristiche diverse, a seconda del linguaggio sorgente e delle modalità di realizzazione. In generale, la traduzione precede l'esecuzione del programma. In questo caso, se le frasi eseguibili del linguaggio sorgente corrispondono alle istruzioni del calcolatore, il traduttore prende il nome di *assemblatore*. Nel caso contrario, i programmi traduttori sono chiamati *compilatori*.

Il linguaggio assemblativo per i microprocessori Intel 80x86, ad esempio, è un linguaggio sorgente simile al linguaggio macchina di tali microprocessori; esso prevede frasi eseguibili del tipo `add` oppure `mov` corrispondenti al repertorio di istruzioni riconosciuto da tali calcolatori, per cui il traduttore di tale linguaggio è chiamato assemblatore. Viceversa, il linguaggio C è un linguaggio sorgente più potente per cui ogni frase eseguibile deve essere tradotta dal compilatore in gruppi di istruzioni in linguaggio macchina (si pensi, ad esempio, alle frasi C che includono espressioni aritmetiche di arbitraria complessità).

In altri casi, la traduzione può andare di pari passo con l'esecuzione del programma: traduttori di tale tipo sono chiamati *interpreti*. Un programma interpretato ha tempi di esecuzione molto maggiori di uno compilato, proprio perché ogni frase del linguaggio sorgente ed ogni variabile può essere ripetutamente tradotta e riesaminata dall'interprete. In prima approssimazione, un programma interpretato ha un tempo di esecuzione circa 50 volte maggiore di quello di un programma compilato. In compenso, gli interpreti occupano meno memoria dei compilatori e offrono una ottima diagnostica

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

durante l'esecuzione per cui sono molto usati in piccoli sistemi e nello sviluppo di linguaggi specializzati. Tra gli interpreti più noti citiamo quello per il linguaggio Basic e quello per il linguaggio Java.

Dal punto di vista del sistema operativo, gli assemblatori e i compilatori sono dei programmi eseguibili scritti in linguaggio macchina che richiedono per la loro esecuzione un file di ingresso, uno di uscita e alcuni file ausiliari temporanei, ossia eliminabili dopo che è terminata l'esecuzione del programma compilatore. Il file d'ingresso contiene il programma sorgente preparato in precedenza servendosi di un opportuno editor. Il file di uscita contiene invece un *programma rilocabile* (sono anche usati i termini: modulo oggetto e programma oggetto). Tale programma è scritto in linguaggio macchina ma non è ancora eseguibile in quanto, solitamente, fa riferimento ad altri programmi e dati esterni, ossia non inclusi nel programma sorgente, che il compilatore non è in grado di tradurre.

Alcuni compilatori più sofisticati prevedono più fasi consecutive. Il "GNU C compiler", ad esempio, prevede tre fasi distinte:

1. *preprocessamento*: frasi del tipo:

   `#include <nomefile>`

   oppure:

   `#ifdef <variabile_compilazione>....  #endif`

   sono tradotte nelle opportune frasi C;

2. *compilazione*: il programma C viene tradotto nel linguaggio assemblativo del processore utilizzato;

3. *assemblaggio*: il programma assemblativo viene tradotto in linguaggio macchina dando luogo ad un modulo oggetto.

Questo approccio consente al programmatore di inserire sequenze di istruzioni in linguaggio assemblativo all'interno del programma sorgente. Il GNU C compiler, ad esempio, prevede la frase C `asm();` per inserire frasi in linguaggio assemblativo all'interno del programma C. Tale caratteristica risulta fondamentale per scrivere il codice di un sistema operativo: per pilotare i dispositivi di I/O, infatti, è necessario agire sulle porte di I/O mediante istruzioni in linguaggio assemblativo. Per fortuna, solo una limitata parte del sistema operativo deve necessariamente essere codificata in linguaggio

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

assemblativo. La rimanente parte, circa il 90% del codice, può essere scritta in un linguaggio ad alto livello quale il C.

3.2.2 Linkaggio di programmi

In effetti, l'uso di programmi rilocabili si rivela indispensabile in applicazioni complesse dove è facile superare le decine di migliaia di frasi: in tali casi, non è conveniente prevedere un unico programma ma è di gran lunga preferibile decomporre l'applicazione in varie unità di programma o moduli compilabili separatamente. La nozione di modulo dipende dal linguaggio sorgente utilizzato: in un linguaggio quale il C, non solo la funzione `main()` che corrisponde al programma principale, ma anche una qualsiasi funzione (o gruppi di funzione e/o strutture di dati) può essere definito come modulo.

In genere, gli standard di programmazione esistenti suggeriscono di usare moduli di lunghezza non superiore alle centinaia di frasi. A questo punto, il quadro risulta più complesso: ogni modulo è compilato separatamente e dà luogo ad un corrispondente *modulo oggetto*. Questi ultimi non sono tuttavia logicamente indipendenti: il modulo A può includere una chiamata ad una funzione contenuta in un altro modulo B, ignoto al momento della compilazione di A. Allo stesso modo, i moduli interagiscono su variabili o strutture di dati comuni contenute in moduli diversi da quello attualmente compilato. Una conseguenza della decomposizione in moduli è quindi che il compilatore non è in grado di associare un indirizzo rilocabile ad ogni funzione e ad ogni variabile contenuta in un modulo: esso esegue una traduzione parziale, creando quindi una tabella di riferimenti esterni per tutti quei nomi che non ha trovato all'interno del modulo.

La figura 3.1 illustra un semplice esempio di collegamenti tra tre moduli: il modulo A include una chiamata alla funzione `prog1()` inclusa nel modulo B. Il modulo B, a sua volta, include una chiamata alla funzione `prog2()` inclusa nel modulo C.

Quando viene compilato il modulo A, il compilatore non è in grado di "risolvere" l'etichetta `prog1`, anche se la dichiarazione del tipo `extern int prog1(...)` presente nel codice lo aiuta nello stabilire che `prog1` è effettivamente un nome esterno.

Il compilatore inserisce quindi nella tabella dei simboli esterni `extern` sia l'etichetta `prog1` che l'indirizzo (o gli indirizzi) in cui tale etichetta appare.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

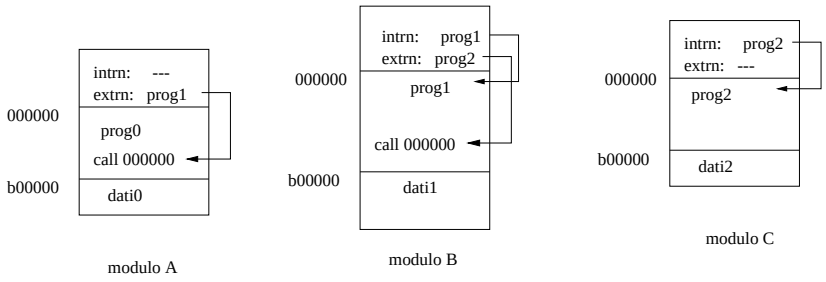

Figura 3.1: Un esempio di riferimenti tra moduli.

Lo stesso avviene durante la compilazione del modulo B. La tabella `intern` presente in ogni modulo elenca tutti i simboli definiti nel modulo insieme ai corrispondenti indirizzi in cui essi appaiono nel modulo.

Si rende quindi necessaria una nuova funzione del sistema operativo, il correlatore (in inglese, *linker*) per terminare la compilazione dei vari moduli. Servendosi delle indicazioni contenute nelle tabelle dei riferimenti esterni presenti in ogni modulo oggetto, il correlatore fonde i vari moduli oggetto in un unico file e completa all'interno dei moduli la compilazione inserendo gli indirizzi mancanti nelle varie parti del codice.

Il risultato è un *file eseguibile* pronto ad essere caricato in memoria e quindi eseguito. Come vedremo nel capitolo 8, dopo la funzione di linking che produce un file eseguibile, viene eseguita un'altra funzione chiamata *loading* che precede immediatamente l'esecuzione del programma.

Riprendendo l'esempio precedente, possiamo vedere nella figura 3.2 come risulta il file eseguibile prodotto dal linker in corrispondenza ai tre moduli A, B e C. Le frecce tra etichette esterne e programmi indicano le "cuciture" effettuate dal linker: ad esempio, accanto all'etichetta `prog2` nel modulo B, il linker ha inserito l'indirizzo iniziale di `prog2` nel modulo linkato. Durante l'esecuzione del programma, `prog2` potrà essere correttamente invocato tramite indirizzamento indiretto.

3.2.3 Librerie statiche e dinamiche

Come abbiamo già visto nella sezione 1.2, l'esistenza di correlatori offre ai programmatori la possibilità di ricorrere a programmi già messi a punto da

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

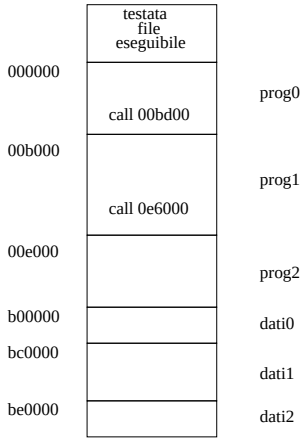

Figura 3.2: Un esempio di collegamenti tra moduli.

altri: sono le cosiddette librerie di programmi che hanno oggi un ruolo fondamentale nella messa a punti dei programmi. Tutti linguaggi di programmazione fanno un ampio uso di tali librerie, per cui non è addirittura possibile eseguire alcun programma in un qualsiasi linguaggio se non lo si correla con numerosi altri programmi inclusi in una o più librerie.

Come è ovvio, ogni linguaggio di programmazione ha le sue apposite librerie, per cui vi saranno librerie di programmi per il linguaggio C, per il C++, per il Cobol e così via.
Nell'effettuare i collegamenti tra le funzioni C scritte dall'utente e quelle già presenti in apposite librerie, il collegatore o linker può fare uso di due tipi di librerie, ognuna delle quali presenta vantaggi e svantaggi.

- *libreria statica*: contiene il codice oggetto di funzioni; il collegatore aggiunge al codice oggetto corrispondente alla compilazione dei programmi dell'utente il codice di tutte le funzioni di libreria utilizzate. Il programma eseguibile corrispondente è autosufficiente, nel senso che tutti i moduli oggetto indirizzati sono contenuti nello stesso file eseguibile. Lo svantaggio di tale approccio è che la stessa funzione di libreria viene duplicata in vari file eseguibili, per cui lo spazio su disco viene sfruttato in modo poco efficace. Il compito del loader risulta alquanto semplice, proprio perché il file eseguibile è autosufficiente.

- *libreria dinamica*: il collegatore si limita ad inserire nei file oggetto

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

corrispondenti alla compilazione dei programmi dell'utente i nomi delle funzioni di libreria utilizzate e gli indirizzi nel codice in cui compaiono chiamate a tali funzioni. Durante l'esecuzione del programma, i programmi inclusi nelle librerie vengono caricati dinamicamente in una apposita area di memoria (a meno che non siano già state caricate in precedenza) e viene effettuato un collegamento dinamico tra le istruzioni del programma utente e gli indirizzi delle funzioni richieste. Come vedremo più avanti nel paragrafo *Supporto durante l'esecuzione*, il compito del caricatore risulta più complesso.

Il vero problema è tuttavia un altro: le librerie di programma evolvono nel tempo ed i parametri usati da alcune funzioni della libreria possono cambiare da una versione all'altra. Quando ciò avviene, può succedere in alcuni sistemi operativi che l'applicazione selezionata dall'utente non possa essere eseguita su un determinato sistema poiché esso fa uso di una versione della libreria incompatibile con quella richiesta dall'applicazione.

Nel file system di Unix le cartelle `/bin` e `/sbin` sono tradizionalmente riservate per contenere file eseguibili linkati facendo uso, rispettivamente di librerie dinamiche e statiche. Le cartelle `/usr/local/bin` e `/usr/local/sbin` dovrebbero essere usate dai sistemisti per inserirvi ulteriori file eseguibili linkati facendo uso, rispettivamente, di librerie dinamiche e statiche. Le librerie di programmi di uso comune sono di solito inserite nella cartella `/lib`.

Ulteriori librerie specializzate sono inserite nella cartella `/usr`, anche se non esiste ancora uno standard che specifica, per ogni tipo di libreria, la corretta collocazione nell'albero delle cartelle. Un semplice esempio può servire ad illustrare l'affermazione precedente.

Nelle distribuzioni di Linux, la libreria di funzioni per l'interfaccia grafica X Windows ha il nome di percorso `/usr/X11R6/lib`. Altre librerie specializzate, quali quella che contiene le funzioni attinenti alla rappresentazione di immagini nel formato jpeg, sono incluse nella cartella `/usr/local/lib`. In altri casi ancora, diversi programmi applicativi codificati in modo modulare fanno uso di librerie proprie aggiuntive che devono essere inserite in apposite cartelle. L'applicazione netscape, ad esempio, richiede una propria libreria di funzioni che deve essere inserita nella cartella `/usr/local/netscape`.

In generale, quella che può sembrare una eccessiva rigidità dell'applicazione, ossia il dovere inserire una libreria di funzioni con un nome di percorso

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

prefissato, si traduce in un notevole beneficio per l'utente. Grazie a tale approccio, è possibile inserire nei programmi applicativi linkati in modo dinamico appositi test per verificare se le librerie dinamiche installate nel sistema sono compatibili con l'applicazione da installare, segnalando eventuali incompatibilità all'utente.

Poiché ogni libreria ha un apposito nome di percorso, risulta inoltre possibile, fare convivere diverse versioni della stessa libreria: questo può risultare indispensabile per potere eseguire sia applicazioni più antiche che fanno uso di versioni precedenti di una libreria che applicazioni più recenti che invece richiedono una versione più recente della stessa libreria ed incompatibile con quella precedente.

3.2.4 Immissione di programmi

Un programma è un documento immesso dall'utente tramite tastiera che consiste in una stringa di caratteri. La funzione editor del sistema operativo assiste l'utente nel creare nuovi programmi oppure nel modificare programmi esistenti. A tale scopo, visualizza sullo schermo i caratteri appena digitati consentendo così di rilevare eventuali errori di battitura; inoltre, consente all'utente di correggere parti di documento tramite operazioni di inserimento e cancellazioni di caratteri. Infine, l'editor si serve del sistema di archiviazione, sia per memorizzare in appositi archivi su disco nuovi programmi immessi dagli utenti, sia per reperire l'archivio contenente il programma su cui operare ulteriori modifiche.

Da vari anni sono ormai disponibili editor specializzati chiamati *editor guidati dalla sintassi* per mettere a punto programmi scritti in uno specifico linguaggio di programmazione. Tali editor si avvalgono della sintassi del linguaggio per riconoscere le parole chiavi contenute nel programma, per evidenziarle tipograficamente e, sopra tutto, per effettuare un riconoscimento automatico di eventuali errori di sintassi. Il loro uso migliora sensibilmente la produttività del programmatore.

3.2.5 Caricamento in memoria di un file eseguibile

Il caricamento di un file eseguibile in memoria RAM precede immediatamente la sua esecuzione e viene effettuato automaticamente dal sistema operativo in seguito alla richiesta di esecuzione del programma da parte dell'utente.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

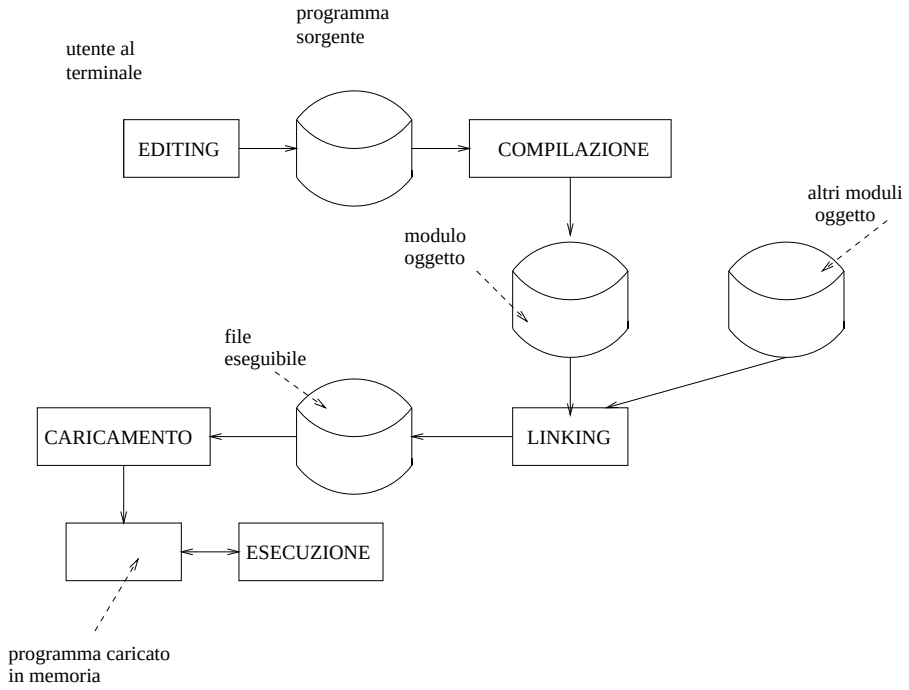

Figura 3.3: Fasi di messa a punto di un programma.

Le attività associate al caricamento dipendono dallo schema di gestione della memoria utilizzato e dal formato del file eseguibile. La Figura 3.3 illustra le fasi di messa a punto di un programma utilizzando le varie funzioni del sistema operativo appena descritte. Ritorneremo su questo argomento nel capitolo 8

3.3 Supporto durante l'esecuzione

Si è discusso in precedenza delle varie fasi attraverso le quali passa un programma fino a iniziare l'esecuzione. Supponiamo ora che l'utente al terminale richieda l'esecuzione di un programma. Inizia a questo punto da parte del sistema operativo, una fase di controllo del programma utente che si protrae fino alla sua terminazione.

Un primo tipo di controllo è quello relativo alla terminazione corretta del programma. Quando ciò si verifica, il NUCLEO provvede ad inviare un

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

apposito messaggio al processo di sistema associato al terminale (vedi cap. 7) che può quindi rilasciare l'area di memoria e le altre risorse impegnate dal processo utente e prepararsi ad accettare il prossimo comando dell'utente.

Un altro tipo di controllo è quello relativo alla terminazione erronea del programma. Grazie ad esso, il sistema operativo è capace di riprendere il controllo anche in seguito terminazioni anomale di un processo utente. Le terminazioni anomale controllabili sono quelle che inducono la CPU a generare un apposito segnale di interruzione quando si verificano. In effetti, esistono specifici segnali di interruzione in grado di rilevare i seguenti eventi:

- esecuzione da parte della CPU di una istruzione erronea (codice operativo non valido o indirizzo erroneo)
- indirizzo non valido (il programma cerca di indirizzare un cella di RAM inesistente)
- diritti d'accesso insufficienti (il programma cerca di indirizzare un'area protetta senza avere i privilegi necessari)
- trabocco (overflow) numerico durante una operazione aritmetica

Quando il programma viene linkato in modo dinamico anzicché statico, il sistema operativo deve provvedere un altro supporto fondamentale: un *caricatore/linker dinamico* di programmi inclusi nelle librerie dinamiche. Il ruolo di tale supporto è di ritardare il linking dei programmi di libreria richiesti dal programma in esecuzione.

In alcuni casi, il traduttore prevede di collegare al programma utente un modulo di supporto all'esecuzione che effettua ulteriori controlli, oltre a quelli svolti dall'hardware del calcolatore. Il linguaggio Pascal, ad esempio, prevede un apposito modulo che svolge varie funzioni, tra cui quella filtrare ogni indirizzamento di variabili di tipo `array` per verificare se i valori degli indici rientrano nell'intervallo dichiarato. Nel caso opposto, il modulo interrompe l'esecuzione del programma e prepara un messaggio di diagnostica specificando il nome dell'array e i valori degli indici al momento dell'errore.

Un altro prezioso strumento per la messa a punto di programmi è il *debugger simbolico*. Tale funzione richiamabile tramite appositi parametri in fase di compilazione e collegamento, dà luogo al caricamento in memoria di un modulo aggiuntivo che interagisce con l'utente al terminale e serve a controllare

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

l'esecuzione del programma. Mediante tale modulo è possibile, ad esempio, richiedere l'esecuzione del programma fino ad arrivare ad una frase specifica del programma sorgente e leggere quindi il valore attuale di alcune variabili. In tale ottica, gli interpreti si possono considerare come i moduli di controllo più completi poiché essi controllano ogni singola frase del programma anziché un limitato sottoinsieme. Come è già stato osservato, il prezzo da pagare, però, per tale ricchezza di controlli è quello di avere tempi di esecuzione del programma interpretato molto più elevati di quelli del programma compilato.

3.4 Salvataggio/ripristino di dati

Anche se gli attuali file system sono alquanto affidabili, non si può escludere a priori il verificarsi di malfunzionamenti hardware dei dischi che potrebbero causare la perdita di tutte le informazioni in essi contenute. Per questo motivo, è consigliabile effettuare periodicamente il backup, ossia il salvataggio su supporti rimovibili del contenuto dei dischi del sistema. A tale scopo sono stati messi a punto programmi di utilità che consentono di svolgere tale attività in modo alquanto semplice. Le caratteristiche salienti di tali programmi sono le seguenti:

- salvataggio/ripristino su una serie di volumi rimovibili opportunamente etichettati mediante un apposito programma di utilità che prevede una apposita interfaccia per guidare l'utente nella fase di montaggio/smontaggio di volumi;
- possibilità di comprimere/decomprimere gli archivi riducendo così la quantità di memoria complessiva richiesta per il backup;
- possibilità di effettuare un backup selettivo basato sulla data dell'ultimo aggiornamento di ogni archivio. In tale modo vengono salvati periodicamente i soli archivi che sono stati modificati dopo l'ultimo backup effettuato.

3.5 Amministrazione del sistema

Il termine *amministrazione del sistema* sta ad indicare una serie di funzioni attinenti all'uso corretto delle risorse del sistema multiprogrammato da parte

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

degli utenti. In particolare, sono stati messi a punto programmi di utilità per risolvere i seguenti problemi:

- *identificazione dell'utente*: Oggi la soluzione più adottata consiste nell'identificare ogni utente tramite un nome utente (username) ed una parola chiave (password). L'utente deve, per potere accedere alle risorse del sistema, identificarsi e digitare la propria parola chiave. Solo dopo che l'identificazione è stata portata a termine con successo, l'utente può accedere al sistema. E' opportuno individuare un responsabile, l'amministratore del sistema, il quale è l'unico abilitato a creare nuovi utenti od a rimuovere utenti già esistenti.

- *conteggio delle risorse utilizzate e relativo addebito*: In molte aziende i costi del sistema informativo sono distribuiti tra i reparti che usufruiscono del servizio. E' quindi importante misurare l'uso da parte dei vari utenti delle principali risorse del sistema quali tempo di CPU, il prodotto del tempo per lo spazio su disco utilizzato, i volumi di stampa, ecc.. Le statistiche raccolte sull'uso delle risorse sono anche molto utili per individuare comportamenti anomali nonché eventuali colli di bottiglia. Potrebbe risultare, ad esempio, dalle statistiche di utilizzo della stampante veloce che essa causa un ritardo nella terminazione dei job per via delle code di stampa sempre piene.

- *autorizzazione alla condivisione delle risorse*: Risulta spesso necessario, nel caso di progetti che prevedono gruppi di programmatori ed utenti, prevedere una condivisione controllata delle cartelle e degli archivi di uso comune. Come nel caso della identificazione degli utenti, è opportuno concentrare nelle mani dell'amministratore del sistema anche le autorizzazioni o revoche di condivisione dell'informazione in modo da avere un responsabile unico per tale attività che può risultare alquanto critica.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

Capitolo 4

IL FILE SYSTEM

4.1 Introduzione

Le memorie di massa, quali i dischi e i nastri magnetici, hanno caratteristiche diverse da quelle della memoria RAM.

In primo luogo, sono di tipo non volatile e consentono quindi di conservare a basso costo e per un periodo prolungato di tempo informazioni degli utenti.

In secondo luogo, tali memorie hanno dei tempi di indirizzamento variabili che dipendono sia dall'indirizzo richiesto che dallo stato attuale del circuito di indirizzamento. Nel caso di un nastro magnetico, ad esempio, il tempo di indirizzamento dipenderà dalla distanza tra la porzione di nastro dove andrà eseguita l'operazione di I/O e quella attualmente posizionata dalla testina di lettura/scrittura. Nel caso di un disco a braccio mobile, invece, il tempo di indirizzamento dipenderà dalla distanza tra il cilindro contenente la traccia richiesta e quello attualmente posizionato dal braccio mobile dell'unità a disco. In conseguenza, diventa importante collocare l'informazione nelle memorie di massa in modo da ridurre, per quanto possibile, i tempi di indirizzamento.

In terzo luogo, infine, le informazioni contenute in tali memorie non sono direttamente accessibili alla CPU: a differenza della memoria principale, non esistono, ad esempio, istruzioni che consentono alla CPU di trasferire informazioni tra registri interni e aree di disco. La stessa funzione è invece ottenuta utilizzando processori specializzati chiamati *processori di I/O* ed eseguendo le seguenti istruzioni:

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

1. istruzioni per preparare i parametri (indirizzo del disco, indirizzo della traccia, indirizzo del settore, numero di byte da trasferire, indirizzo in memoria) da trasferire al processore di I/O al quale è collegato il disco. Nei calcolatori di tipo mainframe tali istruzioni prendono il nome di *programma di canale*;

2. istruzione di tipo `start i/o` per attivare il processore di I/O. Eseguita l'ultima istruzione, il programma si pone in attesa del segnale di interruzione di "fine I/O" da disco che indicherà il completamento dell'operazione.

Come si è appena visto, il reperimento e il trasferimento di informazioni contenute in memoria di massa risulta alquanto complesso. Nasce quindi l'esigenza di includere nei sistemi operativi moduli chiamati sistemi di archiviazione (in inglese, *file system*) che rendano più agevole l'accesso alle informazioni contenute in tali memorie.

4.2 Caratteristiche dei file system

Il file system svolge i seguenti compiti:

- semplificare gli accessi al disco: grazie al file system, il programmatore che intende leggere o scrivere dati su disco non deve specificare gli indirizzi fisici dei settori da trasferire; il disco appare invece come un contenitore di file e directory e sono previste apposite frasi nei linguaggi di programmazione che consentono di creare, modificare e cancellare file e directory;

- consentire al programmatore di organizzare i suoi dati servendosi di una delle strutture logiche di dati previste ed offrire per ognuna gli operatori necessari ad operare su di essa;

- gestire i vari dischi collegati al sistema, assegnando o rilasciando aree di memoria secondaria in base alle richieste dei programmi che utilizzano il file system.

Le cartelle (in inglese, *directory*) sono state introdotte per consentire agli utenti di raggruppare file tra loro affini nella stessa directory. Suddividendo

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

i numerosi file contenuti in un disco in cartelle diverse le quali, a loro volta, possono includere altre cartelle oltre che file, risulta semplificato il reperimento dei singoli file e, come si vedrà più avanti, risulta possibile fare uso di schemi di protezione differenziati.

4.3 Protezione delle informazioni

Oltre a conservare nel tempo informazioni raggruppate in file, i file system offrono in generale alcuni schemi di protezione con finalità diverse. Accenniamo soltanto, per motivi di spazio, a due di essi:

- gestione automatica di copie di riserva o backup. Ogni volta che viene modificato un file, il file system crea automaticamente un file contenente la versione precedente. Questo consente all'utente di non perdere i suoi dati nel caso di operazioni erronee. In alcuni file system, anziché una sola copia di backup viene creata una serie storica delle versioni precedenti indicizzata, appunto, sul numero di versione (versione 1 per il file appena creato, 2 per la prima modifica, ecc.). In tali file system viene lasciato all'utente il compito di cancellare le versioni considerate superflue.

- accesso riservato a cartelle o file. In un sistema multiprogrammato, i file di utenti diversi sono registrati in dischi comuni ed è quindi necessario garantire uno schema di protezione che impedisca ad utenti non autorizzati di accedere ai file di altri utenti. A tale scopo, si rivela molto utile il concetto di cartella poiché in tale modo risulta alquanto agevole raggruppare tutti i file di un utente in una stessa cartella facendo "vedere" all'utente la porzione di disco corrispondente alla sua cartella. In pratica, un simile approccio deve essere in qualche modo mitigato per consentire la condivisione controllata di informazioni.

4.4 Il filesystem di Unix

L'intero sistema operativo Unix è basato su un file system che rimane tuttora valido, pur essendo stato introdotto un quarto di secolo fa. È opportuno quindi iniziare la descrizione di Unix partendo da tale file system.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

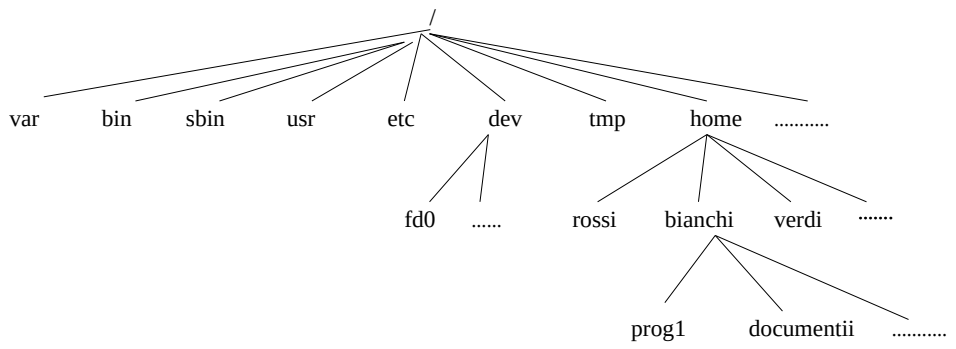

Figura 4.1: Collocazione dei file nello spazio dei nomi.

4.4.1 File

Un file Unix è un contenitore di informazioni strutturato come una sequenza di byte. Il NUCLEO Unix non interpreta il contenuto di un file. Sono disponibili diverse librerie di programmi che realizzano astrazioni ad un livello più elevato: facendo uso dei programmi inclusi in tali librerie, è possibile ad esempio considerare un file come composto da una sequenza di record, ognuno dei quali è suddiviso in campi; è inoltre possibile caratterizzare uno o più campi di un record come campi chiave ed effettuare indirizzamenti veloci basati sull'uso di campi chiave. Tutti i programmi inclusi in tali librerie fanno uso tuttavia delle chiamate di sistema standard del NUCLEO Unix.

Dal punto di vista dell'utente, i file sono collocati in uno spazio dei nomi organizzato ad albero come indicato nella Figura 4.1.

Tutti i nodi dell'albero, tranne i nodi foglia, denotano nomi di cartelle. Ogni nodo di tipo cartella specifica quali nomi di file e quali nomi di cartelle sono contenuti in esso.

Un nome di file o directory consiste in una sequenza di caratteri ASCII, con l'eccezione del carattere '/' e del carattere nullo '\0' usato come terminatore di stringhe. Ogni file system Unix pone un limite alla lunghezza massima di un nome di file, tipicamente 255 caratteri.

La cartella associata alla radice dell'albero è chiamata cartella radice (in inglese, *root directory*). Per convenzione, il suo nome è rappresentato col carattere '/'. I nomi di file all'interno della stessa cartella devono essere

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

distinti tra loro, anche se lo stesso nome di file può essere usato in cartelle diverse.

Unix associa una cartella attiva (in inglese, *current working directory*) ad ogni processo; essa fa parte del contesto di esecuzione del processo ed identifica la cartella attualmente usata dal processo.

Per identificare un file, il programmatore fa uso di un nome di percorso (in inglese, *pathname*) che consiste in una alternanza di caratteri '/' e di nomi di cartella. L'ultimo componente del nome di percorso può essere il nome di una cartella, oppure il nome di un file, a seconda che l'oggetto da identificare sia una cartella oppure un file.

Se il primo componente di un nome di percorso è uno '/', allora il nome di percorso è di tipo assoluto, poiché la ricerca deve iniziare a partire dalla cartella radice. Nel caso opposto il nome di percorso è relativo poiché la ricerca inizia dalla cartella attiva del processo in esecuzione.

Nel costruire nomi di percorso, sono anche usati i simboli '.' e '..'. Essi indicano, rispettivamente, la cartella attiva e la cartella che contiene la cartella attiva. Se la cartella attiva è la cartella radice, il significato dei simboli '.' e '..' coincide.

4.4.2 Link hard e soft

Ogni nome di file incluso in una cartella è chiamato *hard link* o più semplicemente *link*. Lo stesso file può avere più link inclusi nella stessa cartella, o in cartelle diverse, e quindi può avere più nomi.

Il comando Unix `ln f1 f2` è usato per creare un nuovo hard link avente nome di percorso `f2` relativo al file avente nome di percorso `f1`.

Gli hard link hanno due limiti.

- non è possibile creare un hard link relativo ad una cartella poiché questo potrebbe trasformare l'albero delle cartelle in un reticolo con cicli, rendendo con ciò impossibile la identificazione di un file in base al suo nome di percorso;
- un nuovo hard link `f2` può essere creato soltanto nel file system che contiene il file `f1`; questo è un limite in quanto i sistemi Unix più recenti consentono di installare più file system di tipo diverso in più dischi o partizioni.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

I suddetti limiti sono superati facendo uso di *soft link* (sono anche chiamati *link simbolici*). Essi consistono in appositi file di dimensione ridotta che contengono il nome di percorso di un file. Tale nome di percorso è arbitrario; esso può identificare un file inserito in un qualsiasi file system, o perfino un file inesistente.

Il comando Unix `ln -s f1 f2` è usato per creare un nuovo soft link avente nome di percorso `f2` relativo al file avente nome di percorso `f1`. Quando questo comando viene eseguito, il file system crea un file di tipo link simbolico (vedi prossimo paragrafo) e scrive in esso il nome di percorso `f1`; inserisce quindi nella cartella opportuna una nuova voce contenente l'ultimo nome del nome di percorso `f2`. In questo modo, diventa possibile tradurre automaticamente ogni riferimento ad `f2` in un riferimento ad `f1`.

4.4.3 Tipi di file

I file Unix sono divisi nelle seguenti classi:

- file regolari;
- cartelle;
- link simbolici;
- device file orientati a blocchi;
- device file orientati a caratteri;
- pipe con nome o FIFO;
- socket

I primi tre tipi di file sono i componenti del file system Unix: l'informazione (dati o programmi) è contenuta nei file regolari; l'uso di cartelle consente di realizzare uno spazio dei nomi di file organizzato ad albero, e quindi di fare uso di nomi di percorso; i link simbolici, infine, offrono una maggiore flessibilità di gestione.

I device file identificano dispositivi di ingresso/uscita quali dischi, tastiera o mouse. Essi sono stati introdotti da Unix per realizzare uno standard di programmazione C in base al quale le stesse chiamate di sistema `open()`,

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

`close()`, `read()` e `write()` devono potere essere utilizzate per indirizzare dati contenuti vuoi in un file tradizionale che in un qualsiasi dispositivo di ingresso/uscita.

Le pipe con nome o FIFO sono usate come un meccanismo veloce per la comunicazione tra processi. In pratica, il file di tipo pipe serve soltanto a memorizzare il nome della pipe, mentre lo scambio di dati tra processi avviene in RAM senza fare uso del file system.

I socket sono utilizzati sia come strumento di sincronizzazione locale che per la connessione in rete.

4.4.4 Descrittore di file ed inode

Unix effettua una distinzione netta tra file e descrittore di file. In altre parole, tutti i tipi di file tranne i device file e le FIFO sono considerati come contenitori di sequenze, eventualmente nulle, di caratteri. In altre parole, un file non include al suo interno alcuna informazione che ne descrive la tipologia o la struttura, quale ad esempio, la lunghezza del file oppure un delimitatore di tipo End-Of-File (EOF) che segnala la fine del file stesso.

Tutte le informazioni necessarie al file system per gestire un file sono contenute in una struttura di dati chiamata *inode*. Ogni file possiede un proprio inode ed il file system usa il numero dell'inode come identificatore del file. Esistono svariate realizzazioni di file system per Unix che differiscono in modo notevole una dall'altra. In ogni caso, ognuna di esse deve rispettare lo standard POSIX che richiede la presenza dei seguenti attributi tra quelli che caratterizzano un inode:

- tipo di file type (vedi sopra);
- numero di hard link associati al file;
- lunghezza del file in byte;
- identificatore del dispositivo che include il file;
- numero dell'inode associato al file;
- User ID dell'utente possessore del file;
- Group ID del file;

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

- alcune date e ore che specificano, ad esempio, quando l'inode è stato indirizzato per ultimo e quando è stato modificato per ultimo;
- i diritti d'accesso ed il "file mode" (vedi appresso).

4.4.5 Diritti d'accesso e file mode

I potenziali utenti di un file sono divisi in tre classi:

- l'utente possessore del file: quando un file è creato da un programma, lo User ID dell'utente per conto del quale tale programma viene eseguito viene usato per caratterizzare il possessore del file;
- gli utenti che appartengono allo stesso gruppo (che hanno lo stesso stesso group ID) dell'utente possessore del file;
- i rimanenti utenti (others).

Tre diversi tipi di diritti d'accesso chiamati Read (diritto di lettura), Write (diritto di scrittura) ed Execute (diritto di esecuzione del codice) sono previsti per ognuna di queste tre classi. L'insieme di diritti d'accesso associati ad un file consiste quindi in nove indicatori (flag) binari.

Altri tre indicatori chiamati suid (SetUser ID), sgid (Set Group ID) e sticky definiscono il file mode. Descriviamo il loro significato nel caso in cui il file sia un file contenente un programma eseguibile:

- un processo che esegue un file conserva solitamente lo User ID dell'utente possessore del processo; se però il file eseguibile ha il flag suid impostato ad uno, allora il processo riceve lo User ID del possessore del file;
- un processo che esegue un file conserva solitamente il Group ID associato all'utente possessore del processo; se però il file eseguibile ha il flag sgid impostato ad uno, allora il processo riceve il Group ID associato all'utente possessore del file;
- se il file eseguibile ha il flag sticky impostato ad uno, il NUCLEO deve cercare di mantenere in memoria il programma anche dopo che è stato seguito (questo flag non è più molto usato poiché esistono tecniche di gestione della memoria basate sul Demand paging che lo rendono superfluo).

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

4.4.6 Chiamate di sistema per la gestione di file

Quando un utente indirizza il contenuto di un regular file, oppure quello di una cartella, egli indirizza in pratica dei dati registrati in un block device, ossia in un dispositivo di ingresso/uscita di tipo disco. In questo senso, un file system può essere considerato come una astrazione a livello utente della organizzazione fisica di una qualche partizione di disco. Poiché l'utente non è abilitato, per motivi di sicurezza, ad interagire direttamente col disco, ogni operazione attinente ad un file deve essere svolta in Kernel Mode.

Per questo motivo, Unix prevede un insieme di chiamate di sistema per la gestione dei file; quando un processo intende interagire con uno specifico file, esso invoca la chiamata di sistema opportuna passando il nome di percorso del file come parametro. Esaminiamo brevemente tali chiamate di sistema.

4.4.7 Apertura di un file

Per potere indirizzare un file, esso deve essere stato "aperto" in precedenza. A tale scopo, il processo esegue la seguente chiamata di sistema:

```
fd = open(path, flag, mode)
```

I tre parametri usati hanno il seguente significato:

- *path*: Nome del percorso (assoluto o relativo) del file da aprire.
- *flag*: Specifica come il file deve essere aperto: solo lettura, solo scrittura, lettura/scrittura, scrittura in fondo al file (append); specifica anche, nel caso in cui non esista alcun file avente nome di percorso path, se esso deve essere automaticamente creato.
- *mode*: Specifica i diritti d'accesso (nel solo caso in cui il file venga creato).

Il NUCLEO gestisce tale chiamata di sistema creando un oggetto "file aperto" e restituendo al processo un identificatore chiamato descrittore di file. Un oggetto "file aperto" contiene:

- alcune strutture di dati per la gestione del file, quali un puntatore all'area di memoria in cui verranno copiati i caratteri del file, un campo offset che indica l'indice all'interno del file dell'ultimo carattere indirizzato e così via;

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

- alcuni puntatori a funzioni (i metodi dell'oggetto) che il processo è autorizzato ad invocare; tale insieme dipende dal valore del parametro flag.

Diamo ora alcune ulteriori informazioni relative all'interazione tra file e processo che si evincono dalla semantica POSIX.

Un descrittore di file rappresenta l'interazione tra un processo ed un file aperto, mentre un oggetto "file aperto" contiene dati relativi a tale interazione.

Lo stesso oggetto "file aperto" può essere identificato da più descrittori di file. Più processi possono aprire lo stesso file in modo concorrente: in tale caso, il file system assegna ad ognuno di essi un apposito descrittore di file ed un apposito oggetto "file aperto". Quando ciò si verifica, il file system di Unix non tenta in alcun modo di sincronizzazione le diverse operazioni di ingresso/uscita eseguite da più processi sullo stesso file. È comunque disponibile la chiamata di sistema `flock()` che consente a più processi di sincronizzarsi sull'intero file o su una porzione di esso. Per creare un nuovo file, è anche possibile fare uso della chiamata di sistema `create()` che è gestita dal NUCLEO alla stregua di una `open()`.

4.4.8 Indirizzamento di un file

Dopo averlo aperto in precedenza, è possibile indirizzare un file regolare Unix in modo sequenziale oppure casuale (random); i device file e le FIFO sono solitamente indirizzati in modo sequenziale. In entrambi i tipi di accesso, il NUCLEO aggiorna il valore del campo offset dell'oggetto "file aperto" in modo che esso punti al prossimo carattere da leggere o da sovrascrivere all'interno del file.

Il file system assume di norma che il nuovo indirizzamento è sequenziale rispetto a quello effettuato per ultimo: le chiamate di sistema `read()` e `write()` fanno sempre riferimento all'attuale valore del campo offset dell'oggetto "file aperto". Per modificare tale valore, è necessario invocare in modo esplicito la chiamata di sistema `lseek()`.

Quando un file viene aperto, il NUCLEO imposta il campo offset alla posizione del primo byte nel file, ossia pone offset uguale a 0. La chiamata di sistema `lseek()` opera sui seguenti parametri:

```
newoffset = lseek(fd, offset, whence);
```


scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

che hanno i seguenti significati:

- *`fd`*: Descrittore di file del file aperto;
- *`offset`*: Specifica un intero usato per calcolare il nuovo valore di offset;
- *`whence`*: Specifica se `newoffset`, il nuovo valore del campo offset, deve essere calcolato come il valore di offset (offset dall'inizio del file), la somma di offset più l'attuale valore del campo offset (offset dalla posizione attuale), la differenza tra l'indice dell'ultimo byte nel file e offset (offset dalla fine del file).

La chiamata di sistema `read()` fa uso dei seguenti parametri:

```
nread = read(fd, buf, count);
```

che hanno i seguenti significati:

- *fd*: Descrittore di file del file aperto che si intende leggere;
- *buf*: Indirizzo dell'area di memoria nello spazio degli indirizzi del processo dove trasferire i dati letti;
- *count*: Numero di byte da leggere.

Nel gestire tale chiamata di sistema il NUCLEO cerca di leggere count byte dal file identificato da fd, a partire dal byte individuato dal campo offset nel descrittore dell'oggetto "file aperto". In alcuni casi (fine del file, pipe vuota, ecc.), il NUCLEO non riesce a leggere tutti i byte richiesti.

Il valore `nread` restituito dalla chiamata di sistema specifica il numero di byte effettivamente letti. Il nuovo valore del campo offset è ottenuto sommando il valore precedente a `nread`. I parametri della chiamata di sistema `write()` sono simili a quelli della `read()`.

Il NUCLEO esegue le chiamate di sistema `read()`, `write()` e `lseek()` in modo atomico: due chiamate di sistema non possono essere eseguite simultaneamente sullo stesso file.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

4.4.9 Chiusura di un file

Quando un processo non ha più necessità di indirizzare un file, esso invoca la chiamata di sistema:

```
res = close(fd);
```

che rilascia l'oggetto "file aperto" relativo al descrittore di file `fd`. Quando un processo termina, il NUCLEO chiude tutti i file ancora aperti del processo.

4.4.10 Cambiamento di nome e cancellazione di file

Per potere cambiare nome oppure cancellare un file, un processo non deve aprire il file. In effetti, tali operazioni non agiscono sul contenuto del file bensì sul contenuto di una o più cartelle dove inserire o rimuovere il nome del file. La chiamata di sistema:

```
res = rename(oldpath, newpath);
```

cambia il nome di un link al file, mentre la chiamata di sistema:

```
res = unlink(pathname);
```

decrementa di uno il contatore di link al file. Il file viene cancellato dal file system quando tale contatore assume il valore 0.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

Capitolo 5

INTERFACCIA CON L'UTENTE

5.1 Introduzione

In un sistema operativo dedicato, ogni utente collegato al sistema deve essere in grado, servendosi dei diversi dispositivi di ingresso previsti, di specificare in modo agevole quale tra i vari servizi riconosciuti ed eseguibili dal sistema egli intende ottenere. In conseguenza, il sistema operativo deve includere appositi programmi per riconoscere i segnali di ingresso inviati dall'utente; tali programmi costituiscono complessivamente una *interfaccia* tra l'utente da una parte, e il resto del sistema operativo e l'hardware dall'altra.

Le interfacce dei sistemi operativi possono avere caratteristiche alquanto diverse, anche se il loro compito rimane sempre quello di identificare la richiesta effettuata dall'utente nonché gli eventuali parametri ad essa associati. Si possono distinguere tre approcci possibili basati, rispettivamente, sull'uso di menu, comandi ed icone.

5.2 Interfaccia a menu

Nelle interfacce basate sull'uso di *menu*, l'utente non deve ricordare i nomi dei diversi programmi che intende eseguire poiché ogni menu che appare sullo schermo include un elenco di funzioni (o classi di funzioni) eseguibili e, per ognuna di esse, è fornita una breve descrizione nonché il nome del tasto da premere per richiederne l'esecuzione. In generale, data la limitata

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

capacità dello schermo, è necessario ricorrere a una gerarchia di menu per potere descrivere tutte le funzioni offerte dal sistema operativo. Il pregio di tali interfacce, tuttora utilizzate in sistemi transazionali ed all'interno di vari programmi applicativi, sta nella loro semplicità: in primo luogo, esse non richiedono l'uso di un terminale grafico ma possono operare anche su semplici terminali alfanumerici. In secondo luogo, i programmi di gestione di tali interfacce sono alquanto semplici e compatti. Le interfacce a menu si rivelano tuttavia troppo rudimentali per potere consentire ad un utente di interagire con un moderno sistema operativo.

5.3 Interfaccia a comandi

In tale interfaccia l'utente interagisce col sistema inviando ad essi dei comandi. Ogni *comando* è costituito da una sequenza di caratteri alfanumerici seguita dal tasto di <INVIO>. I comandi hanno una loro sintassi ben definita che è usata da un modulo del sistema operativo chiamato *interprete di comandi*; tale modulo analizza ogni comando ed esegue le azioni richieste. Tipicamente il comando inizia col nome seguito da una serie di parametri ed opzioni.

In realtà, è più corretto parlare negli attuali sistemi operativi di *linguaggi di comandi* anziché di semplici comandi. In effetti, è possibile creare file contenenti sequenze di comandi: quando l'interprete di comandi riceve il nome di tale file come programma da eseguire, passa ad interpretare i vari comandi contenuti nel file.

I linguaggi di comandi prevedono l'uso di variabili locali, variabili di sistema e frasi condizionali, per cui rappresentano, in diversi casi, una valida alternativa ai linguaggi compilati. I programmi scritti in un linguaggio di comandi prendono il nome di *script*.

Al solito, la scelta se realizzare una applicazione facendo uso di uno script oppure di un programma compilato (ad esempio, dal compilatore C) dipende dai tempi di risposta richiesti: l'esecuzione di un script risulta ovviamente più lenta poiché ogni comando è interpretato dall'interprete di comandi. D'altra parte, i sistemi operativi includono solitamente biblioteche molto ampie di comandi già predisposti che possono essere opportunamente combinati in uno script, per cui il tempo di messa a punto dell'applicazione risulta molto inferiore quando si ricorre ad uno script.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

5.3.1 Lo shell di Unix

L'interfaccia tra l'utente ed il sistema operativo Unix è una interfaccia a comandi. I comandi Unix sono alquanto potenti per due motivi diversi.

- Non esiste un insieme predefinito di comandi: ogni programma eseguibile o interpretabile tramite uno degli interpreti inclusi nel sistema operativo è considerato come un comando. L'utente può quindi creare comandi personalizzati e richiederne l'esecuzione semplicemente digitando il nome del nuovo comando seguito dagli opportuni parametri e premendo il tasto di <INVIO>.

- Unix consente di creare comandi composti da comandi più semplici. A tale scopo, Unix include un apposito linguaggio di comandi che si distingue da altri simili linguaggi per la sua ricchezza e flessibilità. Esso è un vero e proprio linguaggio di programmazione dotato di frasi di controllo e chiamate di procedura. A differenza dei linguaggi di programmazione convenzionali quali il C oppure il FORTRAN, i comandi scritti in tale linguaggio non devono essere compilati da un apposito compilatore ma vengono immediatamente letti, interpretati ed eseguiti da un interprete di comandi chiamato *shell* (guscio). Esistono diversi shell (C shell, Bourne shell, bash shell, ecc.) che differiscono leggermente l'uno dall'altro. Viene inoltre offerto all'utente la possibilità di definire il proprio shell da usare in applicazioni particolari.

Vediamo ora quali sono le caratteristiche più interessanti di tale linguaggio di comandi che, secondo una consuetudine invalsa, chiameremo shell.

Un *comando Unix* è un frase composta dal nome del comando seguito eventualmente da parametri separati da uno spazio. Dopo aver digitato il comando, è necessario digitare il tasto di <INVIO> per indicare il completamento del comando stesso. Nell'eseguire, ad esempio, il comando:

```
cp   file1 file2
```

lo shell effettua una copia del file chiamato `file1` ed assegna a tale copia il nome `file2` (se esiste già un altro file avente tale nome, esso viene distrutto e sostituito dal nuovo).

I caratteri immessi da tastiera e quelli visualizzati sullo schermo del terminale sono considerati dallo shell come due file speciali chiamati, rispettivamente,

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

`input` e `output`. In assenza di altre informazioni, lo shell assume che il file `stdin` (file di ingresso) sia associato alla tastiera e quello `stdout` (file di uscita) sia associato al video. Nell'eseguire, ad esempio, il comando:

```
date
```

lo shell visualizza la data e l'ora attuale sul video.

La ridefinizione dell'input e dell'ouput è una delle varie caratteristiche che rendono lo shell molto flessibile. Usando gli operatori ">" per ridefinire l'output e "<" per ridefinire l'input, è possibile specificare file di ingresso e di uscita diversi da quelli standard. Il comando:

```
date > file1
```

scrive, ad esempio la data e l'ora attuale nel file `file1` anziché sul video.

Servendosi di tale linguaggio, è possibile specificare mediante frasi di controllo del tipo `if then else` e `while`, l'ordine mediante il quale eseguire la sequenza di comandi contenuta nel programma e controllare tale esecuzione in funzione dei parametri ricevuti e delle condizioni di terminazione dei vari comandi eseguiti.

Daremo diversi esempi di script Unix in apposite esercitazioni. Suggeriamo comunque allo studente di provare il maggior numero possibile di comandi tra quelli documentati tramite il comando `man`.

5.3.2 Struttura dello shell

L'interprete di comandi o shell è un programma memorizzato in un apposito file eseguibile del file system. Durante l'inizializzazione del sistema, Unix lancia uno shell per ognuno dei terminali installati; ognuno di tali shell visualizza un messaggio di "login" sul proprio terminale e rimane in attesa fino a quando l'utente non inizia una sessione comunicando il suo nome e la relativa parola d'ordine.

Se il login risulta corretto, lo shell responsabile del terminale crea uno shell secondario per gestire le richieste del nuovo utente e si pone in attesa della terminazione (o del passaggio allo stato di background) dello shell appena creato. Quando l'utente al terminale conclude, mediante il comando `logout`,

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

la sessione di lavoro, lo shell elimina tutti gli eventuali shell secondari creati durante la sessione.

La distinzione tra shell principale e shell secondari può risultare oscura a prima vista: perché non fare interpretare direttamente dallo shell principale i comandi digitati dall'utente anziché lanciare shell secondari? La risposta è semplice: la gerarchia di shell consente una flessibilità operativa molto maggiore agli utenti più smaliziati. Diamo un paio di esempi per giustificare questa affermazione:

- facendo seguire ad un comando il carattere '&', il comando diventa un comando di "background" e lo shell ridiventa immediatamente attivo senza aspettare la terminazione del comando di background lanciato. L'utente può, ad esempio, lanciare una stampa che richiede un tempo notevole tramite un comando di backgroud e continuare ad usare il terminale mentre la stampa è in atto;

- in alcuni sistemi Unix, tra cui Linux, è possibile creare più console e commutare da una all'altra. In Linux, digitando <CTRL><ALT>F2, ..., <CTRL><ALT>F7, si possono lanciare altre console oltre a quella iniziale che condividono tutte lo stesso monitor.

Vediamo ora come lo shell associa al nome di un comando il file contenente il programma o lo script da eseguire. In teoria, si potrebbe richiedere di digitare ogni volta l'intero pathname ma tale soluzione non sarebbe gradita dall'utente in quanto lo costringerebbe a digitare lunghe sequenze di caratteri, ad esempio `/usr/bin/man` anziché `man`. Il problema viene risolto introducendo una apposita *variabile shell* chiamata `PATH` che contiene la sequenza delle directory, separate dal carattere ':' dove cercare il file avente un nome prescelto.

Il contenuto di `PATH` può essere personalizzato inserendo una nuova definizione, diversa da quella di default, nel file `.profile` contenuto nella home directory dell'utente.

Possiamo usare il comando:

```
echo $PATH
```

per visualizzare sullo schermo il valore della variabile `PATH`, ossia i nomi di tutte le directory che verranno esaminate dallo shell per cercare un file il cui

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

nome sia uguale a quello del comando. In risposta a tale comando, appare sul monitor una sequenza di pathname del tipo:

```
/usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin:/sbin:.
```

Se il comando digitato dall'utente non può essere eseguito, lo shell segnala una condizione di errore. Questo avviene per due possibili motivi:

- il file system non contiene un file avente il nome del comando;
- il file avente il nome del comando non è né un file eseguibile né un file di comandi per il quale sia disponibile un opportuno interprete (tale tipo di file è chiamato *comando indiretto*).

Lo shell è in grado di distinguere un file eseguibile da un file contenente comandi da interpretare, ossia un comando da un comando indiretto, leggendo i primi byte del file: essi contengono un *numero magico*, ossia un identificatore che specifica qual è il programma richiesto per eseguire il file (caricatore o specifico interprete). Non esiste infatti un solo interprete: esistono diversi interpreti per gli shell Unix ed esistono inoltre interpreti del tutto diversi per linguaggi interpretati quali Java.

Come è ovvio, se il file contiene un numero magico ignoto al sistema operativo, lo shell ritorna un messaggio di errore del tipo:

```
command not found
```

Se invece lo shell lanciato per eseguire il comando va in esecuzione, lo shell che lo ha lanciato si mette in attesa della sua terminazione. Il suddetto schema è ricorsivo: se uno dei comandi è a sua volta un comando indiretto, un terzo shell passa a interpretarlo e cosí via.

Concludiamo questa presentazione sommaria degli shell Unix con una osservazione importante. Oltre ad essere digitati da utenti al terminale, i comandi possono essere lanciati da un qualsiasi programma in esecuzione tramite una chiamata di sistema del tipo `execve()` (vedi Capitolo 8). È quindi possibile scrivere un programma che simula la sequenza di azioni svolte da un utente al terminale.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

5.4 Interfacce grafiche

Le interfacce basate sull'uso di icone sono state introdotte per rispondere a particolari esigenze di facilità d'uso e di apprendimento e sono quindi rivolte in primo luogo ad utenti non necessariamente esperti. A differenza dei due tipi di interfacce descritte in precedenza che richiedono soltanto un terminale alfanumerico ed una tastiera, le interfacce grafiche richiedono invece l'uso di un terminale grafico, di una tastiera e di un apposito dispositivo d'ingresso chiamato mouse mediante il quale è possibile spostare il cursore grafico sullo schermo ed inviare segnali al sistema premendo uno dei pulsanti collocati in cima ad esso. L'idea di base è quella di offrire all'utente un menu grafico composto da ideogrammi chiamati *icone*.

Il primo sistema operativo commerciale con interfaccia grafica è stato messo a punto per i personal computer Macintosh della Apple negli anni '80. In tale sistema, il menu iniziale includeva una serie di icone corrispondenti ai file dell'utente, a quelli contenenti programmi del sistema operativo e ad altri file di servizio come la "cartella vuota" e il "cestino". Per cancellare, ad esempio, un file, ad esempio il file di nome A, in tale sistema è sufficiente:

1. posizionare il cursore dentro all'icona A (questo si ottiene spostando il mouse sulla scrivania finché la freccia del cursore non sia contenuta nell'icona);

2. spostare, tenendo premuto il pulsante del mouse, l'icona A in quella denominata "cestino" rilasciando quindi il pulsante.

Per aprire un documento gestito da una apposita applicazione, è sufficiente effettuare un doppio clic sull'icona corrispondente.

In alcuni casi, però, l'uso della tastiera rimane fondamentale: ogni qualvolta si intende creare una nuova cartella, oppure cambiare il nome di un file, è necessario trasmettere l'informazione al file system facendo uso della tastiera. Oggi le interfacce grafiche sono diventate uno standard per i sistemi operativi dei personal computer e sono molto diffuse nei sistemi operativi MacOs della Apple, in quelli di tipo Unix ed in quelli della famiglia Microsoft Windows quali Windows 98 o Windows 2000.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

5.4.1 Interfacce grafiche per Unix

A differenza di altri sistemi operativi quali MacOS della Apple, oppure Windows NT della Microsoft, che incorporano l'interfaccia grafica all'interno del NUCLEO, l'interfaccia grafica per Unix è sempre realizzata tramite una applicazione di tipo client/server. In effetti, Unix è nato prima che esistessero i terminali grafici necessari per realizzare tale tipo di interfaccia.

Il più noto programma applicativo Unix che realizza una interfaccia grafica si chiama X windows. Esso è stato sviluppato inizialmente presso lo MIT ed è oggi lo standard de facto di interfaccia grafica per sistemi Unix. Sono stati sviluppati numerosi client specializzati per X windows, ad esempio per emulare terminali alfanumerici (`xterm`), per visualizzare documentazione Unix all'interno di una finestra (`xman`), per visualizzare in forma grafica la struttura del file system e delle sue cartelle (`xfm`), per rappresentare e collocare finestre (`fvwm`, ecc.), ecc.

5.5 Interfaccia grafica o a comandi?

Le interfacce a comandi continuano ad essere le interfacce preferite dai programmatori professionisti per la loro estrema flessibilità d'uso, anche se possono risultare poco gradite ad un utente occasionale il quale non realizza perché sia necessario fare uso di comandi alquanto complessi per svolgere semplici funzioni.

L'evoluzione di Windows NT è sigificativa a tale rispetto: le prime versioni di Windows NT offrivano soltanto una finestra terminale alfanumerico di tipo "emulatore MS-DOS", ossia lo shell di MS-DOS che è alquanto rudimentale e non all'altezza del resto del sistema operativo.

Le versioni più recenti di Windows NT introducono un gestore di linguaggi di comandi chiamato Window Scripting Host; esso è in grado di controllare l'esecuzione in modalità interpretativa di script scritti nel linguaggio Visual Basic, oppure in quello Java Script. Window Scripting Host consente all'utente di eseguire script in due modi possibili: direttamente dal desktop cliccando sull'icona che rappresenta lo script, oppure digitando una riga di comando nell'apposita finestra alfanumerica chiamata "command console".

È importante osservare, inoltre, come Windows NT consenta di eseguire script in modi diversi. Ciò può essere fatto:

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

- tramite Internet Explorer richiedendo ad una macchina client di eseguire uno script contenuto all'interno di una pagina html;
- tramite Internet Information Server il quale è in grado di gestire pagine del tipo "Active Server Page"; in altre parole, consente al server di inviare script ai client su Internet.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

Capitolo 6

NUCLEO E PROCESSI

6.1 Introduzione

Pur tenendo conto della continua diminuzione del costo dell'hardware, i processori (circuiti in grado di eseguire istruzioni programmabili registrate in memoria RAM) continuano ad essere considerati risorse hardware pregiate il cui uso deve essere ottimizzato. Per questo motivo, gli attuali sistemi operativi sfruttano il potenziale parallelismo hardware dell'elaboratore avviando e controllando l'esecuzione di più programmi o sequenze di istruzioni sui processori disponibili. Viene in tale modo aumentata la produttività o *throughput* del sistema, ossia il numero di programmi eseguiti per unità di tempo. Per conseguire tale risultato, è necessario fare uso di tecniche di gestione alquanto sofisticate.

L'idea di base comune a tali sistemi è quella di caricare simultaneamente in memoria più programmi e di eseguirne alcuni in parallelo, utilizzando i diversi processori a disposizione. Tale impostazione è nota come *multitasking*.

È interessante osservare come sia possibile realizzare sistemi multitasking con ottime prestazioni, anche per calcolatori dotati di due soli processori: una CPU e un processore di I/O. In effetti, l'esecuzione di un programma è costituita da una successione di fasi di elaborazione sulla CPU e fasi di attesa per il completamento di operazioni di I/O che impegnano il processore di I/O lasciando logicamente inattiva la CPU. Durante tali fasi di attesa, il sistema operativo multitasking può mandare in esecuzione sulla CPU altri programmi tra quelli caricati in memoria, migliorando così la produttività del sistema.

La Figura 6.1 illustra in modo semplificato l'aumento di produttività derivan-

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

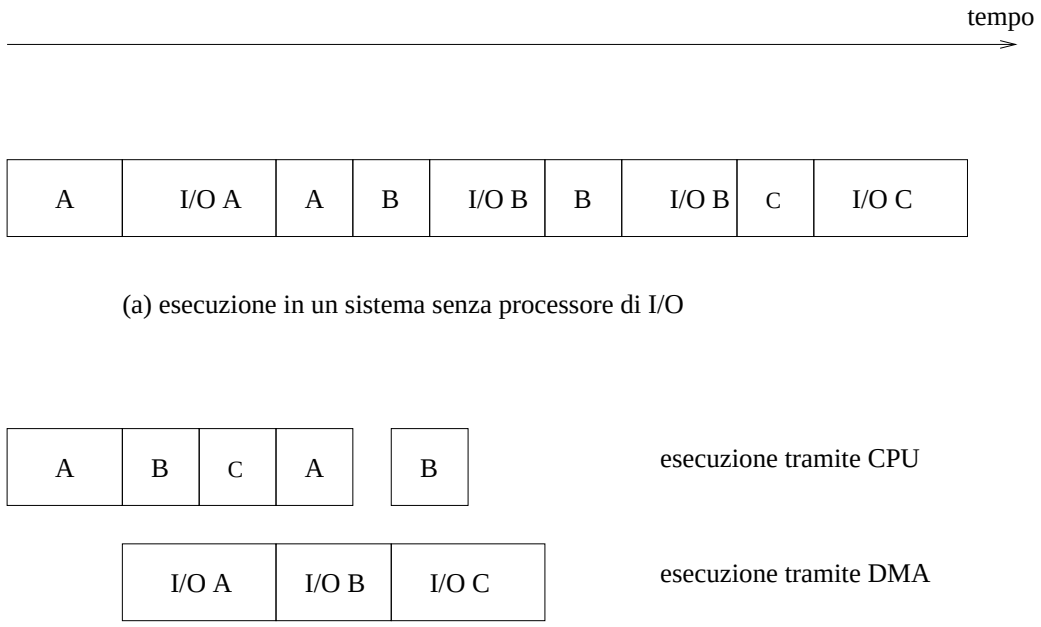

Figura 6.1: Un esempio semplificato di sistema multitasking.

te dalla esecuzione concorrente di tre programmi, chiamati A, B e C , rispetto alla esecuzione sequenziale in ambiente uniprogrammato.

Come si può osservare dalla figura, l'esecuzione multitasking di più programmi distinti consente, in generale, di aumentare sensibilmente la produttività del sistema, ossia il numero di programmi eseguiti per unità di tempo. Oggi la gestione multitasking è utilizzata in tutti i tipi di sistemi operativi (gestione a lotti, sistemi interattivi, sistemi transazionali, controllo di processi). Vedremo in un capitolo successivo come i diversi obiettivi che tali sistemi devono conseguire si rifletta in diverse tecniche di gestione dei processori.

Da un punto di vista funzionale, si distinguono i seguenti tipi di parallelismo:

- *multitasking*: esecuzione di programmi indipendenti sulla CPU e sul processore di I/O;
- *multiprogrammazione*: multitasking con l'aggiunta di tecniche di protezione della memoria che assicurano che un programma in esecuzione

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

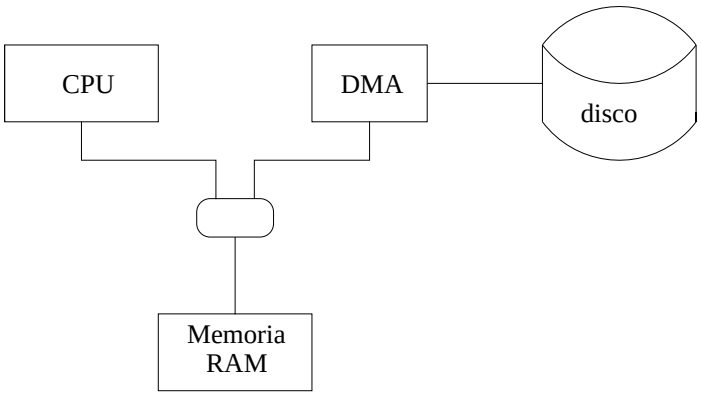

Figura 6.2: Architettura di un sistema concorrente.

non possa accedere alle aree di memoria assegnate agli altri programmi eseguiti insieme ad esso;

- *multiprocessing* multiprogrammazione estesa ad elaboratori dotati di più CPU e processori di I/O.

Si noti come il multitasking non implichi la multi-utenza, ossia l'uso simultaneo del sistema da parte di più utenti. In effetti, vari sistemi operativi messi a punto per personal computer sono sistemi mono-utenti, ossia sistemi che prevedono l'uso del sistema da parte di un solo utente alla volta.

Ciò nonostante, diversi noti sistemi operativi per personal computer quali MacOs della Apple o Windows 95 della Microsoft sono sistemi multitasking che consentono l'esecuzione di più programmi indipendenti a beneficio di un unico utente.

6.2 Architettura e modello di esecuzione

Da alcuni decenni ormai, i calcolatori, anche quelli del tipo personal computer contengono più processori che condividono una memoria comune e devono pertanto essere considerati come sistemi concorrenti in grado di svolgere più attività indipendenti allo stesso tempo. L'architettura hardware a cui si farà riferimento è rappresentata schematicamente in Figura 6.2.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

Come si può osservare, esistono almeno due processori autonomi, ossia una CPU e un processore di ingresso/uscita (il nome usato per tale tipo di processore dipende dal tipo di calcolatore: in quelli di tipo personal computer viene chiamato Direct Memory Access o DMA, mentre nei mainframe prende il nome di processore di I/O o canale di I/O.

Nei sistemi mainframe o server aventi elevate prestazioni, sono solitamente presenti più CPU e più processori di I/O. I diversi processori del sistema accedono ad una memoria comune ed è presente un apposito circuito hardware che risolve eventuali conflitti derivanti dal fatto che più processori richiedono di accedere alla memoria simultaneamente: grazie a tale circuito, ogni processore può essere programmato come se fosse l'unico a accedere alle informazioni contenute in memoria.

Esiste quindi nei sistemi concorrenti un parallelismo potenziale di operazioni a livello hardware, nel senso che è possibile fare in modo che processori distinti eseguano allo stesso tempo istruzioni appartenenti a programmi diversi.

Vediamo ora come deve essere organizzato un sistema operativo per sfruttare al meglio i processori tra loro indipendenti presenti nell'elaboratore.

6.3 Processi e risorse

La nozione di *processo* (in inglese process o task) è emersa all'inizio degli anni '60 insieme allo sviluppo dei primi sistemi multitasking.

In tali sistemi è necessario distinguere l'attività svolta da un processore dalla esecuzione di un programma: infatti, in un intervallo di tempo prefissato, un processore può alternativamente eseguire sequenze di istruzioni appartenenti a programmi diversi mentre, durante lo stesso intervallo, l'esecuzione di un programma su quel processore o su altri può essere ripresa e sospesa più volte.

Indichiamo con X un generico programma sequenziale eseguibile su un processore di tipo p. Per distinguere l'attività di un processore da quella di un programma in esecuzione, si definisce, in questo secondo caso, il processo P(X) associato al programma X come la sequenza di istruzioni eseguite da un processore di tipo p per eseguire X.

Anziché parlare di esecuzione di X, si introdurrà la nozione più precisa di *avanzamento del processo P(X) all'istante t*: ogni processo richiede di ese-

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

guire in un ordine prefissato n(X) istruzioni tra quelle contenute in X; l'avanzamento di P(X) è quindi un numero i con: $0 < i < n(X)$ che esprime il fatto che, all'istante t, il sistema ha eseguito le prime i istruzioni di P(X).

Nel modello considerato, il compito del sistema operativo diventa quello di controllare l'avanzamento di un gruppo di processi con il duplice obiettivo di soddisfare le richieste degli utenti e di mantenere impegnati i diversi processori.

Per meglio capire come questo possa essere realizzato, si considera l'avanzamento di un processo P(X) come una serie di transizioni da uno stato all'altro e si distinguono tre stati di un processo chiamati: esecuzione, pronto e bloccato.

- *in esecuzione*: P(X) è in esecuzione quando un processore sta eseguendo istruzioni di X;
- *pronto*: P(X) è in attesa di un processore in grado di eseguire istruzioni di X;
- *bloccato sull'evento E*: P(X) bloccato sull'evento E quando, per potere procedere, esso deve attendere il verificarsi di tale evento.

Il NUCLEO può decidere di fare passare un processo dallo stato di pronto a quello di esecuzione; la funzione che sceglie, tra i vari processi, quello da mettere in esecuzione è chiamata *scheduling* e verrà discussa alla fine di questo capitolo.

In pratica, quindi, lo stato di pronto deve essere decomposto in vari stati che tengono conto dei fattori sopra indicati. Gli stati appena indicati sono una schematizzazione che, nella pratica, deve essere espansa notevolmente, come appare dagli esempi successivi.

Passando allo stato bloccato, vi sono numerosi motivi, tra loro diversi, per cui un processo rimane in tale stato; ogni specifico cambiamento di stato del sistema potrà servire, di norma, a sbloccare solo alcuni tra i vari processi bloccati.

Lo stato bloccato deve quindi essere decomposto in tanti stati quanti sono i tipi di eventi che possono bloccare i processi. Anche lo stato di esecuzione deve essere decomposto in più stati poiché è necessario associare ad ogni processore il processo P(X) in esecuzione su di esso.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

6.4 Il NUCLEO coordinatore di processi

Il modulo del sistema operativo che coordina l'avanzamento dei processi e notifica ad essi il verificarsi di eventi prende il nome di *NUCLEO* (in inglese, kernel). Chiaramente, il NUCLEO non è un processo esso stesso ma costituisce la parte più critica dell'intero sistema operativo. Come vedremo in un successivo capitolo, le parti meno critiche del sistema operativo possono essere realizzate tramite appositi processi mentre particolare attenzione va posta nel progettare un NUCLEO affidabile ed efficiente.

La comunicazione tra un processo ed il NUCLEO avviene tramite un insieme predefinito di funzioni che prendono il nome di *chiamate di sistema* (in inglese, *system call*).

L'insieme di chiamate di sistema disponibili costituisce in un certo senso il biglietto da visita del sistema operativo: ogni servizio aggiuntivo offerto da un sistemi operativo, ad esempio una gestione sofisticata di file oppure uno schema di protezione rivolto a gruppi di utenti, dà luogo ad apposite chiamate di sistema. Tali funzioni devono essere considerate come delle porte di accesso al NUCLEO: un processo non può richiedere di eseguire un generico programma del NUCLEO ma può soltanto richiedere di eseguire una delle chiamate di sistema esistenti.

Spesso, viene introdotto un ulteriore strato di software tra i programmi degli utenti e quelli del NUCLEO: si tratta di funzioni di tipo API (Application Programming Interface) contenute in apposite librerie di programmi (vedi sezione 1.3). Tali funzioni che rendono più agevole la richiesta di servizi al NUCLEO includono sempre una o più chiamate di sistema.

La Figura 6.3 illustra le interazioni tra programma utente, API, chiamata di sistema e NUCLEO: in tale figura, si fa riferimento alla API `malloc()` usata dai programmatori C per ottenere memoria dinamica. Tale funzione non è una chiamata di sistema bensì una API che utilizza la chiamata di sistema `brk()` per ottenere memoria dinamica dal NUCLEO.

In conclusione, il ruolo della libreria API è essenzialmente quello di facilitare la richiesta di servizi al NUCLEO: le API possono essere considerate come delle chiamate di sistema ad alto livello che risultano più facili da utilizzare delle chiamate di sistema.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

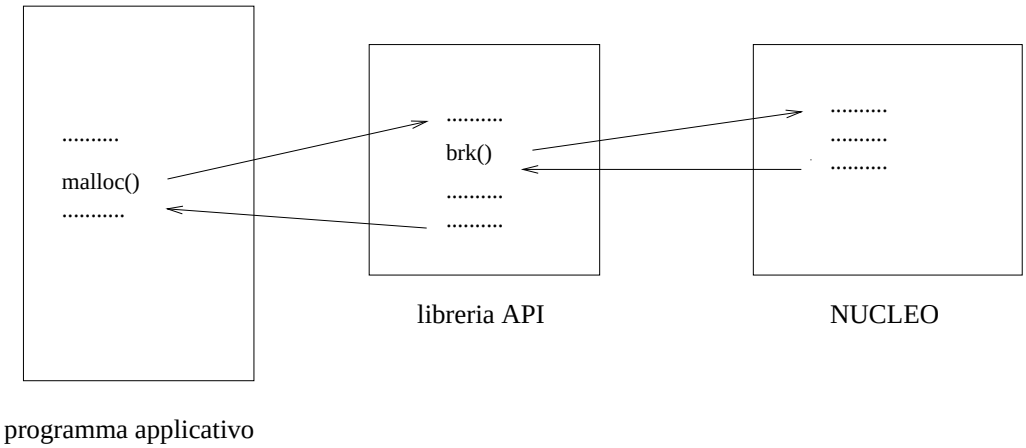

Figura 6.3: API, chiamate di sistema e NUCLEO.

6.5 Risorse

Procedendo nella presentazione del modello, si osserva che i processi, come gruppo di entità autonome, non sono sufficienti a rappresentare il funzionamento di un sistema concorrente: in realtà, i processi interagiscono tra di loro durante l'avanzamento, riflettendo in questo modo il fatto che i processori comunicano tra di loro e si sincronizzano tramite svariati segnali di controllo. Nei sistemi concorrenti non si può escludere, anzi è un caso frequente, che due o più processi richiedano simultaneamente l'uso dello stesso processore, oppure che essi indirizzino lo stesso lettore di disco.

Per consentire di trattare in maniera uniforme l'assegnazione ai processi di componenti hardware e software del sistema, si introduce il concetto di *risorsa* come qualunque oggetto che i processi usano e che condiziona il loro avanzamento. In base a tale definizione, l'avanzamento di un processo è condizionato dalla disponibilità di risorse di vario tipo. Uno dei compiti tipici dei sistemi operativi multitasking è quello di controllare l'uso di risorse da parte dei processi in modo da risolvere eventuali conflitti.

Prima di descrivere, nel prossimo paragrafo, come ciò possa essere realizzato, si introducono alcuni attributi importanti che servono a meglio caratterizzare le varie risorse gestite. A tale scopo, si classificano le risorse in:

- *risorse permanenti*: tale risorsa può essere utilizzata ripetutamente da

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

più processi senza che il suo stato venga modificato. Le interazioni tra un processo P(X) e una risorsa R avvengono secondo il seguente schema:

1. richiesta di R da parte di P(X);
2. assegnazione da parte del sistema operativo di R a P(X);
3. utilizzo di R da parte di P(X);
4. rilascio di R da parte di P(X).

Una risorsa permanente è *seriale* se può essere utilizzata da un solo processo alla volta, è *condivisa* (shared) nel caso opposto.

Un'area di memoria contenente dati usati da un solo processo è una risorsa permanente seriale. Viceversa, un'area di memoria contenente dati usati da più processi, ad esempio una libreria di programmi, è una risorsa permanente condivisa.

- *risorse consumabili*: sono oggetti transienti gestiti dal sistema operativo dall'istante in cui sono prodotte a quello in cui sono consumate. Esse possono essere prodotte da processi, o dal NUCLEO stesso mentre possono essere consumate da processi oppure dal NUCLEO.

  Si è visto in precedenza che un processo P(X) può essere bloccato in attesa di un evento di tipo E. Si supponga che un altro processo Q(Y) sia quello le cui azioni daranno luogo al verificarsi di un evento di tipo E. In tale caso, si dirà che P(X) è sincronizzato rispetto a Q(Y), nel senso che rimarrà bloccato fino a quando Q(Y) non avrà eseguito una determinata azione.

  Parleremo più diffusamente di tali risorse quando affronteremo il problema della sincronizzazione tra processi.

6.6 Scheduling di processi

Come detto in precedenza, un compito importante del NUCLEO è quello di distribuire l'uso della CPU in modo equo tra i vari processi in stato di pronto.

Il problema più generale trattato in questo paragrafo è quello di determinare in che ordine debbano essere soddisfatte le richieste dei processi per risorse

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

di vario tipo. La soluzione di tale problema, chiamata *scheduling* è essenzialmente una strategia che, tenendo conto di diversi obiettivi, consente al sistema operativo di scegliere il prossimo processo a cui assegnare una delle risorse gestite, non appena essa è nuovamente disponibile.

Poiché tali risorse hanno caratteristiche tra loro molto diverse, un sistema operativo include, in generale, più funzioni di scheduling. Per semplicità, si concentrerà l'attenzione in questo paragrafo su due di esse chiamate *scheduling a breve termine* e *scheduling a lungo termine*.

Tornando alla nozione di scheduling, è importante osservare che, in assenza di obiettivi espliciti, qualunque strategia ha lo stesso valore, per cui potrebbe risultare soddisfacente, in tale contesto, usare funzioni di scheduling che determinano in modo del tutto casuale l'ordine secondo il quale soddisfare le richieste dei processi.

In realtà, questo non avviene poiché ogni sistema operativo è progettato per conseguire, sia pure con enfasi diverse, due obiettivi principali: la prevedibilità dei tempi d'esecuzione dei processi e l'utilizzazione delle risorse.

6.6.1 Prevedibilità dei tempi d'esecuzione

Tale obiettivo si riferisce al fatto che, sia nei sistemi batch che in quelli interattivi, è importante garantire all'utente un livello di servizio minimo, qualunque sia l'occupazione del sistema al momento in cui è eseguito il job o il comando.

Tale *livello di servizio* può essere definito in vari modi: nei sistemi interattivi, si misura generalmente come tempo di risposta del sistema nell'eseguire un semplice comando, ad esempio un comando di editing. Nei sistemi a lotti (batch), si usano invece misure più complesse come il *tasso di servizio*, ossia il numero di unità di servizio ricevute dal job per unità di tempo. Tali unità di servizio tengono conto sia del tempo di CPU che del numero di KByte x ora (area di memoria RAM per unità di tempo) ottenute dal job da quando è stato immesso nel sistema.

6.6.2 Utilizzazione delle risorse

Questo obiettivo è considerato prioritario nei sistemi di grandi dimensioni e, in particolare, nei sistemi di gestione a lotti. Data la diversità delle risorse

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

gestite, il sistema operativo stabilisce una gerarchia di priorità: le risorse più pregiate sono i processori, seguiti dalla memoria, dalla memoria secondaria e, infine, dalle unità a nastro e a disco che consentono di montare volumi rimovibili e dalle stampanti.

È utile sottolineare che mentre è relativamente agevole utilizzare in modo ottimale una singola risorsa, risulta invece impossibile, date le caratteristiche dei job eseguiti in multiprogrammazione, determinare uno scheduling che ottimizzi tutte le risorse allo stesso tempo. Una buona utilizzazione delle risorse, ottenibile tramite una messa a punto (tuning) dei parametri associati alle varie funzioni di scheduling, consente di aumentare la produttività del sistema e quindi, indirettamente, di migliorare la prevedibilità dei tempi di risposta a parità di carico di lavoro.

6.6.3 Scheduling a breve termine

Lo *scheduling a breve termine* è utilizzato per gestire le risorse hardware più pregiate del sistema, ossia i processori. Si indichi con P il processo in esecuzione su un processore. Tale funzione di scheduling è eseguita quando:

- P passa nello stato di bloccato sull'evento E, per cui rilascia il processore che può essere riassegnato ad un qualche altro processo;
- P rilascia volontariamente la CPU e passa nello stato di pronto (vedi chiamata di sistema `sched_yield()` descritta nel prossimo capitolo);
- il NUCLEO verifica che P ha esaurito il suo quanto di tempo;
- un processo diverso da quello in esecuzione passa dallo stato di bloccato a quello di pronto.

In questo secondo caso, è opportuno chiamare la funzione di scheduling per eseguire un controllo poiché potrebbe verificarsi che sia più conveniente mettere in esecuzione un altroprocesso nello stato di pronto P' anziché continuare l'esecuzione di P.

In pratica, lo scheduling a breve termine è eseguito con frequenza elevata poiché i suddetti cambiamenti di stato di processi si verificano decine o centinaia di volte al secondo. Per questi motivi, la funzione è interamente realizzata da programmi del NUCLEO e non può essere svolta da un processo.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

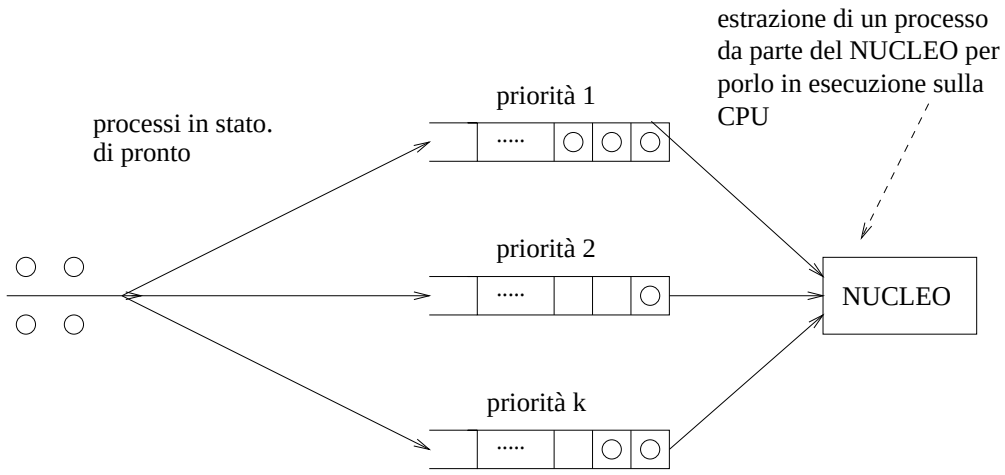

Figura 6.4: Code di priorità tra i processi in stato di pronto.

Si illustrano brevemente due tipiche strategie valide, rispettivamente, per sistemi interattivi e per quelli con gestione a lotti.

Priorità decrescenti e multiplamento nel tempo della CPU

Si associa ad ogni processo una priorità dinamica che può essere aggiornata durante il suo avanzamento: un processo interattivo inizia con la priorità massima $prior1$; se utilizza un tempo $T1$ di CPU senza terminare, la sua priorità diventa $prior2 < prior1$; se utilizza un tempo $T1+T2$ di CPU senza terminare, la sua priorità diventa $prior3$; e così via fino a raggiungere una priorità minima $priork$.

Lo stato di pronto è decomposto in tante code quante sono le priorità (vedi Figura 6.4).

Nello scegliere un nuovo processo da porre in esecuzione, il NUCLEO considera per primo la coda dei processi aventi priorità massima; se tale coda è vuota, considera quella dei processi aventi priorità immediatamente inferiore a quella massima, e così via. Quando un processo ha una determinata priorità pi, il tempo di CPU Ti non è concesso al processo in un unico blocco ma in un gruppo di *ni quanti di tempo* di durata Qi ($ni \times Qi = Ti$), secondo la tecnica della partizione di tempo (time-sharing) già introdotta nel primo capitolo.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

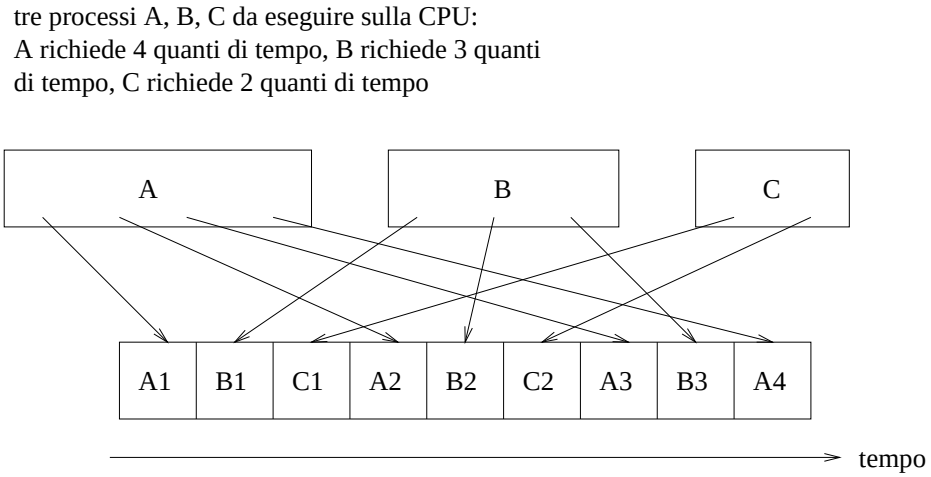

Figura 6.5: Time sharing della CPU.

Tale tecnica, illustrata tramite un semplice esempio in Figura 6.5, consiste nell'assegnare a turno un quanto Qi di CPU ad ogni processo con priorità pi; quando un processo termina il suo quanto, esso viene posto in fondo alla coda opportuna.

La motivazione dietro tale strategia è duplice: privilegiare i processi che richiedono poco tempo di CPU e garantire ad essi un tasso di avanzamento uniforme grazie al multiplamento nel tempo della CPU.

Priorità variabili in funzione dell'uso dei processori

La strategia usata da sistemi per la gestione a lotti (batch) è quella di ottimizzare la produttività del sistema multiprogrammando gruppi di processi che impegnano i diversi processori del sistema. A tale scopo, il sistema operativo misura ogni T unità di tempo le percentuali di utilizzo dei processori da parte di ognuno dei processi utenti multiprogrammati. In base a tali rilevamenti, è possibile identificare i processi che, durante l'ultimo intervallo di tempo misurato, hanno utilizzato maggiormente la CPU da quelli che hanno utilizzato maggiormente i processori di I/O. In seguito all'esame effettuato, la funzione di scheduling diminuisce le priorità dei processi del primo tipo e aumenta quelle dei processi del secondo tipo. La giustificazione per tale scel-

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

ta è che, impegando maggiormente i processori di I/O e quindi i dispositvi di I/O in generale, aumenteà la produttività del sistema.

6.6.4 Scheduling a lungo termine

Tale tipo di scheduling avviene solo nei sistemi operativi di gestione a lotti dotati di un batch monitor in grado di controllare l'esecuzione dei vari job nel sistema (vedi sezione 2.4.2. In tali sistemi, i vari job da eseguire hanno due importanti caratteristiche:

- l'utente dichiara tramite apposite richieste nella scheda iniziale del job e/o nelle schede iniziali dei vari step le risorse richieste dall'intero job, oppure dal singolo step

- le richieste si riferiscono spesso a risorse che richiedono un lungo periodo di inizializzazione prima di poter essere utilizzate (risorse con elevato tempo di setup). Molti job richiedono, ad esempio, di montare volumi di dischi o nastri prefissati, oppure di inserire sulla stampante carta di tipo speciale, ecc. In ognuno di questi esempi è richiesto l'intervento dell'operatore per montare e smontare volumi o per cambiare carta e la durata di tali interventi è dell'ordine dei minuti.

Le funzioni che realizzano lo *scheduling a lungo termine* sono eseguite solo quando inizia o termina un lavoro (job) di un utente.

Distinguiamo due tipi di situazioni:

- lo scheduling di job per quanto riguarda l'uso di unità periferiche dedicate (stampanti, unità a nastro, plotter, ecc.) è solitamente effettuato dall'operatore del sistema: usando opportuni comandi privilegiati della console del sistema, egli può assegnare unità periferiche a un job e iniziarne l'esecuzione.

- lo scheduling di altre risorse, ad esempio la memoria principale o lo spazio su disco, è effettuato da processi di sistema (processi specializzati che svolgono funzioni del sistema operativo, vedi ultimo capitolo) chiamati *INITIATOR*: essi corrispondono ai processi shell in un sistema interattivo e cercano di ottenere tutte le risorse necessarie per eseguire uno step di un job.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

In alcuni casi, può essere effettuata la *preemption* o *prerilascio* di uno o più step di un job per consentire ad un altro step di un nuovo job di andare in esecuzione immediatamente.

Se, ad esempio, l'INITIATOR di un nuovo job che dà luogo a un processo con priorità massima non trova la memoria necessaria per caricare il programma richiesto, il NUCLEO può richiedere il prerilascio da parte di altri processi meno prioritari delle aree di memoria ad essi assegnate. In tale caso, il NUCLEO dovrà salvare su disco tutte le aree di memoria usate dai processi preemptati in modo da essere in grado di rimettere successivamente tali processi in esecuzione.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

Capitolo 7

PROGRAMMAZIONE CONCORRENTE

7.1 Introduzione

Rispettando l'approccio "top down" seguito finora, illustriamo il supporto offerto dal sistema operativo nel realizzare programmi concorrenti, ossia programmi in grado di sfruttare nel modo più efficiente possibile i diversi processori presenti nel sistema.

Rimandiamo invece ad un capitolo successivo la descrizione delle principali azioni svolte dal NUCLEO nel trattare le chiamate di sistema attinenti alla programmazione concorrente.

7.2 Definizione di programma concorrente

I programmi si possono classificare in *sequenziali* e *concorrenti*: nel primo caso, l'esecuzione di un programma dà luogo ad un singolo processo; nel secondo caso, dà luogo a più processi che competono tra loro per l'uso delle risorse del sistema.

Da un punto di vista funzionale, non vi è differenza tra i due tipi di programmazione: tutto quello che può essere realizzato da un programma concorrente può anche essere realizzato da un programma sequenziale. Dal punto di vista della produttività, invece, i programmi concorrenti sono, in generale, più efficienti di quelli sequenziali, specialmente su macchine dotate di più CPU.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

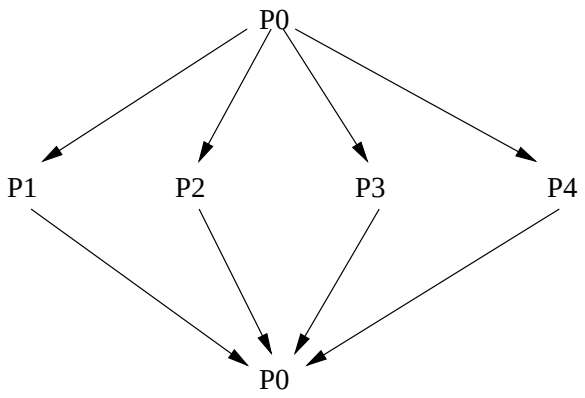

Figura 7.1: Un esempio di sincronizzazione tra processi.

Esistono numerose applicazioni, ad esempio nel campo del calcolo numerico ed in quello della elaborazione delle immagini, in cui l'algoritmo utilizzato si presta ad essere codificato come un gruppo di programmi autonomi che possono essere eseguiti indipendentemente uno dall'altro.

Oltre a dare luogo a più processi, i programmi concorrenti hanno un'altra importante caratteristica: devono fare in modo che i vari processi si sincronizzino tra loro in modo corretto.

Abbiamo visto in precedenza come l'interprete di comandi crei un nuovo processo per mandare in esecuzione il file eseguibile specificato dall'utente, per poi porsi in attesa della terminazione di tale processo. Già da questo semplice esempio, appare evidente la necessità di introdurre appositi *vincoli di sincronizzazione* tra processi: lo shell principale riprenderà la sua esecuzione soltanto quando lo shell secondario lo avrà avertito in qualche modo che ha svolto il compito assegnatoli.

Un altro classico esempio di sincronizzazione tra processi è illutrato nella figura 7.1. Il processo principale P0 crea quattro processi figli P1, ..., P4 che possono operare in parallelo su dati diversi. Soltanto quando tutti i processi figli hanno terminato, P0 può riprendere l'esecuzione dopo avere eliminato i quattro processi creati in precedenza. In questo caso, il vincolo di sincronizzazione è del tipo: "aspetta la terminazione di P1 e di P2 e di P3 e di P4".

Come si scrive un programma concorrente? Il modo più facile è quello di

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

servirsi di apposite librerie di programma quali PVM od altre, che rendono la programmazione concorrente alquanto agevole.

Un altro modo, più a basso livello, consiste nell'usare un linguaggio di programmazione convenzionale arricchito da apposire API o chiamate di sistema per:

- creare nuovi processi;
- eliminare processi esistenti;
- imporre vincoli di sincronizzazione tra processi concorrenti.

È importante ricordare che le suddette API o chiamate di sistema sono soltanto degli strumenti per realizzare programmi concorrenti e non costituiscono di per sé una soluzione ad uno specifico problema di interazione.

In tale ottica, si può affermare che un qualsiasi linguaggio di programmazione sequenziale come il Pascal o il Cobol, arrichito con apposite API o chiamate di sistema, si trasforma in un linguaggio concorrente mediante il quale diventa possibile realizzare gli schemi di interazione richiesti.

Descriviamo ora in modo alquanto generale le caratteristiche delle chiamate di sistema offerte dai NUCLEI dei sistemi operativi multitasking.

7.2.1 Creazione e eliminazione di processi

Le chiamate di sistema per la creazione dinamica di processi differiscono per quanto riguarda lo spazio degli indirizzi e, in generale, i diritti d'accesso del nuovo processo alle varie risorse del sistema. Il loro uso dipende dalla tipologia del sistema operativo da realizzare.

Nel caso più semplice, il processo creato dinamicamente condivide le risorse possedute dal processo che lo ha creato. In particolare, la protezione degli spazi degli indirizzi si applica solo a gruppi di processi, ossia il processo iniziale più tutti i rimanenti processi eventualmente creati condividono lo stesso spazio degli indirizzi e il sistema operativo si limita a garantire che un gruppo di processo non possa accedere allo spazio degli indirizzi di un altro gruppo di processi.

Per eseguire una chiamata di sistema del tipo `crea_processo()`, il NUCLEO preleva dai parametri associati alla chiamata le informazioni necessarie (ad

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

esempio, il nome del nuovo processo e l'indirizzo della prima istruzione da eseguire) e crea un nuovo processo che pone nello stato di pronto.

Altre chiamate di sistema del tipo `elimina_processo()` consentono di eliminare processi esistenti.

7.3 Creazione e eliminazione di processi in Unix

Per concretezza, focalizziamo la discussione relativa alla creazione ed alla eliminazione di processi sul sistema operativo Unix.

7.3.1 Creazione di processi in Unix

La chiamata di sistema

```
pid = fork();
```

è utilizzata per creare un processo figlio che è una copia conforme del processo padre. Come evidenziato dalla assenza di parametri, il processo figlio ottiene una copia di tutte le risorse possedute dal padre, e quindi una copia del programma da eseguire. Vi è un'unica differenza che può essere sfruttata dai programmatori per distinguere il figlio dal padre: il valore `pid` dell'identificatore di processo restituito dalla `fork()`. Esso vale 0 quando il codice è eseguito dal processo figlio, mentre ha un valore positivo quando è eseguito dal processo padre.

L'esempio successivo illustra una applicazione della `fork()`. Il processo padre esegue tre azioni distinte:

- crea il processo figlio;
- esegue la chiamata di sistema `sleep(10)` per autosospendersi durante 10 secondi;
- quando ritorna in esecuzione, elimina il processo figlio eseguendo la chiamata di sistema `kill()`.

Il processo figlio esegue un ciclo senza fine durante il quale invia un messaggio di stampa ogni secondo.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

```
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int main()
{
    pid_t pid;
    int retcode;
    printf("inizio prova\n");
    /* crea processo figlio */
    pid = fork();
    if (pid != 0)
        /* programma eseguito da processo padre */
        {
        printf("PID processo figlio= %d\n", (int)pid);
        printf("processo padre inizia attesa di 10 secondi\n");
        sleep(10);
        printf("processo padre elimina processo figlio\n");
        retcode = kill(pid, SIGKILL);
        printf("processo figlio terminato\n");
        }
    else
        /* programma eseguito da processo figlio */
        {
        printf("processo figlio inizia\n");
        pid = getpid();
        while (1)
            {
            printf("processo figlio %d lavora\n", (int)pid);
            sleep(1);
            }
        }
    exit(0);
}
```

Come appare dall'esempio appena illustrato, la chiamata di sistema

```
retcode = kill(pid, SIGKILL);
```


scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

consente di inviare il segnale `SIGKILL` al processo avente identificatore `pid`, e quindi ha come effetto quello di eliminare il processo destinatario.

Si osservi inoltre come, nella parte di codice eseguita dal processo figlio, si fa uso della chiamata di sistema `getpid()` per ottenere il PID del processo figlio, poiché il valore iniziale, ossia 0, restituito dalla `fork()` non è attendibile.

Si osservi infine che non soltanto le variabili locali ma anche quelle globali *si riferiscono ad un singolo processo*. Ne consegue che processi distinti non possono scambiarsi dati tramite variabili globali.

Accenneremo nel prossimo paragrafo ad alcune tecniche che consentono ai processi di scambiarsi dati.

7.3.2 Sincronizzazione tra processi

Arricchiamo il modello processi/NUCLEO considerato nel capitolo precedente in modo da consentire al programmatore di specificare vincoli di sincronizzazione tra i processi.

L'estensione consiste nel fare uso di un particolare tipo di risorsa chiamata "risorsa consumabile" (vedi paragrafo 6.5). Tutte le diverse forme di sincronizzazione tra processi tipiche della programmazione concorrente sono infatti risolte tramite opportune produzioni e consumi di risorse consumabili di vario tipo. In generale, le risorse consumabili sono divise in classi e corrispondono a classi di eventi dello stesso tipo che si verificano in istanti diversi.

Due o più processi interagiscono quando si contendono l'uso di risorse permanenti. Esistono molti schemi di interazione possibili ma essi sono tutti riconducibili a due tipi fondamentali chiamati *mutua esclusione* e *produttore/consumatore*.

7.3.3 Ruolo delle risorse consumabili

Un qualsiasi vincolo di sincronizzazione tra P(X) e Q(Y) può essere espresso molto semplicemente tramite risorse consumabili affermando che P(X) è il *produttore* di una risorsa consumabile R(E) associata ad eventi di tipo E e che Q(Y) è il *consumatore* di R(E).

In generale, ogni evento significativo dà luogo alla creazione di una apposita risorsa consumabile che continuerà ad esistere fino a quando non sia stato riconosciuto e trattato dal sistema l'evento corrispondente.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

I vari costrutti di programmazione utilizzati per descrivere la sincronizzazione tra processi implicano tutti risorse consumabili di tipo software create e consumate dal NUCLEO e dai processi. Rispetto alle risorse permanenti, quelle consumabili si differenziano per i seguenti motivi:

- il numero di risorse consumabili presenti nel sistema è variabile nel tempo;
- la vita di una risorsa consumabile è limitata: inizia nell'istante in cui essa è prodotta e termina nell'istante in cui è consumata.

7.3.4 Mutua esclusione

Il problema della mutua esclusione di una risorsa permanente R contesa da più processi P1, ..., Pn è quello di garantire che, ad ogni istante, vi sia al più un processo Pi che occupi R e che ogni processo richiedente ottenga l'uso di R entro un intervallo limitato di tempo.

Ricordando quanto affermato in precedenza, si osserva che si presenta un problema di mutua esclusione ogni qualvolta si intende rendere seriale una qualche risorsa permanente del sistema.

In effetti, risorse permanenti hardware come le aree di memoria o i lettori di disco non sono di per sé seriali, anche se è necessario, per un corretto avanzamento dei processi che le utilizzano, che esse lo diventino. In modo analogo, risorse permanenti software come una tabella di dati, una lista o un file non sono di per sé seriali, anche se, per mantenere la coerenza dei dati in esse contenute, è necessario che gli aggiornamenti siano eseguiti da un processo alla volta.

Esistono varianti al problema appena citato: in alcuni casi, i processi che richiedono la risorsa possiedono ognuno una priorità, ossia un numero che esprime il tipo di privilegio che essi hanno rispetto alle risorse. Quando si libera una risorsa, essa è assegnata al processo con priorità massima e, nel caso ve ne sia più di uno, a quello che ha atteso più a lungo.

Un'altra variante prevede, oltre alle priorità, l'uso del prerilascio (in inglese, *preemption*) della risorsa per soddisfare immediatamente le richieste di processi prioritari. In questo caso, non appena giunge una richiesta da parte di un processo avente priorità sufficientemente elevata, il NUCLEO sospende il processo che usa attualmente la risorsa ponendolo in stato di pronto ed assegna quindi la risorsa al nuovo processo.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

7.3.5 Produttore/consumatore

Questo secondo schema di interazione è alla base di tutti gli scambi di informazione tra processi tra loro interagenti. Si distingue il processo produttore di informazioni da quello consumatore. Si noti come, a differenza della mutua esclusione che si applica anche a processi logicamente indipendenti (processi operanti su dati diversi), lo schema del produttore/consumatore implica che due o più processi si scambino informazioni, ossia che operino su dati comuni.

Si presentano alcune importanti varianti del problema che schematizzano problemi di sincronizzazione comuni ai sistemi operativi multitasking

- *un produttore/un consumatore*: l'unico produttore P produce in istanti imprevedibili informazioni che devono essere raccolte e trattate dall'unico consumatore Q. Q non può procedere se l'informazione richiesta non è disponibile; viceversa, P può produrre, anche se Q non ha ancora elaborato l'informazione ricevuta in precedenza. Se questo si verifica, si avrà una perdita di informazioni.

  Una interazione del genere si verifica tra la tastiera di un terminale e il processo Q incaricato di raccogliere i caratteri generati: ogni tasto premuto dà luogo ad un carattere che è trasferito dal canale di I/O in un byte di memoria. Ogni carattere può essere considerato come una risorsa consumabile R prodotta da tastiera e consumata da Q. Quando è pronto a ricevere un nuovo carattere da tastiera, Q chiede di consumare una unità di R; se R non è disponibile, il NUCLEO bloccherà Q sull'evento "nuova produzione di una unità di R"; altrimenti, Q è autorizzato dal NUCLEO a consumare R.

- *più produttori/un consumatore*: la funzione di spooling di uscita descritta nel par. 2.4.2 è un programma concorrente che prevede una interazione di tipo produttore/consumatore tra i vari processi utenti che producono file di stampa e il processo OUTPUT WRITER che li consuma trasferendoli sulla stampante.

- *più produttori/più consumatori*: per aumentare le prestazioni del programma concorrente (posto che esistano le opportune risorse hardware) può essere opportuno dedicare un gruppo di processi per svolgere in multiprogrammazione, e quindi con maggiore efficienza, la stessa funzione. Nei sistemi per la gestione a lotti, ad esempio, i lavori sono

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

suddivisi in classi in base alla priorità e alle risorse richieste. Ad ogni classe è associato almeno un processo del sistema operativo chiamato INITIATOR; tale processo esamina le richieste del lavoro e dei suoi passi e provvede a soddisfarle, oltre che a controllare l'esecuzione di ogni passo. Allo stesso tempo, è realizzato uno spooling di entrata che consente a più unità di input locali o remote di inviare simultaneamente lavori. A tale scopo, ogni unità di input è gestita da un apposito processo di sistema chiamato INPUT READER; tale processo smista i lavori ricevuti, copiandoli in file diversi a seconda della loro classe e notifica quindi uno degli INITIATOR associati alla classe del lavoro appena ricevuto.

La Figura 7.2 illustra le principali interazioni tra i processi di sistema in un sistema per la gestione a lotti.

Come appare dalla figura, sono presenti due schemi di interazione del tipo più produttori/più consumatori: il primo è relativo all'immissione di dati nel sistema; le risorse consumabili sono i lavori prodotti da più INPUT READER e consumati da più INITIATOR. Il secondo è relativo alla emissione di dati nel caso in cui siano presenti più stampanti veloci omogenee; le risorse consumabili sono gli file di stampa prodotti da processi in esecuzione e consumati da più OUTPUT WRITER.

7.3.6 Primitive di sincronizzazione

Esistono numerose varianti, sia per quanto riguarda i nomi, sia per quanto riguarda la definizione delle chiamate di sistema che riguardano la sincronizzazione tra processi. Tali chimate di sistema prendono il nome di *primitive di sincronizzazione*. Per motivi di spazio, se ne descrivono soltanto alcune. Come nel caso delle chiamate di sistema per la creazione/eliminazione di processi, illustreremo per motivi di concretezza alcune primitive di sincronizzazione per il sistema operativo Unix.

- *`lock()/unlock()`*: sono usate per risolvere problemi di mutua esclusione su una qualunque risorsa. Quando un processo P esegue `lock(X)`, il NUCLEO verifica se esiste una unità di risorsa consumabile X chiamata *semaforo*. Nel caso affermativo, essa viene eliminata e P è rimesso in esecuzione. Altrimenti, P è bloccato dal NUCLEO fino a quando un altro processo non crei una unità di X. Quando un processo P esegue

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

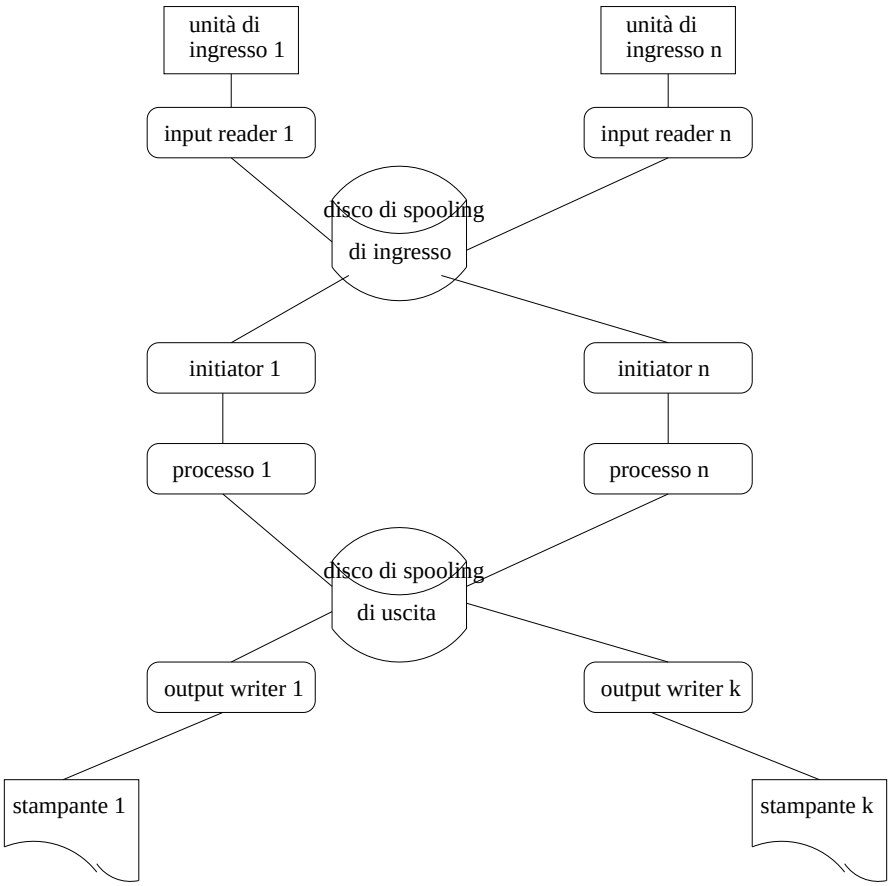

Figura 7.2: Interazione tra processi di sistema in un sistema per la gestione a lotti.

`unlock(X)`, il NUCLEO verifica se esiste almeno un processo bloccato su X, nel caso affermativo, sblocca il primo di essi; altrimenti, crea una unità di X. In entrambi i casi, rimette P in esecuzione.

La serializzazione di una risorsa permanente R contesa da più processi si ottiene associando a R una risorsa consumabile XR e facendo in modo che ogni processo esegua `lock(XR)` prima di utilizzare R e `unlock(XR)` dopo averla utilizzata. Spesso, per motivi di efficienza, il NUCLEO offre un insieme prefissato di risorse consumabili X1, ..., Xn alle quali è possibile attribuire un significato arbitrario. Anche semplici problemi

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

del tipo produttore/consumatore possono essere risolti con le suddette primitive, serializzando tramite un apposita risorsa consumabile Xi una variabile Ni che rappresenta il numero di oggetti di tipo i prodotti e non ancora consumati.

- *`send()` e `receive()`*: consentono a processi diversi di scambiarsi informazioni, garantendo allo stesso tempo la protezione degli spazi degli indirizzi di ogni processo. La risorsa consumabile è il *messaggio*, ossia una sequenza di caratteri spesso di lunghezza fissa, per motivi di efficienza. Il contenuto di un messaggio è arbitrario ed è stabilito dal processo mittente.

  Servendosi della primitiva `send(dest,m)`, il mittente invia ad un altro processo chiamato `dest` il messaggio `m`. Un'altra primitiva del tipo `receive(mitt,m)` consente ad un processo di richiedere un nuovo messaggio. In pratica, ad ogni processo è associata una coda di messaggi pendenti, ossia trasmessi e non ancora ricevuti.

  Come nel caso della `unlock,()` la `send()` non può bloccare il processo che la esegue. Viceversa, se un processo P esegue una receive e il NUCLEO verifica che non vi sono messaggi pendenti destinati a P, esso blocca P fino a quando un altro processo non esegua una send destinata ad esso.

  Le primitive `send()` e `receive()` sono utilizzate per risolvere problemi di sincronizzazione del tipo produttore/consumatore: nel caso del sistema di spooling di uscita, ad esempio, il processo che chiude un file di stampa invia al processo OUTPUT WRITER un messaggio contenente informazioni circa il file di stampa appena prodotto.

7.3.7 Primitive di sincronizzazione in Unix

Unix include diverse primitive di sincronizzazione. In particolare, le funzioni incluse nel pacchetto chiamato "System V InterProcess Communication (IPC)" consentono di operare su lock, di inviare o ricevere messaggi, e di condividere aree di memoria tra processi. Purtroppo le chiamate di sistema IPC sono piuttosto macchinose da utilizzare, per cui preferiamo introdurre due funzioni Unix che corrispondono alle primitive di sincronizzazione `lock()` e `unlock()` descritte in precedenza.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

L'idea di base per realizzare lock in Unix è quella di sfruttare le caratteristiche del file system Unix, ed in particolare delle chiamate di sistema `open()` con i relativi parametri. Se infatti un file è stato aperto da un processo con il parametro `O_EXCL`, nessun altro processo può aprire il file finché il file non viene chiuso dal processo che lo ha aperto in precedenza.

Possiamo quindi usare i nomi di file come lock: se il file esiste, il lock corrispondente è impegnato, altrimenti risulta libero.

La funzione `lock(const char *p)` illustrata appresso opera su un singolo parametro `p` che è un puntatore al nome del file, ossia al nome del lock che si intende utilizzare; tale nome può essere sia un pathname assoluto che relativo.

```
int lock(const char *p)
{
    int fd, retcode;

    while (1)
        {
fd = open(p, O_RDWR | O_CREAT | O_EXCL);
if ((fd < 0) && (errno == EEXIST))
    retcode = sched_yield();
else if (fd < 0)
    {
    printf("lock open error\n");
    retcode = -1;
    break;
    }
else
    {
    retcode = fd;
    break;
    }
        }
    return(retcode);
}
```

Come si può osservare dal codice, la funzione `lock()` tenta di aprire il file con modalità esclusiva `O_EXCL`: se un file con lo stesso nome già esiste,

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

`lock()` rilascia temporaneamente la CPU eseguendo la chiamata di sistema `sched_yield()` che pone il processo in ultima posizione tra quelli in stato di pronto. Successivamente, quando lo stesso processo andrà in esecuzione, verrà nuovamente eseguita la `open()` finché l'apertura del file riesce (ciclo `while`).

Quando il file è stato creato e quindi aperto, `lock()` termina restituendo il file descriptor `fd` del file corrispondente al lock ottenuto.

La funzione `unlock(const char *p, int fd)` illustrata appresso opera su due parametri: `p` che è un puntatore al nome del lock ottenuto in precedenza e `fd` che è il file descriptor associato al file aperto.

```
int unlock(const char *p, int fd)
{
    int retcode;

    retcode = close(fd);
    if (retcode < 0)
        {
        printf("unlock close error\n");
        return(retcode);
        }
    else
        {
        unlink(p);   /* cancella il file */
        return(0);
        }
}
```

Come si può osservare dal codice, la funzione `unlock()` inizia col chiudere il file corrispondente al lock. Se tale file non esiste (uso improprio dei lock), segnala una condizione di errore tramite il codice di ritorno. Altrimenti, la funzione `unlock()` provvede ad eliminare il file del lock invocando la chiamata di sistema `unlink()`. Poiché esattamente un processo ha aperto il file, il contatore d'uso del file verrà posto a 0 in seguito alla `unlink()` ed il NUCLEO provvederà quindi a cancellare il file dal disco.

Si osservi il ruolo del primo parametro `p`: esso punta al nome del lock ed è necessario per potere invocare la chiamata di sistema `unlink()`. Non esiste

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

infatti in Unix una funzione che consenta di derivare dal file descriptor il corrispondente nome del file.

7.4 Stallo tra processi

Non si può parlare di programmazione concorrente senza accennare agli *errori di sincronizzazione*, ossia ad errori di programmazione causati dall'uso improprio delle primitive di sincronizzazione.

L'errore di interazione più vistoso e più noto prende il nome di *stallo* o *deadlock*: durante l'avanzamento del gruppo di processi tra loro interagenti, uno o più processi vengono posti in stato di attesa e vi rimangono per un tempo indeterminato.

La figura 7.3 illustra un classico caso di programma concorrente che può dare luogo, in determinate circostanze, ad uno stallo tra due processi.

Tale errore è causato da un uso improprio delle primitive di sincronizzazione `lock()` e `unlock()` da parte dei processi: i due processi A e B fanno uso entrambi della stessa coppia di risorse che chiameremo R1 ed R2. Per garantire l'accesso seriale ad ognuna di esse i processi fanno uso di due semafori chiamati `sem_R1` e `sem_R2` e delle relative primitive di sincronizzazione `lock()` e `unlock` descritte nel paragrafo precedente.

Il processo A esegue, ad esempio, una `lock(sem_R1)` prima di accedere ad R1 ed una `unlock(sem_R1, fd)` per rilasciare R1.

Il problema consiste nel fatto che A richiede prima il semaforo `sem_R1` e poi quello `sem_R2`, mentre il processo B richiede prima il semaforo `sem_R2` e poi quello `sem_R1`. Può quindi succedere che, in seguito allo scheduling deciso dal NUCLEO per A e B, A ottenga il semaforo `sem_R1` e subito dopo B ottenga il semaforo `sem_R2`.

A questo punto si determina una condizione di stallo poiché né A né B sono in grado di continuare la loro esecuzione: ognuno aspetta un semaforo posseduto dall'altro.

Come si evitano errori di interazione di tipo stallo in un programma concorrente? La risposta sta nell'esempio appena illustrato: bisogna obbligare i processi a richiedere semafori in un ordine prefissato, ad esempio in un ordine crescente. È compito del programmatore, e non del sistema operativo, scrivere programmi concorrenti che non contenagano errori di interazione.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

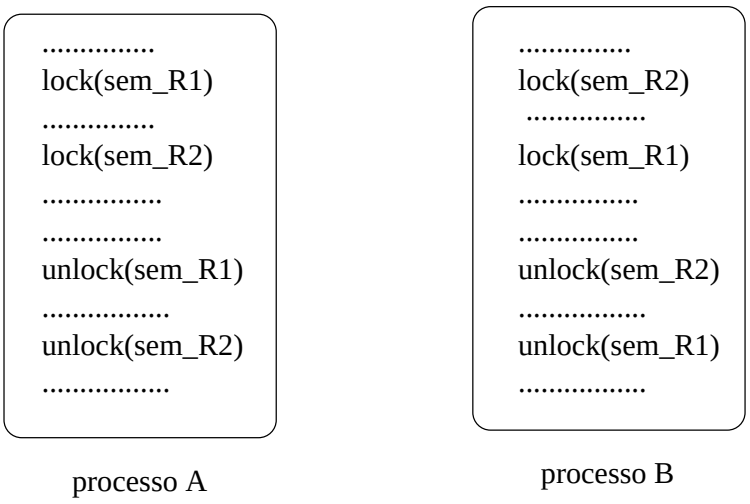

Figura 7.3: Esempio di stallo tra due processi.

7.5 Programmazione in tempo reale

Concludiamo questo capitolo sulla programmazione concorrente dando alcuni cenni su una variante di programma concorrente che è molto utilizzata nell'ambito del controllo di processi.

Un *programma in tempo reale* è un programma concorrente che deve soddisfare dei vincoli relativi ai tempi d'esecuzione di almeno uno dei sottoprogrammi che lo compongono. Programmi di tale tipo sono anche chiamati programmi dipendenti dal tempo.

Un tipico esempio di programma in tempo reale è quello per il controllo di un processo industriale: se il sistema riceve dall'esterno segnali corrispondenti a una situazione che richiede un intervento tempestivo, esso non deve soltanto riconoscere tale situazione ma anche produrre le informazioni richieste entro e non oltre un intervallo prefissato di tempo.

Un altro esempio è quello di un programma per le previsioni metereologiche: in tale caso, il tempo richiesto per eseguire la previsione deve essere inferiore all'inverso della frequenza di rilevazione dei dati metereologici. Anche i sistemi operativi sono, sia pure in misura diversa, dei programmi in tempo reale.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

I due estremi della gamma possono essere rappresentati, da un lato da un semplice sistema interattivo senza particolari vincoli sui tempi di risposta all'utente e, dall'altro, da un sistema dedicato per il controllo di una centrale telefonica elettronica.

Nel primo caso, l'unica parte dipendente dal tempo sono i sottoprogrammi che gestiscono le diverse interruzioni. Nel secondo caso, invece, quasi tutti i sottoprogrammi sono dipendenti dal tempo in quanto ogni ritardo nel trattare una coppia di linee tra cui è in svolgimento una comunicazione si traduce in un malfunzionamento del sistema.

In generale, i sistemi operativi "general purpose" non si prestano ad eseguire correttamente programmi in tempo reale e si preferisce fare uso di sistemi operativi semplificati di tipo "real time" in grado di assicurare tempi di risposta brevi ai processi che lo richiedono.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

Capitolo 8

SPAZIO DEGLI INDIRIZZI DI UN PROCESSO

8.1 Introduzione

Ancora una volta, l'approccio "top down" ci forza a considerare la problematica della gestione della memoria in modo diverso da quello tradizionale. Anziché cominciare dal NUCLEO e descrivere le varie tecniche da esso utilizzate per gestire la memoria RAM, partiamo dai processi ed esaminiamo i motivi per cui essi richiedono o rilasciano memoria; in particolare, studiamo:

- gli eventi più significativi che inducono un processo a richiedere memoria;
- i vari modi, diretti o indiretti, tramite i quali un processo effettua richieste di memoria al NUCLEO.

Rimandiamo invece al prossimo capitolo la discussione di come il NUCLEO assegna effettivamente RAM ai processi.

8.2 Spazio degli indirizzi di un processo

Ci riferiamo in questo contesto a file eseguibili contenenti programmi codificati in linguaggio macchina e pronti ad essere eseguiti dalla CPU. Chiamiamo *indirizzo logico* un gruppo di bit che serve a identificare:

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

- l'indirizzo della prossima istruzione da eseguire;

oppure:

- l'indirizzo di un operando contenuto in RAM (solo nel caso in cui l'istruzione debba accedere ad operandi contenuti nella RAM).

Per semplicità, supponiamo che l'indirizzo logico consista in un gruppo di 32 bit[1]. Negli esempi successivi, rappresenteremo quindi gli indirizzi logici come numeri inclusi nell'intervallo compreso tra `0x00000000` e `0xffffffff`.

Chiamiamo invece *indirizzo fisico* un gruppo di bit usati dal bus per indirizzare una cella di RAM. La maggior parte dei microprocessori fanno uso oggi di indirizzi fisici da 32 bit.

Come vedremo nel capitolo successivo, un compito fondamentale del NUCLEO consiste nel definire un mapping tra indirizzi logici ed indirizzi fisici, per cui durante l'esecuzione di una istruzione, ogni indirizzo logico viene tradotto in un opportuno indirizzo fisico.

Il programmatore può in qualche modo determinare gli indirizzi logici usati dal suo programma ma ignora quali saranno gli indirizzi fisici assegnati al programma durante l'esecuzione. In modo analogo, un compilatore, un assemblatore o un linker possono impostare gli indirizzi logici dei programmi ma non quelli fisici.

Iniziamo col descrivere come il linker (vedi paragrafo 3.2.2) assegna indirizzi logici al programma da eseguire. Procediamo per gradi e supponiamo dapprima che il file eseguibile sia stato linkato in modo statico, ossia che non vengano utilizzate librerie dinamiche.

In questo caso, il linker assegna al processo che dovrà eseguire il programma un gruppo di *regioni di memoria*, ossia intervalli di indirizzi logici, caratterizzate da un indirizzo iniziale e da una lunghezza. Ad un file eseguibile corrispondono quindi diverse regioni di memoria. Per motivi di flessibilità, le regioni di memoria assegnate ad un processo non sono tra loro contigue.

L'insieme di indirizzi logici racchiuso nelle regioni di memoria di un processo prende il nome di *spazio degli indirizzi del processo*: esso include tutti e

[1] In realtà, diversi microprocessori tra cui quelli Intel 80x86, fanno uso di indirizzi logici più complessi composti da due componenti: un segmento da 16 bit ed un offset da 32 bit, l'indirizzo risultante chiamato "indirizzo lineare" è comunque un indirizzo da 32 bit.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

soli gli indirizzi logici che un processo è autorizzato ad usare durante la sua esecuzione.

Ogni indirizzamento effettuato dal processo al di fuori dal suo spazio degli indirizzi viene considerato dal NUCLEO come un indirizzamento non valido dovuto ad un errore di programmazione.[2]

La figura 8.1 illustra un semplice esempio di spazio degli indirizzi di un processo.
Come si osserva dalla figura, il linker ha assegnato al programma 4 regioni di memoria distinte tra loro non contigue:

- *regione codice*: contiene le istruzioni del programma, è lunga 64 KB ed inizia all'indirizzo logico `0x00800000`;
- *regione dati inizializzati*: contiene costanti, è lunga 64 KB ed inizia all'indirizzo logico `0x00c00000`
- *regione dati non inizializzati*: contiene lo heap (memoria dinamica ottenibile tramite la funzione `malloc()`, è lunga 128 KB ed inizia all'indirizzo logico `0x00f00000`
- *regione stack*: contiene lo stack, inizia al massimo indirizzo logico possibile, ossia `0xffffffff` e si espande all'ingiù.

Come fa il loader a assegnare specifiche regioni di memoria alle varie parti del programma? In realtà il loader fa ben poco in quanto le regioni sono già state identificate dal compilatore ed assegnate dal linker.[3]

Approfondiamo la discussione iniziata nel paragrafo 3.2.2 dove non si faceva ancora riferimento a regioni di memoria ma soltanto a variabili esterne.

Riferiamoci alla Figura 3.1: il compilatore ha assegnato al modulo A due section distinte, una per il codice (`prog0`) ed una per le variabili globali (`dati0`) ed ha assegnato a tali section gli indirizzi logici `00000000` e `00b00000`. In modo analogo, quando è stato compilato il modulo B, il compilatore ha assegnato

[2]Una eccezione a tale regola vale per la regione di memoria che contiene lo stack: in tale caso, è possibile che, in seguito ad una serie di `push`, il processo effettui un indirizzamento immediatamente prima dell'inizio della regione (lo stack cresce tradizionalmente per indirizzi decrescenti).

[3]Vedremo più avanti in questo capitolo che altre regioni possono essere assegnate dinamicamente al processo.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

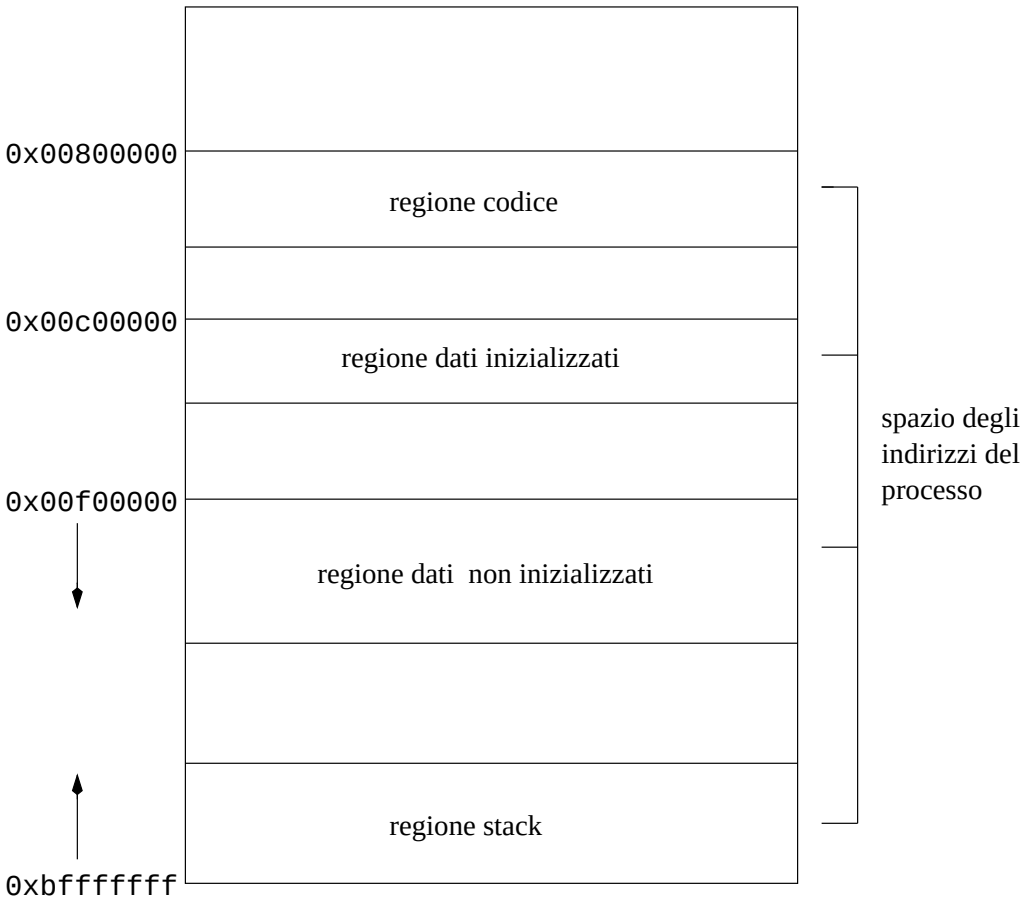

Figura 8.1: Spazio degli indirizzi di un processo.

alla section `prog1` l'indirizzo logico `00000000` e alla section `dati1` l'indirizzo logico `00b00000`. Gli stessi valori sono stati assegnati dal compilatore alle section `prog2` e `dati2` del modulo C.
Dopo il linkaggio dei tre moduli in un unico file eseguibile, le tre section di codice sono state fuse in una regione codice avente indirizzo iniziale `00000000` e, analogamente, le tre section di dati sono state fuse in una regione dati avente indirizzo iniziale `00b00000` (vedi figura 3.2).

Le azioni svolte dal loader dipendono dal tipo di regione e dai registri di indirizzamento presenti nella CPU. Diamo alcuni esempi senza alcuna pretesa di completezza.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

Nel caso di regione codice, è sufficiente impostare nel registro contatore programma (registro `eip` nell'architettura Intel 80x86) il primo indirizzo logico della regione codice. Questo assicura che quando il processo andrà in esecuzione, la CPU eseguirà la prima istruzione contenuta nella regione codice.

Nel caso di regione stack (usata dal compilatore, tra l'altro, per contenere le variabili locali), è sufficiente impostare nel registro puntatore allo stack (registro `esp` nell'architettura Intel 80x86) il primo indirizzo logico della regione stack.

Nel caso di regione dati, il compilatore ha lasciato non risolto (indirizzo esterno) l'indirizzo della section "data" contenente le variabili globali ed ha effettuato indirizzamenti all'interno di tale section in modo rilocabile, avvalendosi di un registro base (registro `ebx` nell'architettura Intel 80x86) opportunamente inizializzato tramite una istruzione del tipo:

```
lea .data, %ebx
```

Durante il linkaggio, il valore corrispondente a `.data` verrà opportunamente impostato.

Da questa discussione preliminare, emerge quindi che quando viene caricato un file eseguibile e viene creato il processo destinato ad eseguire le istruzioni contenute nel file eseguibile, tale processo riceve uno spazio degli indirizzi preparato dal linker.

Vediamo ora come lo spazio degli indirizzi iniziale di un processo possa cambiare durante l'esecuzione del processo: il processo può acquisire nuove regioni di memoria, allargare o restringere la dimensione di regioni di memoria esistenti, o rilasciare regioni di memoria.

Ancora una volta, ci interessiamo alle regioni di memoria e quindi agli indirizzi logici posseduti da un processo e non al come il NUCLEO assegna indirizzi fisici ai vari indirizzi logici.

8.3 Modifiche dello spazio degli indirizzi

La regione "codice" e quella "dati inizializzati" rimangono solitamente immutate durante l'esecuzione del programma. Vice versa, la regione "dati non

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

inizializzati" può crescere in seguito a richieste di memoria dinamica da parte del processo. In modo analogo, la regione "stack" può crescere durante l'esecuzione del programma (ad esempio, per causa dell'annidamento di chiamate di funzioni).

Esiste tuttavia un caso in cui il processo rilascia tutte le regioni di memoria da esso possedute, incluse la regione "codice" e quella "dati inizializzati", ed ottiene un nuovo gruppo di regioni del tutto diverse da quelle ottenute in precedenza. Questo aviene quando il processo effettua una chiamata di sistema per caricare dinamicamente un file eseguibile.

Nei sistemi Unix, la chiamata di sistema che realizza ciò è la `execve()`. Si consideri, ad esempio, il seguente programma C:

```
#include <stdio.h>
#include <unistd.h>

int main(void)
{
    char * const arg[2] = { "/bin/ls", NULL };
    char * const env[1] = { NULL };

    execve("/bin/ls", arg, env);
    fprintf(stderr, "execve() failed\n");
}
```

In questo caso il programa in esecuzione richiede il caricamento del programma `ls` contenuto nella directory `/bin`. In seguito a tale chiamata di sistema, il linker carica `ls` ed assegna al processo un nuovo gruppo di regioni di memoria. Dal punto di vista della programmazione, la `execve()` corrisponde ad un salto senza ritorno, per cui la `fprintf()` viene eseguita solo nel caso in cui la chiamata di sistema non possa essere eseguita correttamente dal NUCLEO.

8.4 Linking dinamico

Supponiamo ora che il file eseguibile sia stato linkato a librerie dinamiche. In questo caso, viene caricato non solo il programma da eseguire ma anche

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

un modulo del linker chiamato *program interpreter* che provvede automaticamente a caricare funzioni di libreria prima di iniziare l'esecuzione del programma.

Illustriamo il ruolo del program interpreter tramite un semplice esempio. Supponiamo che il file eseguibile includa una chiamata alla funzione `sqrt()` inclusa nella libreria C `libm` delle funzioni matematiche.

In seguito al linkaggio dinamico effettuato, il file eseguibile include nella sua testata una serie di voci che si riferiscono a funzioni esterne contenute in qualche libreria; ogni voce consiste nella tripla:

- <funzione esterna>
- <pathname del modulo che definisce la funzione>
- <lista di offset nel modulo in corrispondenza a chiamate alla funzione esterna>

Basandosi sulle informazioni contenute nella suddetta testata, il program interpreter esegue le seguenti azioni:

1. assegna una nuova regione di memoria per ogni libreria dinamica richiesta dal programma;
2. determina l'indirizzo logico della funzione di libreria richiesta sommando all'indirizzo iniziale della regione l'offset all'interno della libreria (ogni libreria include decine o centinaia di funzioni diverse);
3. completa nel codice tutti i riferimenti a funzioni di libreria sostituendo il valore `0` con l'indirizzo logico appropriato (vedi Figure 3.1 e 3.2).

Dal punto di vista delle regioni di memoria ciò implica:

- un gruppo di regioni di memoria per consentire al linker caricato insieme al programma di potere operare;
- una nuova regione di memoria per ogni libreria di funzioni utilizzata.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

Potrebbe sembrare superfluo continuare a riservare regioni di memoria al program interpreter dopo che è iniziata l'esecuzione del programma. In effetti il program interpreter deve continuare ad esistere per consentire al programma di effettuare *linking dinamici* ad altre funzioni.

Nei sistemi Unix, ciò si realizza mediante la chiamata di sistema `dlopen()`: ogni volta che il programma invoca tale chiamata di sistema, entra in funzione il program interpreter per linkare dinamicamente una nuova funzione. Si noti come, a differenza della `execve()`, la `dlopen()` corrisponde ad una chiamata di funzione con ritorno al programma chiamante.

Il seguente programma usa la chiamata di sistema `dlopen()` per caricare dinamicamente dalla libreria matematica `/lib/libm.so.6` del C la funzione radice quadrata `sqrt()` e la applica sul parametro `x`.

```
#include <stdio.h>
#include <dlfcn.h>

int main(int argc, char **argv)
{
    void *handle;
    double (*sqrt)(double);
    double x, y;

    x = 4.1314;
    handle = dlopen ("/lib/libm.so.6", RTLD_LAZY);
    sqrt = dlsym(handle, "sqrt");
    y = (*sqrt)(x);
    printf ("%f\n", y);
    dlclose(handle);
}
```

Si noti che per essere linkato correttamente, bisogna specificare tra i file da linkare la libreria dinamica `ldl`, per cui il comando `gcc` deve essere del tipo:

```
gcc -o prova prova.c -ldl
```


scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

8.5 Mapping di file in memoria

Il concetto di regione di memoria rischia di rimanere alquanto vago se non si specifica in che modo gli indirizzi logici di una regione sono collegati alle varie parti del file eseguibile. Questo è precisamente il ruolo del cosiddetto *mapping di file in memoria*: associare ad una regione di memoria un file (o parte di esso) residente su disco.

In pratica, la stessa chiamata di sistema `mmap()` utilizzata per creare una regione di memoria prevede tra i suoi parametri un parametro di tipo "file descriptor" che identifica un file aperto in precedenza.

A questo punto, la distinzione tra spazio degli indirizzi e RAM assegnata al processo dovrebbe risultare chiara: a mano a mano che il processo esegue istruzioni, esso fa uso di nuovi indirizzi logici ed è compito del NUCLEO provvedere sia ad assegnare nuova RAM al processo, sia a leggere da disco nell'area di RAM ottenuta i dati associati agli indirizzi logici. Tutto ciò avviene in modo trasparente rispetto al programma: come detto in precedenza, il programmatore non ha alcun controllo sugli indirizzzi fisici usati dal suo programma ed è cura del NUCLEO assegnare (o rilasciare) aree di RAM al processo che esegue il programma.

Come vedremo nel prossimo capitolo, la strategia seguita dal NUCLEO consiste nell'allocare RAM al processo il più tardi possibile, ossia quando il processo effettua indirizzamenti in una zona di regione di memoria non ancora caricata in memoria. Questo approccio è di solito efficace in quanto molti processi eseguono soltanto un sottoinsieme delle istruzioni contenute nel programma.

8.5.1 Altri tipi di mapping

Oltre al mapping di file eseguibili utilizzato dal loader per caricare un programma, la chiamata di sistema `mmap()` è anche usata per realizzare due altri tipi di mapping chiamati *mapping anonimo* e *mapping di file di dati*. Entrambi contribuiscono ad ampliare lo spazio degli indirizzi di un processo.

Mapping anonimo

Due delle quattro regioni di memoria illustrate nella Figura 8.1 sono regioni di memoria particolari che non hanno una immagine su disco, ossia un file di

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

riferimento, da cui estrarre le informazioni richieste: la regione stack e quella usata dallo heap.

In entrambi i casi, le regioni nascono vuote ed è il processo a dovere scrivere dati in esse prima di poterli rileggere: lo stack di un processo è inizialmente vuoto e si riempie a mano a mano che il processo invoca funzioni le quali a loro volta invocano altre funzioni (i compilatori C collocano tutte le variabili locali ad una funzione in cima allo stack). In modo analogo, l'area di memoria dinamica ottenuta tramite la API `malloc()` non è inizializzata.

In questi casi, si parla di mapping anonimo per sottolineare il fatto che la regione di memoria corrispondente non ha un file di riferimento su disco. La distinzione tra mapping standard e mapping anonimo è importante dal punto di vista del NUCLEO: quando un processo effettua una scrittura in una delle sue regioni di memoria, le azioni svolte dal NUCLEO sono diverse a seconda del tipo di mapping.

Mapping di un file di dati

Esiste un altro tipo di mapping non illustrato nella Figura 8.1 che può essere usato dai programmatori: quello relativo al file di dati: tale approccio consente di accedere ai dati di un file come se fossero mappati dentro ad un `array`, anziché essere contenuti all'interno del file. Nel caso di indirizzamenti casuali del file, non è quindi necessario fare uso della chiamata di sistema `lseek()` ma è sufficiente indirizzare la voce richiesta dell'array.

Il seguente esempio illustra il mapping di un file di dati chiamato `/tmp/file0` da parte di un programma.

```
/*  leggi il file di testo esistente "/tmp/file0" da 1 MB
    e modificalo tramite scritture casuali. Il file
    "/tmp/file0" deve essere stato creato in precedenza   */

#include <asm/unistd.h>
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <linux/mman.h>
```


scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

```
#define fd0             3

int main(void)
{
    FILE *f0;
    int i, i1, i2, retcode;
    char *p0;

    f0 = fopen("/tmp/file0", "r+");
    p0 = (char *)mmap(NULL, (1024*1024), PROT_READ|PROT_WRITE,
                      MAP_SHARED, f0, 0);
    for (i=0; i < (1024*1024); i++)
        if (p0[i] == '\n')
            p0[i] = '\n';
        else
            {
            i1 = (i*5)%(1024*1024);
            i2 = (i*7)%(1024*1024);
            p0[i] = (p0[i1] + p0[i2]) / 2;
            }
    retcode = fclose(f0);
}
```


scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

Capitolo 9

STRUTTURA INTERNA DEL NUCLEO

9.1 Introduzione

Dagli anni '60, quando apparvero i primi sistemi operativi multiprogrammati, ad oggi le metodologie di progetto si sono affinate fino a giungere ad un buon livello di standardizzazione per quanto riguarda la realizzazione di NUCLEI per architetture a memoria centralizzata condivisa. Sono invece tuttora in corso ricerche e approfondimenti per quanto riguarda la progettazione di sistemi operativi per architetture multiprocessore (sistemi con più CPU) nonché per architetture distribuite (reti di calcolatori, basi di dati distribuite, ecc.). In questo capitolo, si esaminano brevemente alcune delle problematiche più significative attinenti al progetto di NUCLEI di sistemi operativi multiprogrammati mentre, nel capitolo successivo, di descrivono le funzioni svolte dai principali moduli inclusi nel NUCLEO.

9.2 NUCLEO e processi

L'uso della combinazione NUCLEO/processi discusso nel Pararagrafo 6.3 risulta determinante per realizzare in modo semplice ed elegante sistemi operativi di tipo multitasking.

Come detto in precedenza, il NUCLEO deve essere considerato come uno strato intermedio di software tra i programmi più complessi del sistema operativo (interprete di comandi, compilatori, ecc.) e l'hardware dell'elaboratore.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

In base a questa impostazione modulare, i rimanenti programmi del sistema operativo vedono la combinazione hardware + NUCLEO come una macchina virtuale più facile da programmare della macchina reale. Per questo motivo, il NUCLEO è talvolta considerato come una estensione diretta dell'hardware.

Va tuttavia messo in evidenza che esistono alcuni compiti fondamentali, quelli che garantiscono appunto la corretta gestione dei processi, che devono necessariamente essere realizzati dal NUCLEO in quanto coordinatore dell'avanzamento dei processi. In particolare, citiamo:

- la creazione ed eliminazione di processi;
- la commutazione di processi (in inglese, *task switching*). Tale commutazione consiste nel porre in un opportuno stato di attesa il processo in esecuzione e porre quindi in esecuzione il processo selezionato dalla funzione di scheduling;
- la gestione di tipo time sharing del tempo di CPU con i relativi programmi che realizzano l'algoritmo di scheduling;
- la realizzazione di opportuni schemi di interazione tra processi (vedi Paragrafo 6.3);

Oltre a creare processi, coordinarne l'avanzamento ed eliminarli quando necessario, il NUCLEO deve svolgere numerose altre attività quali:

- la inizializzazione del sistema operativo con relativa impostazione delle strutture di dati necessarie e la creazione dei primi processi di sistema;
- la gestione dei segnali d'interruzione provenienti sia dalla CPU che da dispositivi esterni;
- la chiusura ordinata del sistema operativo che precede lo spegnimento dell'elaboratore con il rilascio di tutte le risorse possedute dai processi e la loro eliminazione.

Accenneremo nel prossimo capitolo al modo in cui NUCLEO svolge i diversi compiti ad esso assegnati. Per ora, limitiamoci a studiare che cosa rende il programma NUCLEO diverso dagli altri programmi finora considerati.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

9.3 Che cosa è il NUCLEO

Come i normali programmi, il NUCLEO deve essere opportunamente compilato e linkato dando luogo ad un file eseguibile. Tale file eseguibile non può tuttavia essere caricao da un loader poiché quest'ultimo presuppone l'esistenza di un NUCLEO. Il file eseguibile contenente il NUCLEO viene quindi caricato in RAM tramite una tecnica chiamata *bootstrapping*: le prime istruzioni eseguite servono a trasferire in RAM settori di disco che contengono altre istruzioni le quali, a loro volta, provvedono a leggere la rimanente parte del codice.

Fatto ciò il NUCLEO inizializza le proprie strutture di dati e crea alcuni processi di servizio, tra cui i processi di tipo "login shell" in grado di accettare comandi dagli utenti. Al termine di tale fase di inizializzazione, il NUCLEO diventa operativo.

Quando il NUCLEO è operativo, esso è pronto sia a soddisfare richieste da parte dei processi in esecuzione, sia a registrare nelle proprie strutture di dati il verificarsi di eventi causati da dispositivi di I/O esterni alla CPU. In altre parole, il NUCLEO è in grado di rispondere in tempi brevi a richieste di servizio provenienti sia dalla CPU che da altri dispositivi hardware.

Per capire come ciò sia possibile, è necessario fare riferimento ai segnali di interruzione e al supporto hardware offerto dal calcolatore per realizzare il modello NUCLEO/processi.

Come illustrato in Figura 9.1, il NUCLEO è un programma di tipo "interrupt driven", ossia un gruppo di programmi in qualche modo indipendenti che sono attivati *soltanto* quando si verificano interruzioni di uno specifico tipo. Ognuno di tali programmi indipendenti possiede un proprio "entry point", ossia l'indirizzo della prima istruzione del programma.

Poiché i moderni calcolatori prevedono decine di interruzioni diverse, vi sono decine di "entry point" tramite i quali possono essere eseguiti i corrispondenti programmi del NUCLEO.

Quando l'unità di controllo della CPU rileva un segnale di interruzione, essa:

1. identifica il numero d'ordine i dell'interruzione;
2. passa in Kernel Mode ed usa lo stack Kernel per salvare il contenuto di alcuni registri del processo sospeso;

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

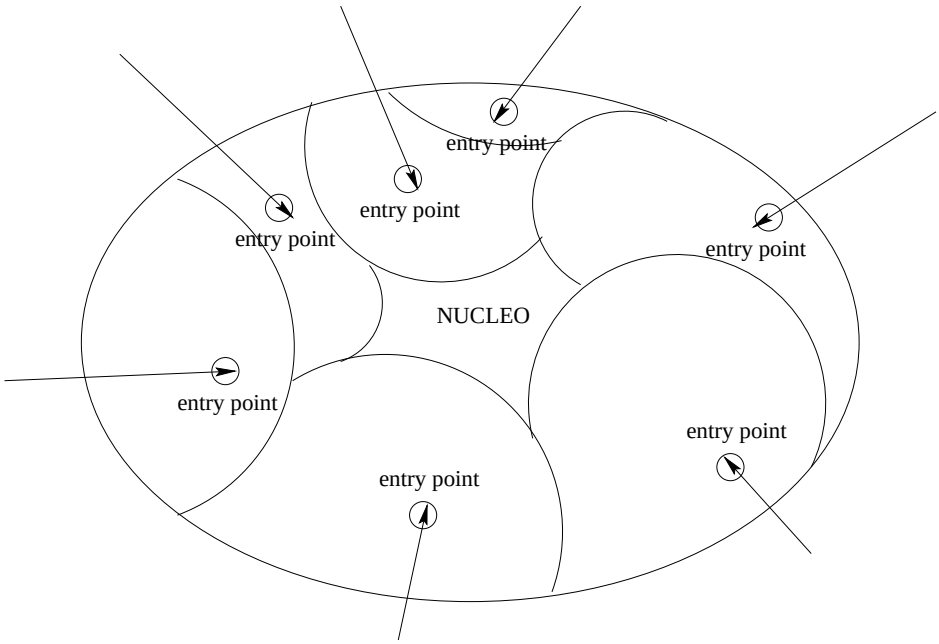

Figura 9.1: Struttura "interrupt driven" del NUCLEO.

3. usa la tabella dei descrittori delle interruzioni ed il numero i per derivare l'indirizzo o "entry point" del programma del NUCLEO atto a trattare interruzioni di tipo i;

4. imposta nel registro contatore programma l'indirizzo derivato al passo precedente.

Al termine delle quattro azioni svolte dall'unità di controllo, la CPU passa ad eseguire la prima istruzione del programma del modulo atto a trattare interruzioni di tipo i. In altre parole, viene eseguito il programma del NUCLEO avente l'"entry point" corrispondente alla interruzione che si è appena verificata.

Per motivi di flessibilità, sia l'indirizzo di base della tabella dei descrittori delle interruzioni, sia i valori degli "entry point" possono essere impostati dal NUCLEO durante la fase di installazione, ossia prima di avere abilitato le interruzioni.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

L'architettura Intel 80x86 include, ad esempio, un registro `idtr` che punta alla base della tabella dei descrittori delle interruzioni. Ogni descrittore lungo 8 byte contiene l'indirizzo logico dell'"entry point" nonché altri flag per la protezione (vedi Manuali Intel per ulteriori dettagli).

9.4 Ruolo dei segnali di interruzione

I segnali di interuzione sono generati sia dall'unità di controllo della CPU, sia da dispositivi esterni alla CPU[1], per rappresentare eventi tra loro molto diversi quali:

- segnalazione da parte di un dispositivo esterno della fine di una operazione di ingresso/uscita;
- segnalazione da parte del chip interval timer: terminazione di un intervallo di tempo prefissato;
- segnalazione da parte dell'unità di controllo della CPU di una condizione anomala verificatasi nell'eseguire una istruzione: codice operativo non valido, indirizzo operando non valido, indirizzo istruzione non valido, ecc.;
- esecuzione di una apposita istruzione il cui effetto è quello di generare un segnale di interruzione (vedi Paragrafo 9.6);
- segnalazione di un evento ad una o più CPU (solo nei sistemi multiprocessor).

9.5 Gestori delle interuzioni

I programmi del NUCLEO che devono trattare specifiche interruzioni prendono il nome di *gestori delle interruzioni* (in inglese, *interrupt handlers*).

In generale, ogni segnale di interruzione richiede un suo apposito gestore: il gestore delle interruzioni emesse periodicamente dal circuito hardware interval timer, ad esempio, non ha niente in comune con quello che gestisce le interruzioni provenienti da tastiera.

[1] La Intel denota come "exception" una segnale di interruzione prodotto dalla CPU e come "interrupt" un segnale prodotto da un dispositivo diverso dalla CPU.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

Come vedremo nel prossimo capitolo su specifici esempi, il verificarsi di un determinato segnale di interruzione può causare cambiamenti di stato tra i processi in vita nel sistema.

Un classico caso è quello delle operazioni di I/O "bloccanti", ossia operazioni che devono essere eseguite prima che il processo possa continuare ad eseguire istruzioni. Se, ad esempio, il processo effettua una chiamata di sistema `read()` per leggere dati da disco, esso dovrà essere bloccato dal NUCLEO fino a quando la lettura dei dati non sarà avvenuta poiché la semantica delle varie forme di `read()` presenti nei linguaggi di programmazione specifica che i dati letti sono immediatamente disponibili per altre elaborazioni da parte del programma.

In questo specifico caso, il segnale di interruzione emesso dal disco per segnalare l'avvenuta lettura dei dati è l'evento atteso dal processo bloccato, per cui tale segnale può essere considerata come una risorsa consumabile prodotta dal controllore del disco e consumata dal processo bloccato sull'operazione di `read()`.

In generale, però, non esiste sempre un processo bloccato per ogni segnale di interruzione che si verifica. Nel caso dei segnali di interruzione emessi periodicamente dall'interval timer, ad esempio, non vi è alcun processo bloccato in attesa di tale evento.

9.6 Chiamate di sistema

La criticità delle funzioni svolte fa sì che il passaggio di controllo ad un qualsiasi programma del NUCLEO non è realizzato come una normale chiamata di procedura, bensì tramite una tecnica speciale che consente non soltanto di passare il controllo ma anche di assicurare la protezione dei programmi e dei dati contenuti nel NUCLEO.

Per distinguerle dalle semplici chiamate di procedura, le chiamate a programmi del NUCLEO prendono il nome di *chiamate di sistema*.

In molte architetture, incluse quelle dei microprocessori più recenti, le chiamate di sistema sono realizzate tramite una apposita istruzione di tipo `int i` (il nome specifico dipende dal tipo di calcolatore utilizzato) che fa passare la CPU dallo stato User allo stato Kernel, generando una apposita interruzione chiamata *interruzione software* o interruzione di programma. Il parametro

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

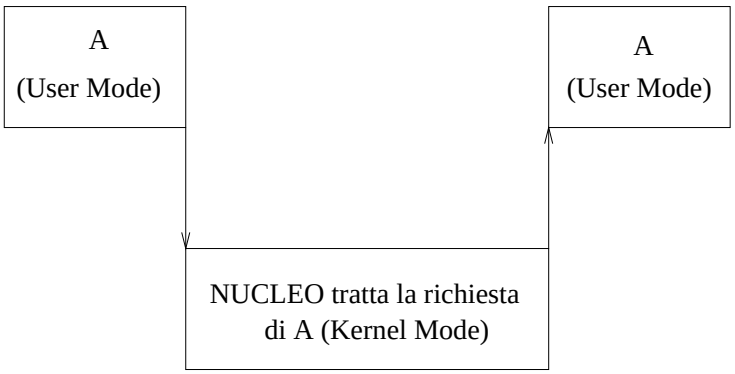

Figura 9.2: Esecuzione di una chiamata di sistema da parte del NUCLEO.

`i` della istruzione `int` identifica la chiamata di sistema (tipicamente viene riservato un byte per tale parametro per cui il NUCLEO è in grado di gestire fino a 256 chiamate di sistema distinte)

In questo modo è assicurata la protezione dei programmi del NUCLEO e delle relative strutture di dati poiché l'unico modo di accedere ad esso da parte di un programma in esecuzione è tramite l'esecuzione della suddetta istruzione `int`.

Quando si verifica una interruzione software, la CPU passa automaticamente ad eseguire le istruzioni del gestore delle chiamate di sistema, ossia dell'apposito programma del NUCLEO abilitato a trattare simili eventi. Il gestore verifica il tipo di chiamata, gli eventuali parametri associati alla chiamata e, nel caso in cui tutto sia regolare, passa il controllo al programma che gestisce la chiamata di sistema avente numero `i`.

Quando il NUCLEO ha eseguito le azioni richieste, esso esegue una apposita istruzione di tipo `rti`, il cui effetto è quello di ripristinare il contesto in stato User del processo che aveva effettuato la chiamata di sistema e di saltare all'indirizzo depositato in precedenza sullo stack Kernel Mode del processo. Dopo avere eseguito la `rti`, la CPU passa quindi ad eseguire l'istruzione successiva alla `int`.

La Figura 9.2 illustra il caso più semplice in cui un processo richiede un servizio al NUCLEO, lo ottiene e riprende quindi la sua esecuzione.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

9.7 Descrittori di risorse

Per svolgere le sue attività, il NUCLEO fa uso di numerosi *descrittori di risorse*, ossia di strutture di dati di vario tipo quali liste, tabelle, vettori di tabelle, ecc.

Esempi classici di descrittori si riferiscono alle seguenti risorse:

- *processo*: ogni processo gestito dal NUCLEO possiede un proprio descrittore che deve essere aggiornato ogni qualvolta il processo cambia stato.
- *memoria RAM*: ogni blocco (page frame) di memoria possiede un descrittore che specifica l'eventuale uso da parte di uno o più processi (nel caso di blocchi condivisi)
- *spazio degli indirizzi di un processo*: ogni processo è abilitato a fare uso di uno specifico insieme di indirizzi lineari, a prescindere dal fatto che le pagine di dati associate a tali indirizzi siano effettivamente presenti in RAM. Come abbiamo visto nel Paragrafo 8.2, il NUCLEO fa uso di un apposito descrittore per ogni regione di memoria assegnata ad un processo.
- *memoria secondaria*: il NUCLEO mantiene per ogni file system montato alcuni descrittori che rappresentano il contenuto del disco; anche se tali descrittori sono già presenti sul disco stesso, risulta molto più efficiente farne una copia in memoria per ridurre il numero di accessi al disco. Nei file system Unix, tali descrittori prendono il nome di "superblocco", "descrittori di gruppi di blocco" e "inode".
- *file aperti da un processo*: l'interazione tra un processo ed un file richiede l'uso di altri descrittori che contengoono diverse informazioni quali il modo in cui è stato aperto il file, il byte del file attualmente scandito dal processo, ecc. Nei file system Unix, tali descrittori prendono il nome di "file object".

A prescindere dal modo in cui sono realizzati, i descrittori di risorsa hanno alcune importanti proprietà in comune.

- ogni descrittore è memorizzato in RAM in un'area di memoria riservata al NUCLEO: per motivi di sicurezza, i processi non devono potere accedere direttamente ad un descrittore di risorsa.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

- ogni descrittore utilizzato dal NUCLEO deve essere opportunamente inizializzato prima di poterlo utilizzare. In generale, durante l'inizializzazione del sistema operativo, vengono riservate le aree di memoria destinate a contenere i vari descrittori e vengono inizializzati opportunamente i campi di ogni descrittore.

- i descrittori sono frequentemente aggiornati dai vari programmi del NUCLEO: di norma, ognuno di essi è considerato come una risorsa seriale che deve essere aggiornata in modo non interrompibile da parte del NUCLEO (vedi paragrafo successivo).

Un esempio può servire a rendere più intuitivo l'ultimo punto: il modo più semplice per codificare lo stato di un processo è di fare uso di apposite liste di descrittori di processo, una per ognuno dei vari stati riconosciuti dal NUCLEO.

Consideriamo ad esempio la lista dei processieseguibili, ossia pronti ad essere eseguiti dalla CPU. Ogni qualvolta la funzione di fine quanto di tempo viene invocata, essa scandisce tale lista per inserire nella posizione appropriata il processo che ha terminato il suo quanto di tempo. Durante tale fase, il NUCLEO non deve essere interrotto finché il descrittore di processo non sia stato inserito. Nel caso opposto, infatti, la seconda attivazione del NUCLEO potrebbe richiedere l'esecuzione di un programma che consulta la stessa lista e vi è la possibilità che essa sia stata lasciata in uno stato non coerente dalla precedente attivazione: ad esempio alcuni dei puntatori utilizzati per realizzare la lista potrebbero non essere stati ancora aggiornati.

9.8 Interrompibilità del NUCLEO

Per motivi di semplicità, l'interazione processo-NUCLEO è stata presentata come una attività sequenziale consistente nei seguenti tre passi:

1. il processo attiva il NUCLEO tramite `int`;

2. la CPU passa in stato Kernel ed il NUCLEO esegue gli opportuni programmi;

3. al termine, il NUCLEO rilascia l'uso della CPU ed il processo riprende l'esecuzione in stato User.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

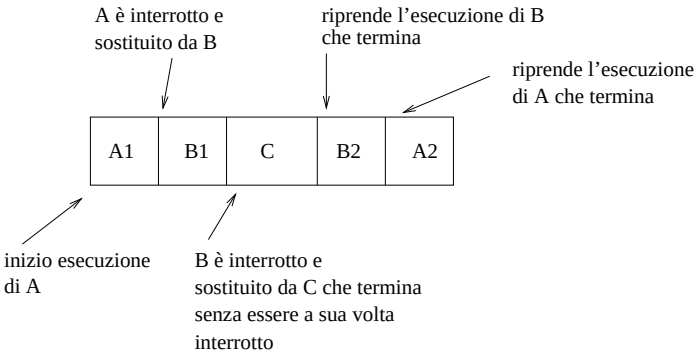

Figura 9.3: Esecuzione annidata di programmi del NUCLEO.

La realtà è molto più complessa poiché il NUCLEO non deve soltanto soddisfare richieste emesse dal processo in esecuzione, ma anche gestire interruzioni di vario tipo quali quelle emesse dai dispositivi di I/O e quelle emesse dal chip interval timer.

In effetti, il NUCLEO non ha una struttura sequenziale come molti programmi, bensì parallela: ogni segnale d'interruzione riconosciuto dall'hardware e gestito dal sistema operativo causa l'attivazione di un apposito programma del NUCLEO. Tale struttura parallela assicura a sua volta la interrompibilità del NUCLEO, ossia la capacità di rispondere ad un segnale di interruzione mentre sta già trattando un altro segnale di interruzione (sia pure di tipo diverso).

Ogni NUCLEO interrompibile deve quindi essere in grado di eseguire in modo annidatole interruzioni, ossia deve essere in grado di sospendere l'esecuzione di un programma di gestione di una interruzione per passare ad eseguire un altro programma di gestione di una interruzione di tipo diverso; terminata l'esecuzione del secondo programma, il NUCLEO deve riprendere l'esecuzione del primo programma.

La Figura 9.3 illustra un esempio di esecuzione annidata di tre programmi distinti del NUCLEO.

In effetti, l'attività più complessa nella progettazione di un NUCLEO consiste nel renderlo il più possibile interrompibile: in realtà, non sarà mai possibile renderlo totalmente interrompibile in quanto alcune operazioni critiche devono essere eseguite con le interruzioni disabilitate e con la certezza che il

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

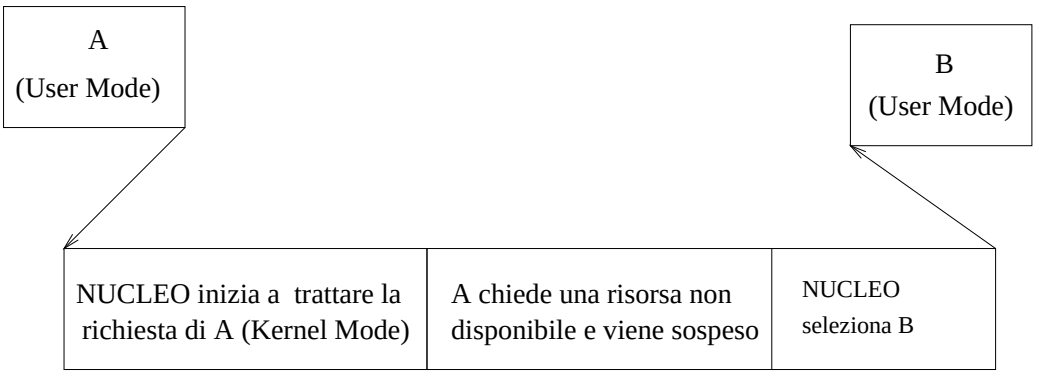

Figura 9.4: Chiamata di sistema che pone il processo in uno stato di attesa.

NUCLEO non verrà interrotto mentre esegue tali operazioni (vedi esempio precedente relativo all'inserimento di un descrittore di processo nella lista dei processi eseguibili).

Per esemplificare quanto appena detto, diamo alcuni esempi dei casi che possono verificarsi in seguito alla attivazione del NUCLEO tramite una qualche interruzione.

Nel caso più semplice, il programma del NUCLEO termina e riprende in stato User l'esecuzione del processo che ha eseguito la `int` (vedi Figura 9.2).

In altri casi, il programma del NUCLEO può richiedere una risorsa attualmente non disponibile, ad esempio, la lettura di un carattere da tastiera che non è stato ancora digitato dall'utente. Quando ciò si verifica, il NUCLEO provvede porre il processo in stato di attesa ed a selezionare un altro processo da porre nello stato di esecuzione (vedi Figura 9.4).

In altri casi ancora, un processo in stato User può essere interrotto per gestire una interruzione di I/O relativa ad un altro processo; inoltre, mentre NUCLEO gestisce tale interruzione, si verifica un'altra interruzione di I/O di tipo diverso relativa ad un terzo processo avente una elevata priorità e tale interruzione fa passare il terzo processo nello stato di eseguibile: in questo caso, il NUCLEO può decidere di rimettere in esecuzione il terzo processo lasciando incompiuto il trattamento della prima interruzione (vedi Figura 9.5). Non tutti i NUCLEI tra quelli più diffusi sono in grado di effettuare tale tipo di "preemption"; quelli in grado di effettuarla sono chiamati "premptive kernel".

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

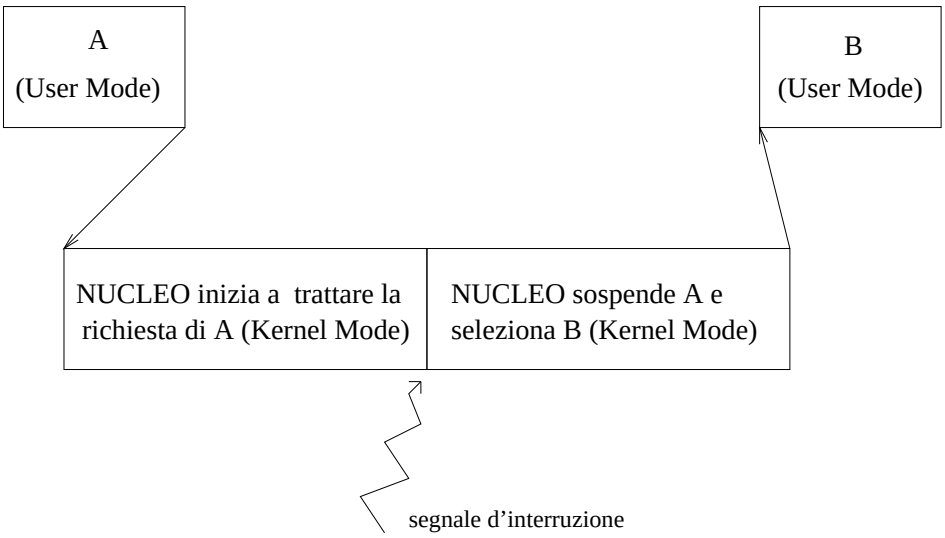

Figura 9.5: Esecuzione di un processo prioritario con gestione interruzione incompiuta.

Gli esempi illustrati non sono in alcun modo esaurienti. Sono molto numerose infatti le possibili combinazioni di interazioni che devono essere considerate nel progettare un NUCLEO interrompibile e non è possibile elencarle tutte.

9.9 Processo o NUCLEO?

Abbiamo elencato nel Paragrafo 9.2 alcune funzioni che debbono necessariamente essere realizzate tramite programmi del NUCLEO ma non abbiamo specificato se altre importanti funzioni del sistema operativo debbano essere realizzate all'interno del NUCLEO, oppure tramite appositi processi. In effetti, esistono a tale riguardo due approcci diversi che portano a realizzare, rispettivamente, micro-NUCLEI oppure NUCLEI monolitici. Vediamo di cosa di tratta.

Il primo approccio privilegia l'uso di processi per realizzare il massimo numero di funzioni possibili del sistema operativo, facendo uso di un NUCLEO ridotto chiamato *micro-NUCLEO* (in inglese, *microkernel*) che svolge pochi compiti essenziali quali la commutazione di processi ed una gestione semplificata delle interruzioni: nella maggior parte dei casi, il micro-NUCLEO

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

riceve il segnale di interruzione ed invia un opportuno messaggio al processo interessato.

Le principali funzioni del sistema operativo vengono svolte da processi specializzati: esiste, ad esempio, un processo per lo scheduling, un processo per la gestione del file sistem, un processo per il controllo dei diritti d'accesso dei processi utente, un processo per la gestione delle aree di memoria RAM e così via.

Seguendo tale approccio, quando un processo utente intende richiedere un qualche servizio da parte del sistema operativo, esso non esegue una chiamata di sistema al NUCLEO ma invia un opportuno messaggio al processo di sistema gestore del servizio richiesto. In particolare, il processo richiedente esegue i seguenti passi:

1. esegue una chiamata di sistema di tipo `send()` (vedi Paragrafo 7.3.6) al processo gestore specificando nel messaggio i parametri associati alla richiesta;

2. si pone in attesa del messaggio di terminazione tramite una `receive()`.

I processi di sistema hanno una struttura ciclica: usano una primitiva di sincronizzazione di tipo `receive()` per verificare se vi sono messaggi, ossia richieste di servizio; se vi è almeno un messaggio, passa ad eseguire la richiesta, invia un messaggio di terminazione (vedi Figura 9.6) al processo richiedente e torna ad eseguire una `receive()`. In pratica, viene realizzata una architettura software di tipo "client/server" dove la maggior parte delle funzioni del sistema operativo sono svolte da processi di sistema di tipo server.

I vantaggi derivanti da tale impostazione sono numerosi:

- il sistema operativo risulta modulare e facile da mantenere ed aggiornare;

- la struttura del NUCLEO risulta estremamente semplificata; in particolare, scompare la necessità di rendere il NUCLEO interrompibile poiché esso è in grado di trattare ogni richiesta in tempi ridotti;

- l'uso di processi di sistema rende più agevole i controlli sulla validità delle richieste; se la richiesta è di tipo gravoso, il server può decidere di creare più processi figli per poterla trattare in parallelo.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

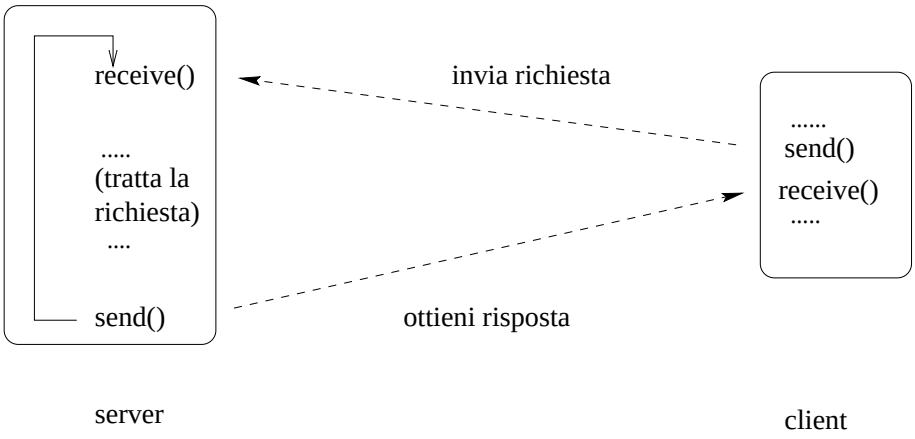

Figura 9.6: Interazione client/server tra processo utente e processo di sistema.

Purtroppo, esiste una grossa limitazione a tale approccio che è quello delle prestazioni: l'introduzione di numerosi processi di sistema fa aumentare notevolmente il numero di commutazioni di processo ed inoltre la comunicazione tra processi realizzata tramite chiamate di sistema di tipo `send()` e `receive()` richiede frequenti interventi del NUCLEO. I risultati sono quindi deludenti; oggi esistono pochi sistemi operativi commerciali basati sull'uso di micro-NUCLEI: il più noto è il sistema Next derivato dal prototipo Mach realizzato presso la Carnegie Mellon University.

Passiamo ora considerare un approccio diametralmente opposto a quello precedente: quello del *NUCLEO monolitico* che include la maggior parte delle funzionalità del sistema operativo all'interno del NUCLEO. In base a tale approccio, il NUCLEO include un elevato numero di programmi in grado di trattare decine o centinaia di chiamate di sistema diverse, che corrispondono ai diversi servizi offerti.

Esistono, ad esempio, chiamate per la gestione del file system, chiamate per la sincronizzazione di processi, chiamate per la richiesta di aree di memoria e così via. Alcune chiamate di sistema possono richiedere tempi di esecuzione elevati, per cui è indispensabile progettare il NUCLEO monolitico in modo interrompibile.

Non esistono molti punti a favore del NUCLEO monolitico se non quello cruciale dell'efficienza. Molti programmi del sistema operativo devono essere

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

considerati alla stregua di programmi in tempo reale: possono essere eseguiti anche migliaia di volte al secondo ed il modo in cui vengono realizzati condiziona le prestazioni dell'intero sistema. Oggi, i più noti sistemi operativi sono realizzati facendo uso di NUCLEI monolitici. In alcuni casi, sono utilizzati un numero limitato di processi di sistema per alleggerire in parte la struttura del NUCLEO ma esso rimane comunque il componente del sistema operativo più complesso.

9.10 Thread o processi leggeri?

I processi descritti in precedenza sono una utile astrazione che consente di strutturare in modo ottimale il NUCLEO di un sistema multitasking. Ciò nonostante, il loro uso presenta alcuni inconvenienti dovuti alla ricchezza di informazioni associata ad ogni descrittore di processo.

In effetti, per effettuare una commutazione di processi, il NUCLEO deve salvare tutte le informazioni richieste nel descrittore del processo precedentemente in esecuzione e prelevare dal descrittore del nuovo processo le corrispondenti informazioni per registrarle nei registri del processore e nelle varie strutture di dati del sistema operativo. Tale operazione chiamata commutazione di processi (in inglese, process switching) può risultare, nel caso di sistemi in tempo reale, eccessivamente costosa in termini di tempo di esecuzione. Per questo motivo, alcuni sistemi operativi fanno uso di un modello di esecuzione più complesso di quello considerato finora.

In tale modello, ogni processo è composto da uno o più *thread*. Ogni thread associato ad un processo condivide l'area di memoria assegnata al processo nonché i file da esso aperti.

Il descrittore di un singolo thread si limita quindi al cosiddetto "contesto di esecuzione", ossia al contenuto dei registri del processore ed a quello dello stack usato dal thread. Lo scheduling (vedi Paragrafo 6.6) non viene più effettuato tra processi, bensì tra thread ed, in generale, il NUCLEO tratta in modo paritetico i thread di uno stesso processo e quelli appartenenti a processi diversi quando deve selezionare il thread più adatto a cui assegnare la CPU.

Dal punto di vista realizzativo, è necessario per potere gestire thread introdurre appositi descrittori che si affiancano ai descrittori di processo; tali descrittori includono pochi campi, tra cui un identificatore di thread, un

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

riferimento al processo a cui appartengono ed un "contesto di esecuzione", ossia uno stack ed un'area di salvataggio dei registri della CPU.

Il tempo richiesto dal NUCLEO per effettuare la commutazione tra due thread risulta solitamente minore di quello richiesto per effettuare la commutazione tra due processi. I thread rappresentano quindi una valida scelta per realizzare applicazioni che condividono strutture di dati comuni e che sono eseguite su macchine dotate di più processori.

Linux non fa uso di thread ma utilizza un approccio diverso per ridurre il costo della commutazione di processi basato sui *processi leggeri*: tali processi condividono buona parte delle risorse possedute da un processo, tra cui lo spazio degli indirizzi e i descrittori di file aperti. La creazione di un processo leggero è più rapida di quella di un processo classico poiché è sufficiente copiare buona parte dei campi puntatore a risorse del processo padre in quelli del processo figlio.

La struttura del NUCLEO non risulta appesantita da un ulteriore tipo descrittore ed è possibile realizzare applicazioni parallele basate su processi leggeri che non hanno nulla da invidiare, in termini di efficienza, a quelle basate sui thread. Attualmente, esistono NUCLEI che sopportano i thread ed altri che sopportano i processi leggeri e non sembrano esservi differenze significative in termini di prestazioni.

9.11 Quale linguaggio di programmazione?

La scelta dei linguaggi di programmazione utilizzati per realizzare il sistema operativo ha una sua importanza in quanto condiziona l'efficienza dell'intero sistema. Alcuni linguaggi di programmazione, infatti, includono dei meccanismi per la chiamata di procedure e per il controllo automatico degli indirizzamenti che rendono il codice macchina generato poco efficiente.

Come già sottolineato in precedenza, la parte critica del sistema operativo è costituita dal NUCLEO, mentre per le altre parti la scelta del linguaggio di programmazione non è determinante ai fini dell'efficienza del sistema. Per quanto riguarda la programmazione del NUCLEO, si assiste ad una evoluzione molto lenta verso linguaggi ad alto livello.

Fino agli anni '70, tutti i NUCLEI erano codificati in linguaggio assemblativo: ad esempio, i NUCLEI dei sistemi operativi OS/360 e successori usati per

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

i mainframe IBM e quello del sistema operativo VAX/VMS della Digital Equipment sono stati codificati interamente in linguaggio assemblativo.

È opportuno precisare che una piccola parte del NUCLEO, quella che riguarda le interazioni con l'hardware, deve *necessariamente* essere codificata nel linguaggio assemblativo per potere utilizzare le apposite istruzioni che operano sui registri della macchina.

Quando, ad esempio, il NUCLEO effettua una commutazione di processi, deve salvare il contenuto dei registri utilizzati dal processo sospeso in una apposita area di memoria (tipicamente, nel descrittore di processo e nello stack Kernel Mode del processo). Per fare ciò, serve un linguaggio di programmazione in grado di indirizzare specifici registri hardware della CPU, ossia serve il linguaggio assemblativo.

Si può stimare tra il 5 e il 10% la percentuale di codice del NUCLEO basata sull'architettura del calcolatore. La rimanente parte del NUCLEO tuttavia può essere scritta in un linguaggio ad alto livello, purché efficiente.

Unix è stato il primo sistema operativo che ha fatto un uso intensivo di un linguaggio ad alto livello per codificare la maggior parte dei programmi del NUCLEO; in effetti, i progettisti di Unix hanno messo a punto un apposito linguaggio, il linguaggio C, per svolgere tale compito.

Dopo alcuni decenni, la situazione non è molto cambiata: molti NUCLEI quali quelli realizzati per i nuovi sistemi Unix continuano ad essere realizzati nel linguaggio C con alcune parti in linguaggio assemblativo. Una innovazione significativa rispetto ai sistemi precedenti consiste nel separare in modo netto le parti del NUCLEO indipendenti dall'architettura da quelle dipendenti da essa. In tale modo risulta molto facilitata la portabilità del NUCLEO da una piattaforma hardware ad una altra.

Perfino un concetto innovativo come quello della programmazione ad oggetti che ha avuto un notevole impatto nella realizzazione di programmi applicativi ha incontrato un limitato successo tra i progettisti di NUCLEI, anche se alcuni sistemi operativi quali Windows NT fanno del linguaggio C++ per codificare alcune funzioni meno critiche del sistema operativo. Il motivo è sempre lo stesso, ossia l'inefficienza dei compilatori per linguaggi ad oggetto quale il C++ rispetto ai compilatori per linguaggi procedurali tradizionali.

L'unica piccola innovazione dal punto di vista delle tecniche di programmazione è consistita nel fare uso di un limitato numero di oggetti realizzati in C tramite strutture che contengono sia dati che puntatori a funzioni.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

Un discorso a parte vale per quelle parti del sistema operativo realizzate tramite processi: poiché esse sono meno critiche dal punto di vista delle prestazioni, l'uso di linguaggi orientati ad oggetto può essere giustificato.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

Capitolo 10

GESTIONE DELLA MEMORIA

10.1 Introduzione

Un compito importante del NUCLEO è quello di gestire le aree di memoria presenti nel calcolatore: la memoria principale (RAM) e la memoria secondaria (dischi magnetici).

Come è ovvio, la gestione della RAM è più critica di quella delle aree di disco, per cui le tecniche di gestione utilizzate sono alquanto diverse. Inoltre, le richieste di RAM da parte di funzioni del NUCLEO sono considerate prioritarie rispetto alle richieste di RAM effettuate da un processo in User Mode.

Per questi motivi, descriveremo in questo capitolo due classiche tecniche di gestione della memoria RAM utilizzate, rispettivamente, per soddisfare:

- assegnazione o rilascio di aree di RAM richieste da funzioni del NUCLEO;
- assegnazione di aree di RAM ad un processo in User Mode.

Descriveremo inoltre una tecnica di gestione dello spazio su disco utilizzata per soddisfare:

- richieste di modifica di dimensioni dei file emesse dal file system;
- richieste effettuate da altri componenti del NUCLEO per ottenere o rilasciare blocchi di disco.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

10.2 Indirizzamento della RAM

Una premessa è d'obbligo prima di iniziare la discussione: gli accessi alla RAM da parte di un processo sono numerosissimi, dato che ogni nuova istruzione da eseguire va letta dalla RAM e che, in alcuni casi, l'esecuzione dell'istruzione richiede ulteriori accessi alla RAM per leggere o scrivere operandi. Per questo motivo, non è possibile pensare a schemi di gestione interamente software ma è necessario ricorrere ad appositi circuiti hardware che consentano determinati tipi di gestione senza aumentare in modo significativo il tempo d'accesso alla RAM che è oggi dell'ordine di poche decine di miliardesimi di secondo.

Un'altra importante considerazione è che la gestione multitasking, più volte citata in precedenza, richiede l'uso di una tecnica di suddivisione della memoria RAM. Infatti, tale tipo di gestione presuppone che, ad ogni istante, vi siano più programmi indipendenti caricati in memoria.

Poiché i programmi hanno tempi di esecuzione diversi e lunghezze diverse, nasce il problema di definire tecniche di indirizzamento della RAM che consentano da un lato di caricare agevolmente nuovi programmi in memoria e, dall'altro, di utilizzare efficientemente la memoria a disposizione.

10.2.1 Paginazione

La *paginazione* è una tecnica di indirizzamento della RAM che fa uso di sofisticati circuiti di traduzione degli indirizzi. Nel descrivere tale tecnica, faremo uso della seguente terminologia:

- *indirizzo logico*: numero da 32 bit usato nelle istruzioni in linguaggio macchina per identificare l'indirizzo della prossima istruzione da eseguire, oppure l'indirizzo di un operando richiesto dall'istruzione;
- *indirizzo fisico*: numero da 32 bit usato dal bus per indirizzare byte di memoria RAM contenuta in appositi chip.

Ogni indirizzo logico è tradotto dal circuito di paginazione in un indirizzo fisico. Come vedremo appresso, indirizzi logici contigui possono essere tradotti in indirizzi fisici non contigui. La memoria RAM disponibile è suddivisa in N blocchi (in inglese, *page frame*) di lunghezza fissa, ad esempio 4096 byte (sono anche usati blocchi di dimensioni maggiori).

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

L'informazione contenuta in uno di tali blocchi prende il nome di *pagina*. La maggior parte delle attuali CPU utilizzano indirizzi logici da 32 bit. In tale caso, ogni indirizzo logico è decomposto in un numero di pagina (i 20 bit più significativi) e un indirizzo relativo nella pagina (i rimanenti 12 bit).

Per ridurre la dimensione delle tabelle, i 20 bit che esprimono il numero di pagina sono decomposti ulteriormente in due gruppi di 10 bit: il primo gruppo indica la posizione nella tabella principale chiamata *Page Directory*, mentre il secondo gruppo indica la posizione in una delle tabelle secondarie chiamate *Page Table*.

La trasformazione dell'indirizzo logico in un indirizzo fisico è effettuata dal circuito di paginazione nel seguente modo (vedi Figura 10.1):

1. ottieni da un apposito registro della CPU l'indirizzo della PageDirectory del processo in esecuzione (nei microprocessori Intel 80x86, tale registro è chiamato `cr3`);
2. usa i 10 bit più significativi dell'indirizzo logico per selezionare la voce opportuna nella Page Directory;
3. leggi il contenuto della voce selezionata al passo precedente per indirizzare la Page Table contenente la pagina da indirizzare;
4. usa i 10 bit intermedi dell'indirizzo logico per selezionare la voce opportuna della Page Table;
5. somma i 12 bit meno significativi dell'indirizzo logico all'indirizzo fisico della page frame di RAM ottenuta al passo precedente ottenendo così l'indirizzo fisico desiderato.

Ogni voce di una Page Directory o di una Page Table occupa 32 bit di cui:

- 20 bit contengono l'indirizzo fisico di una page frame (poiché le page frame sono lunghe $2^{12} = 4096$ byte, non è necessario specificare i 12bit meno significativi dell'indirizzo fisico);
- 12 bit contengono diversi flag che caratterizzano lo stato della page frame ed i suoi diritti d'accesso. Tipicamente, sono usati i seguenti flag:

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

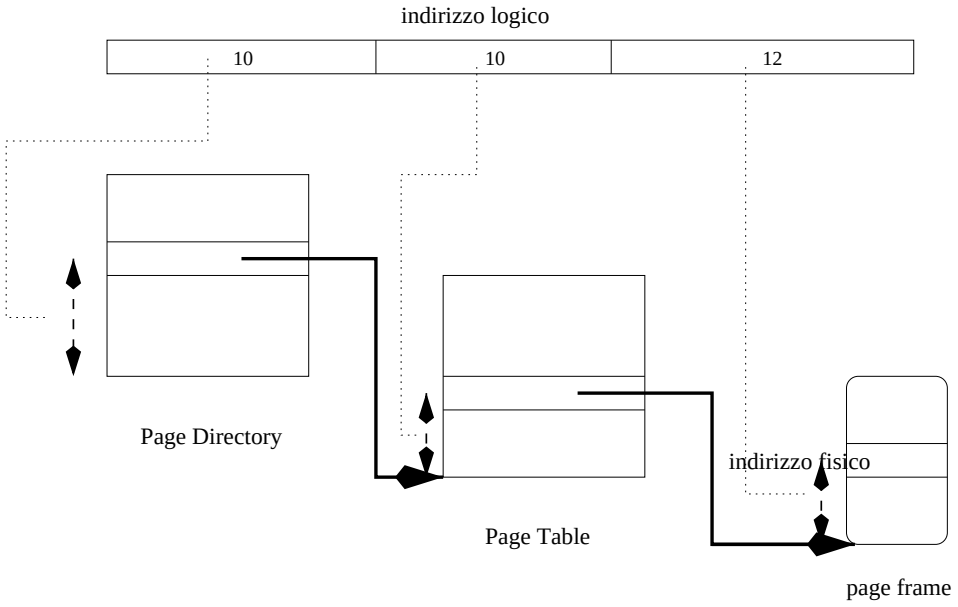

Figura 10.1: Trasformazione di un indirizzo logico in un indirizzo fisico.

- flag Present: vale 0 se la pagina non è attualmente presente in RAM;
- flag User/Supervisor: vale 0 se la pagina può essere indirizzata solo quando la CPU è in Kernel Mode;
- flag Execute: vale 1 se la CPU è autorizzata a prelevare istruzioni dalla pagina;
- flag Read: vale 1 se la CPU è autorizzata a leggere dati dalla pagina;
- flag Write: vale 1 se la CPU è autorizzata a scrivere dati nella pagina.

Se il flag Present vale 0, oppure se i diritti d'accesso non consentono l'indirizzamento richiesto dalla CPU, il circuito di paginazione genera una eccezione di tipo Page Fault che attiva un apposito programma del NUCLEO.

Nell'accedere ad una voce di Page Directory o di Page Table, il circuito di paginazione verifica inoltre se la voce è nulla (tutti i 32 bit hanno il valore

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

0). Anche in questo caso, il circuito di paginazione genera una eccezione di tipo Page Fault.

Prima di generare una eccezione, il circuito di paginazione salva in un apposito registro (registro `cr4` nel caso dei microprocessori Intel 80x86) l'indirizzo logico che ha causato l'eccezione.

Potrebbe sembrare a prima vista che l'uso della paginazione degradi sensibilmente le prestazioni del calcolatore: ogni accesso da parte della CPU ad un dato oppure ad una istruzione contenuti nella RAM richiede tre indirizzamenti consecutivi alla RAM anziché uno solo!

In pratica, sono usati vari accorgimenti per ridurre il tempo di consultazione delle tabelle di traduzione. Oggi tutti i chip CPU includono memorie cache veloci di dimensioni ridotte in cui vengono conservati le coppie di valori (indirizzo logico, voce di Page Table o Page Directory) degli ultimi indirizzamenti effettuati. Grazie a tali cache, non è necessario accedere alla RAM per leggere una voce di Page Directory o di Page Table in quanto tale informazione è già presente nella cache, per cui il tempo medio di accesso alla RAM non si scosta significativamente da quello richiesto per effettuare un singolo accesso.

È importante sottolineare come la paginazione consenta di proteggere efficacemente gli spazi degli indirizzi dei vari processi: in effetti, un processo non può accedere a page frame diverse da quelle assegnateli dal sistema operativo, se non modificando le proprie tabelle di paginazione. Poiché tali tabelle sono contenute nelle page frame riservate al NUCLEO (flag User/Supervisor impostato a 0), esse non possono però essere modificate da un processo in stato User e la protezione è quindi assicurata.

10.2.2 Indirizzi logici riservati ai programmi del NUCLEO

Ogni processo possiede una propria Page Directory ed un gruppo di Page Table. Grazie a tale approccio, gli stessi indirizzi logici possono essere usati da più processi senza che ciò causi alcuna ambiguità poiché la traduzione da indirizzo logico ad indirizzo fisico avviene utilizzando le tabelle di paginazione del processo che richiede l'indirizzamento.

Il NUCLEO invece non è un processo e sarebbe alquanto macchinoso assegnarli apposite tabelle di paginazione. La soluzione preferita consiste invece nel riservare un parte degli indirizzi logici per i programmi del NUCLEO

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

ed includere nelle tabelle di paginazione di ogni processo un mapping tra gli indirizzi logici riservati al NUCLEO e gli indirizzi fisici della RAM installata.

Vediamo come ciò viene realizzato su un esempio concreto. In Linux, l'intervallo di 4 Giga (2^{32}) corrispondente a tutti i possibili indirizzi logici viene suddiviso in due intervalli:

- un intervallo da 0 a 3 Giga meno 1 ($3 \times 2^{30} - 1$) riservato ai processi che eseguono in User Mode;
- un intervallo da 3 Giga a 4 Giga meno 1 ($4 \times 2^{30} - 1$) riservato ai processi che eseguono in Kernel Mode, ossia ai programmi del NUCLEO.

Poiché ogni Page Directory include 1024 voci, ciò significa che le prime 768 voci sono riservate ai processi che eseguono in User Mode, mentre le ultime 256 sono riservate ai programmi del NUCLEO.

Il numero di indirizzi logici a disposizione del NUCLEO è più che sufficiente a mappare tutta la RAM installata, per cui buona parte delle ultime 256 voci della Page Directory sono nulle per indicare che i corrsipondenti indirizzi logici non sono validi in uqanto nojn vi è RAM associata ad essi.
È importante osservare che iul mapping tra indirizzi logici usati dal NUCLEO e indirizzi fisici è una semplice trasformazione lineare:

$$indirizzofisico = indirizzologico - c0000000$$

Ovviamente tale mapping vale solo per i programi del NUCLEO e non per i processi in User Mode. Come vedremo nel prossimo paragrafo, il NUCLEO utilizza un algoritmo di gestione della memoria che assegna aree contigue di RAM, per cui non è necessario modificare il contenuto di alcuna delle ultime 256 voci della Page Directory in seguito ad una assegnazione o rilascio di memoria.

10.3 L'algoritmo Buddy System

Vediamo ora come il NUCLEO gestisce la RAM facendo uso dei circuiti di paginazione. Come è ovvio, l'unità minima di allocazione è la page frame, ossia 4096 byte.

In primo luogo, il NUCLEO deve riservare, durante la fase di inizializzazione, un sufficiente numero di page frame per contenere il codice e le strutture

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

di dati statiche. Le page frame rimanenti prendono il nome di *memoria dinamica* e possono essere allocate e rilasciate dinamicamente per soddisfare le esigenze delle varie funzioni del NUCLEO.

In effetti, oltre a poche strutture statiche di dati, il NUCLEO fa uso di numerose strutture dinamiche di dati che vengono create e successivamente eliminate. Diamo alcuni esempi di tali strutture:

- descrittore di processo: quando viene creato un nuovo processo, è necessario assegnare una nuova area di memoria per contenere il descrittore di processo; quando il processo viene eliminato, tale area di memoria può essere riutilizzata;
- inode: quando viene aperto un file per la prima volta, il file system copia da disco il descrittore del file in una struttura chiamata inode; l'area di memoria usata da un inode puó essere rilasciata quando non vi sono più processi che hanno aperto quel file;
- messaggi: le chiamate di sistema di tipo `sendmessage()` richiedono aree di memoria per contenere i messaggi pendenti destinati ai vari processi; tali aree di memoria possono essere rilasciate quando i processi destinatari hanno "consumato" i messaggi.

Tipicamente, il NUCLEO centralizza la gestione della RAM dinamica tramite due funzioni del tipo `get_mem()` e `free_mem()` che possono essere usate dalle varie funzioni del NUCLEO per ottenere o rilasciare page frame.

Per evitare problemi di frammentazione della memoria libera, è molto usato l'algoritmo *Buddy System* che tenta di accorpare blocchi contigui di page frame libere creando così un singolo blocco libero di dimensioni maggiori. Vediamo in dettaglio come opera tale algoritmo.

L'algoritmo gestisce blocchi di page frame di dimensioni diverse, ad esempio blocchi da 1, 2, 4 ,8, 16, 32, 64, ..., 512 page frame, per cui la funzione `get_mem()` prevede un parametro che specifica la dimensione del blocco richiesto.

Per ogni tipo di blocco, il Buddy System gestisce un apposito descrittore che specifica gli indirizzi dei blocchi liberi e quelli dei blocchi occupati. Se consideriamo ad esempio i blocchi da 64 page frame, l'intera RAM verrà vista come un vettore di bit dove ogni bit denota 64×4 KB di RAM; il bit vale 0 se il blocco è libero, 1 se è occupato.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

Nel soddisfare una richiesta per un blocco di dimensione k, l'algoritmo Buddy System consulta dapprima il descrittore per blocchi di dimensione k: se vi è almeno un bit uguale a 0, pone tale bit a 1 e ritorna l'indirizzo iniziale del blocco appena individuato. Se non vi sono bit uguali a 0, consulta il descrittore per blocchi di dimensione $2 \times k$ e così via fino a trovare il primo descrittore che include un blocco libero.

Se non vi è alcun descrittore contenente un bit uguale a 0, la richiesta di memoria non può essere soddisfatta e la funzione `get_mem()` ritorna un codice di errore. Altrimenti, viene selezionato il primo descrittore avente un bit uguale a 0 ed avviene la suddivisione del blocco in un blocco occupato ed uno o più blocchi liberi.

Facciamo un esempio specifico per meglio chiarire come ciò si verifica. Supponiamo che la richiesta era per 64 page frame e che il blocco libero più piccolo occupa 256 page frame. In questo caso, il blocco libero da 256 page frame viene dichiarato occupato e viene suddiviso in un blocco occupato da 64 page frame, un blocco libero da 64 page frame ed un blocco libero da 128 page frame. I relativi descrittori di blocchi da 64, 128 e 256 page frame vengono opportunamente modificati.

Le rilasciare un blocco, l'algoritmo Buddy System verifica se il blocco "gemello" (in inglese "buddy") di quello che si è appena liberato è anch'esso libero. Nel caso affermativo, fonde i due blocchi liberi buddy in un unico blocco libero di dimensione doppia rispetto a quella del blocco che si è reso libero.

Anche in questo caso, facciamo un esempio specifico per meglio chiarire cosa si intende per coppia di blocchi buddy. Consideriamo i blocchi da 16 K, ossia i blocchi da 4 page frame. Gli indirizzi fisici associati a tali blocchi sono multipli di 16 K: $0, 16, 32, 48, 64, 80, \ldots$ I blocchi buddy di tale gruppo sono 0 con 16, 32 con 48, 64 con 80 e così via. In effetti, fondendo il blocco 64 con quello 80 si ottiene un blocco da 32 K avente indirizzo iniziale multiplo di (32×4) K. Viceversa, due blocchi contigui da 16 K quali il blocco 48 e quello 64 non sono buddy perché l'indirizzo iniziale del blocco ottenuto fondendo tale coppia di blochi non è un multiplo di (32×4) K.

10.4 Estensioni del Buddy System

Un limite dell'algoritmo Buddy System appena illustrato è che l'unità minima di allocazione di memoria è un page frame, ossia 4 KB. Se le richieste di

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

memoria delle funzioni del NUCLEO sono limitate (poche decine o centinaia di byte), assegnare un intera page frame quando basterebbe una piccola frazione di essa costituisce uno spreco di memoria.

I questi casi, è possibile realizzare un sistema di cache software che include blocchi liberi da 32, 64, 128, 256, 512, 1024, 2048 byte e soddisfare le richieste delle funzioni del NUCLEO cercando il blocco libero avente dimensione maggiore o uguale a quello del numero di byte richiesti: se, ad esempio, la funzione richiede 100 byte otterrà un blocco da 128 byte.

In pratica si riapplica l'algoritmo Buddy System con dimensioni dei blocchi che sono potenze di 2. Quando non vi sono più blocchi liberi di una determinata dimensione, l'algoritmo invoca la funzione `get_mem()` per ottenere una nuova page frame libera e la suddivide in blocchi della dimensione richiesta. Viceversa se una page frame usata per contenere blocchi di una dimensione prefissata, ad esempio 128 B, si libera, essa viene mantenuta nella cache dall'algoritmo e potrà quindi essere riutizzata quando vi sarà richiesta per nuovi blocchi senza dovere invocare il Buddy System.

10.5 Demand Paging

Il NUCLEO è un componente privilegiato del sistema operativo, per cui le funzioni del NUCLEO ottengono direttamente tutta la memoria dinamica richiesta.

Del tutto diverso è l'assegnazione di memoria ai processi. In questo caso vale la strategia opposta: l'assegnazione di RAM ad un processo viene ritardata il più a lungo possibile, ossia fino a quando il processo esegue una istruzione che necessita la presenza di una area di memoria non ancora allocata. Tale strategia prende il nome di *demand paging* (paginazione a richiesta) e presuppone l'esistenza nel NUCLEO di appositi descrittori che identificano le regioni di memoria possedute dal processo (vedi capitolo 8). Vediamo in che modo il NUCLEO è in grado di realizzare il demand paging.

Quando un processo inizia ad eseguire istruzioni, esso non ha ancora ottenuto alcuna page frame di RAM da parte del NUCLEO. Ha invece ottenuto diverse regioni di memoria, per cui prima di iniziare l'esecuzione del processo, il NUCLEO provvede ad inizializzare opportunamente i descrittori di memoria del processo.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

Alcune regioni di memoria mappano (vedi sezione 8.5) porzioni del file eseguibile associato al processo: la regione codice mappa una porzione del file eseguibile, la regione dati inizializzati mappa un'altra porzione del file eseguibile e così via. L'indicazione della porzione di file mappata è registrata nel descrittore di regione.

Altre regioni di memoria, ad esempio la regione stack non mappano alcuna regione di file ma costituiscono un mapping anonimo, per cui quando il processo tenterà di accedere ad un indirizzo incluso in una di tali regioni sarà sufficiente assegnare ad esso una page frame di RAM dinamica senza doverla inizializzare con dati o codice letti dal file eseguibile.

Continuiamo con la descrizione di cosa avviene quando un processo ottiene per la prima volta l'uso della CPU. Nell'eseguire la prima istruzione avente un determinato indirizzo logico, il circuito di paginazione indirizzerà la voce opportuna della Page Directory e la troverà posta a 0. In conseguenza, genererà una eccezione di tipo "errore di pagina" ed un apposito programma del NUCLEO chiamato "gestore degli errori di pagina" prenderà il controllo (vedi sezione 9.3).

A questo punto il gestore prende in considerazione tre casi possibili:

- l'indirizzo logico che ha causato l'eccezione non appartiene allo spazio degli indirizzi del processo: in questo caso, il NUCLEO elimina il processo che ha causato l'eccezione;
- l'indirizzo logico che ha causato l'eccezione appartiene allo spazio degli indirizzi del processo ma il processo non è autorizzato ad accedere alla page frame nel modo richiesto (tentativo di scrittura su pagine di sola lettura, ecc.): anche in questo caso, il NUCLEO elimina il processo che ha causato l'eccezione;
- l'indirizzo logico che ha causato l'eccezione appartiene allo spazio degli indirizzi del processo ed il processo è autorizzato ad accedere alla page frame nel modo richiesto: in questo caso va attivato il demand paging che viene realizzato in modo diverso a seconda che il mapping della regione coinvolta sia di tipo anonimo oppure sia associato ad una porzione di file.
  - mapping anonimo: il gestore invoca `get_mem()` per ottenere una page frame libera ed inserisce l'indirizzo fisico di tale page frame

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

nei 20 bit più significativi della voce della Page Directory o Page Table. I flag della voce sono impostati in base ai valori associati al descrittore della regione di memoria; a questo punto l'eccezione è stata trattata e l'esecuzione del processo può ripendere;

– mapping di una porzione di file: oltre ad eseguire le azioni svolte per il mapping anonimo, il gestore provvede ad iniziare iuna lettura di 4 KB della porzione di file mappata e pone il processo nello stato di "attesa della fine lettura dati da disco"; in questo caso, l'esecuzione del processo potrà riprendere solo dopo la fine della lettura dei dati richiesti da disco.

10.6 Swapping

Un problema perenne dell'informatica è quello della limitata quantità di RAM disponibile. In effetti, esiste una sorta di effetto autostrada per cui, non appena si introducono elaboratori con una maggiore quantità di RAM, i progettisti software sviluppano applicazioni che saturano tale tipo di risorsa.

L'introduzione di sistemi multitasking rende il problema ancora più critico poiché la RAM dell'elaboratore deve essere condivisa tra più programmi indipendenti.

Non esiste, viceversa, limitazione per quanto riguarda lo spazio degli indirizzi dei vari processi, ossia l'insieme di indirizzi logici che il processo può utilizzare. In effetti, tutti i moderni sistemi operativi sfruttano appieno gli indirizzi da 32 bit garantendo ad ogni processo uno spazio degli indirizzi di 4 GB, ossia 4096 MB. Nel caso di architetture più recenti che fanno uso di registri da 64 bit, la dimensione dello spazio degli indirizzi potrebbe raggiungere il valore astronomico di 2 elevato alla 64, ossia oltre 16 miliardi di miliardi.

Per alleviare il problema causato dalla insufficiente quantità di RAM disponibile, è stata introdotta una interessante tecnica di gestione della memoria, molto utilizzata negli attuali sistemi multiprogrammati, che prende il nome di *swapping*.

Tale tecnica può essere considerata come una estensione della paginazione descritta in precedenza e coinvolge la gestione simultanea di una area di disco e di una area di memoria. Sia il disco che la memoria sono gestiti tramite blocchi di lunghezza fissa, tipicamente 4 KB. Ogni blocco contiene quindi una pagina di dati.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

Al solito, un esempio aiuta a chiarire il ruolo dell'area di swap. Supponiamo di avere una RAM da 64 MB e di avere definito un'area di swap da 128 MB. In questo caso, il sistema di swapping dà l'illusione al programmatore di disporre di una RAM di $64 + 128 = 192$ MB. Ovviamente, si tratta di una illusione, nel senso che la RAM aggiuntiva viene simulata tramite un'area di disco, per cui il tempo di accesso a pagine contenute nell'area di swap è molto elevato.

Per quanto riguarda le tabelle di paginazione, una pagina appartenente al processo ma swappata su disco viene identificata impostando i 32 bit della voce di Page Table nel seguente modo:

- il flag di Present viene posto a 0;
- i rimanenti bit sono usati per specificare l'indirizzo nell'area di swap dove è registrata la pagina.

A questo punto, è necessario parlare dei criteri con cui una pagina viene trasferita dalla RAM nell'area di swap (attività di *swap out*) o, viceversa, viene trasferita dall'area di swap nella RAM (attività di *swap in*).

Di norma, il NUCLEO evita di ricorrere all'area di swap fino a quando ciò risulta possibile. Se però un processo richiede una nuova pagina e non vi è più RAM dinamica disponibile, il NUCLEO effettua uno swap out su una delle pagine presenti in RAM, liberando così una page frame ed usa tale page frame per soddisfare la richiesta del processo.

Come detto in precedenza, il NUCLEO modifica la voce della Page Table della pagina swappata su disco impostando il flag Present a 0. Quando il processo tenta di accedere ad una pagina swappata su disco, il gestore delle eccezioni di page fault riconosce, dal fatto che la voce della Page Table non è nulla ma che il flag Present è uguale a 0, che la pagina richiesta è stata swappata su disco. In questo caso, ottiene una page frame libera ed avvia una procedura di swap in per leggere la pagina da disco.

Quando la pagina contenente l'indirizzo logico richiesto non è presente in RAM e non esiste una page frame libera, il NUCLEO fa uso di un apposito *algoritmo di sostituzione* per selezionare una pagina tra quelle presenti in memoria da scrivere su disco e legge nella page frame appena liberatasi la pagina richiesta. Durante il doppio trasferimento di pagine, il processo è

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

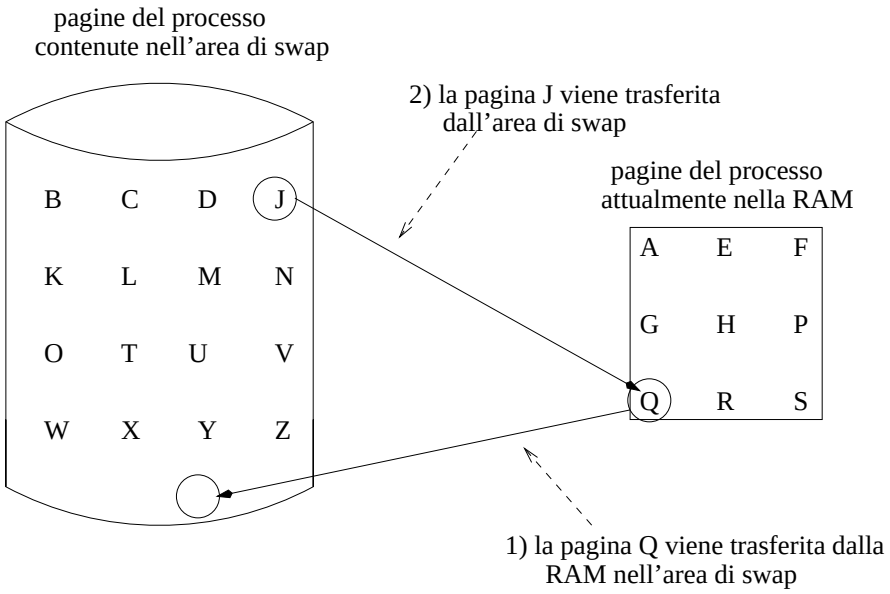

Figura 10.2: Swapping.

posto nello stato di bloccato. La Figura 10.2 illustra tale tecnica su un esempio semplificato: il processo in esecuzione richiede di indirizzare la pagina J attualmente non presente in RAM. Poiché non esistono page frame libere, l'algoritmo di sostituzione seleziona la pagina Q e la trasferisce su disco; dopo di che può essere caricata nella page frame resasi libera la pagina richiesta J ed il processo riprende l'esecuzione.

Gli algoritmi di sostituzione utilizzati sono basati su una proprietà statistica dei programmi chiamata *località*: è stato osservato analizzando le sequenze di indirizzi logici emessi da vari processi durante un qualsiasi intervallo di tempo che tali indirizzi non sono uniformemente distribuiti nello spazio degli indirizzi ma che tendono invece a concentrarsi in poche pagine. La spiegazione intuitiva di tale fenomeno è che la maggior parte dei programmi sono strutturati in sottoprogrammi e vettori o matrici di dati per cui, quando un processo esegue un sottoprogramma, esso concentra gli indirizzamenti nelle pagine contenenti il sottoprogramma e le strutture di dati associate.

Per questo motivo, è conveniente (dal punto di vista statistico) scegliere come pagina da trasferire su disco quella che non è stata indirizzata da più tempo,

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

mentre si può supporre che quelle indirizzate più recentemente saranno invece nuovamente richieste dal processo.

Sfruttando tale proprietà e introducendo hardware aggiuntivo per eseguire in modo efficiente l'algoritmo di sostituzione, è possibile fare uso di aree di swapping ed ottenere tempi medi d'indirizzamento di poco superiore al tempo d'indirizzamento della memoria fisica.

In conclusione, lo swapping, realizzabile tramite estensioni hardware e software del sistema, consente di disporre di uno spazio degli indirizzi maggiore dello spazio di memoria fisica assegnato al processo. I vantaggi per il programmatore sono che egli non deve più curare tramite apposite frasi di ingresso/uscita i trasferimenti di programmi tra memoria e disco ma può scrivere il programma come se disponesse di tutta la memoria necessaria. L'efficienza di un programma eseguito con l'area di swapping attivata dipende sia dalla sua località, sia dal rapporto tra il numero di pagine e il numero di page frame ottenute.

10.7 Gestione dello spazio su disco

Anche lo spazio su disco è una risorsa che deve essere gestita opportunamente. Il file system è il principale interessato nell'ottenere blocchi di disco liberi per potere ampliare la dimensione di un file. In mdo analogo, il file system rilascia blocchi di disco quando cancella un file o quando ne riduce la dimensione.

A differenza della RAM, la gestione dello spazio su disco non richiede appositi circuiti hardware quali l'unità di paginazione ma può essere interamente realizzata in software da appositi programmi del NUCLEO. In effetti, i tempi di indirizzamento del disco sono molto elevati per cui il peso relativo dei programmi per la gestione dello spazio su disco risulta trascurabile.

La gestione dello spazio su disco è realizzzata in modo alquanto semplice considerando il disco come un vettore di *blocchi fisici* aventi numeri d'ordine $0, 1, 2, \ldots$. Ogni blocco ha una dimesione fissa, tipicamente 1024 o 2048 B.

Il file system usa un'altra grandezza per identificare i blocchi che compongono un file: ogni file è considerato come un vettore di *blocchi di file* aventi numeri d'ordine $0, 1, 2, \ldots$. Ogni blocco di file ha la stessa dimensione di un blocco fisico. Espandere un file significa ottenere ulteriori blocchi di file; troncare un file significa eliminare blocchi di file a partire dall'ultimo.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

Si noti l'analogia tra indirizzo fisico di un page frame e blocco fisico da un lato, e tra indirizzo logico di una pagina di dati e blocco di file dall'altro. Come per la gestione della RAM, ogni descrittore di file include una apposita struttura che associa ai blocchi di file i corrispondenti blocchi fisici. In questo modo, il file system può effettuare indirizzamenti all'interno del file facendo riferimento a blocchi di file e può quindi ignorare quali specifici blocchi fisici sono stati assegnati al file.

In conclusione, la gestione dello spazio consiste quindi in una semplice paginazione realizzata in software che consente di:

- sfruttare al meglio lo spazio su disco (un semplice vettore di bit identifica i blocchi fisici liberi e quelli occupati);
- consentire di espandere o contrarre la dimensione dei file in modo arbitrario senza dovere fare uso di blocchi fisici contigui.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

Capitolo 11

GESTIONE DEI DISPOSITIVI DI I/O

11.1 Introduzione

L'esistenza di numerosi dispositivi di I/O, ognuno con caratteristiche particolari, complica notevolmente la struttura del NUCLEO che deve includere appositi programmi di gestione per ognuno dei dispositivi "sopportati". Nei NUCLEI attuali, oltre il 50% del codice è dedicato alla gestione dei dispositivi di I/O.

Data la complessità dell'argomento, la trattazione svolta in questo capitolo risulterà necessariamente sommaria. Per concretezza, faremo alcuni riferimenti al modo in cui il sistema operativo Unix gestisce le operazioni di I/O.

11.2 Architettura di I/O

Prima di iniziare a descrivere il modo in cui il NUCLEO gestisce i dispositivi di I/O, è opportuno dare alcuni cenni sulla *architettura di I/O*, ossia sul modo in cui i dipositivi di I/O sono collegati agli altri componenti dell'elaboratore.

Come indicato nella Figura 11.1, il dispositivo di I/O (stampante, disco, tastiera, mouse, ecc.) è collegato ad un apposito controlllore hardware. Tale controllore è collegato, a sua volta, ad una interfaccia la quale include alcune porte di I/O collegate al bus.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

Dal punto di vista del programmatore, l'accesso ai dispositivi di I/O si realizza tramite le porte di I/O. Ogni porta di I/O possiede un apposito indirizzo fisico. Esistono inoltre apposite istruzioni (istruzioni `in` e `out` nell'architettura Intel 80x86) che consentono di trasferire dati da una porta di I/O ad un registro della CPU o viceversa. Tali istruzioni includono appositi indirizzi di I/O da 16 bit che si possono considerare come logici e fisici allo stesso tempo: in effetti, la CPU non fa uso del circuito di paginazione nell'eseguire le suddette istruzioi.

Le porte di I/O consentono di realizzare le seguenti funzioni:

- inviare un comando al dispositivo;
- leggere lo stato del dispositivo;
- leggere dati provenienti dal dispositivo
- inviare dati al dispositivo.

La maggior parte dei dispositivi di I/O prevede la possibilità di emettere appositi segnali di interruzione per segnalare la fine di una operazione di I/O. Esiste inoltre un apposito flag, programmabile tramite porta di I/O, che abilita o disabilita le interruzioni di I/O emesse dal dispositivo.

Dispositivi con elevato tasso di trasferimento (ad esempio, i dischi magnetici) possono essere collegati ad un processore autonomo di I/O chiamato Direct Memory Access Controller (DMAC). Anche in questo caso, è possibile programmare tramite appositi flag di una porta di I/O l'attivazione del DMA per il dispositivo. Grazie al DMAC, il NUCLEO è in grado di avviare una operazione di I/O relativamente complessa, ad esempio la lettura di più blocchi di dati da disco in un'area prefissata di RAM, per poi passare a svolgere altre attività senza aspettare la terminazione dell'operazione di I/O. Quando il DMAC ha terminato di trasferire dati, invia un segnale alla interfaccia di I/O interessata, la quale genera, a sua volta, un segnale di interruzione.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

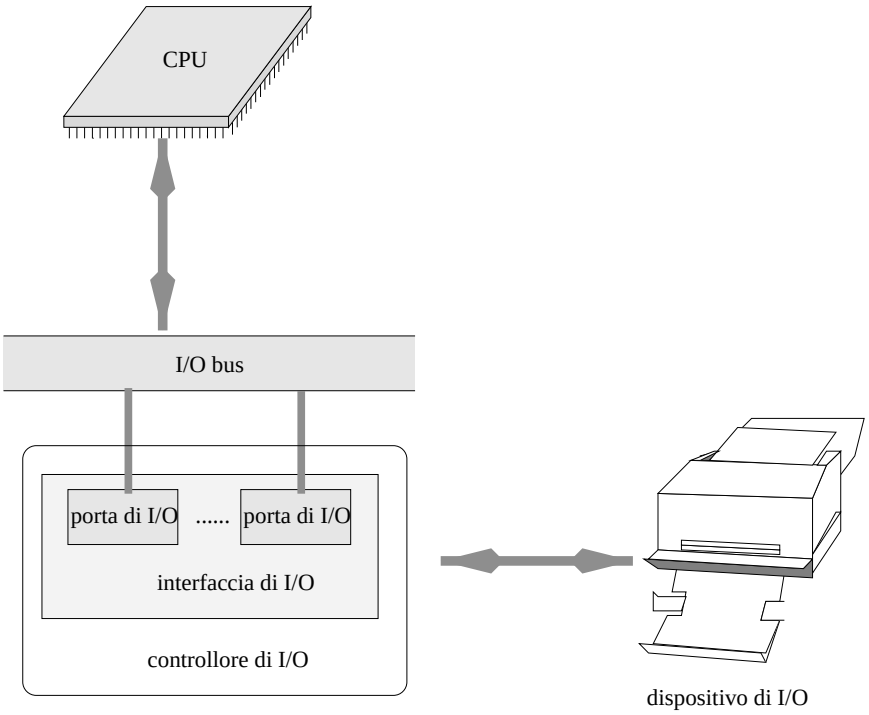

Figura 11.1: Architettura di I/O.

11.3 Dispositivi di I/O riconosciuti dal file system

Ogni sistema operativo tenta di "virtualizzare" nel modo più efficace possibile i vari dispositivi di I/O sopportati offrendo al programmatore una interfaccia omogenea.

Per concretezza, diamo alcuni cenni sul come ciò viene realizzato nel sistema operativo Unix.

In primo luogo, Unix riconosce due tipi di dispositivi di I/O:

- dispositivi a caratteri: sono dispositivi lenti (tastiera, stampante, ecc.) che trasferiscono dati un carattere alla volta e non richiedono buffer; solitamente, l'accesso ai dispositivi a caratteri è di tipo sequenziale;
- dispositivi a blocchi: sono dispositivi più complessi (disco rigido, lettore

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

CD-ROM, ecc.) che trasferiscono dati un blocco alla volta e richiedono pertanto un buffer per contenere tale blocco; l'accesso ai dispositivi a blocchi è di tipo casuale.

Ogni dispositivo è identificato da Unix tramite due identificatori chiamati *major number* e *minor number*. Ad esempio, i vari dischi gestiti dallo stesso controllore di disco hanno lo stesso major number ma minor number diversi.

Ogni dispositivi è gestito dal file system di Unix tramite un apposito device file di tipo orientato a blocchi oppure orientato a caratteri (vedi Paragrafo 4.4.3).

11.3.1 Programmazione di un device file

In Unix, la programmazione ad alto livello di un device file non differisce da quella di un file standard: le stesse "file operation" `open()`, `close()`, `read()`, `write()`, `lseek()`, `ioctl()`, ecc. usate sui file standard possono essere usate su device file.

Il programma illustrato appresso (si tratta essenzialmente di una versione semplificata del comando Unix `cp`) mette in evidenza la versatilità del file system Unix: il programma di copiatura tra file è in grado di operare su sia su device file che su file standard e, in questo secondo caso, prescinde dal file system in cui sono registrati i file standard (Ext2, VFAT, ecc.):

```
inf = open(p, O_RDONLY, 0);
outf = open(q, O_WRONLY|O_CREAT|O_TRUNC, 0600);
do  {
    n = read(inf, buf, 4096);
    write(outf, buf, n);
    } while (n);
close(outf);
close(inf);
```

Ovviamente, vi sono limitazioni legate al tipo di dispositivo di I/O, per cui ogni dispositivo ammette un sottoinsieme di tutte le file operation esistenti.

Se il dispositivo è un dispositivo di solo input, ad esempio la tastiera, la file operation `write()` non è applicabile a tale dispositivo. In modo analogo, se il dispositivo è di tipo orientato a caratteri, la file operation `lseek()` non è applicabile a tale dispositivo.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

Solitamente, tutti i device file sono inclusi nella directory `/dev`. Per potere includere un nuovo dispositivo di I/O in tale directory, è necessario averlo prima "registrato". Con tale termine si intende l'attività mediante la quale il sistema operativo acquisisce informazioni circa un nuovo tipo di dispositivo. I principali dati passati nella fase di registrazione riguardono:

- major e minor number del dispositivo;
- funzioni specializzate che realizzano il gruppo di file operation applicabili al dispositivo (se una funzione non è applicabile, viene passato un puntatore nullo).

11.4 Supporto del NUCLEO

Vediamo ora in che modo il NUCLEO supporta la gestione dei dispositivi di I/O. A seconda del tipo di dispositivo, il NUCLEO può offrire il seguente livello di supporto:

- nessun supporto;
- supporto limitato alla porta di I/O;
- supporto totale.

L'esempio più ovvio in Unix del primo caso riguarda l'interfaccia grafica del monitor: il NUCLEO di Unix non offre alcun supporto per l'interfaccia grafica, per cui viene usato una apposita applicazione (solitamente, X Window) per gestire la grafica dello schermo nonché il mose per disegnare il cursore grafico. Tale applicazione programma le porte di I/O mediante apposite istruzioni di tipo `in` e `out` dopo avere ottenuto dal sistema operativo i necessari permessi per accedere alle porte di I/O (vedi chiamata di sistema `ioperm()`).

Un esempio relativo al secondo caso riguarda le porte seriali presenti nel calcolatore: il NUCLEO sopporta le porte seriali presenti (`/dev/ttyS0`, `/dev/ttyS1`, ecc.) consentendo al programmatore di usare le file operation `read()` e `write()` su tali porte considerate come dei device file..

Il NUCLEO non sopporta invece alcuno dei vari dispositivi collegabili alla porta seriale (modem, mouse, stampante, ecc.). La gestione di tali dispositivi deve essere effettuata da appositi programmi applicativi esterni al NUCLEO.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

Il terzo caso è il più impegnativo in quanto richiede la realizzazione di un apposito programma chiamato gestore del dispositivo di I/O (in inglese *I/O driver*) per ognuno dei dispositivi riconosciuti dal sistema operativo. Tutti i dispositivi collegati alla porta parallela, ad uno dei vari tipi di bus presenti nel sistema (EIDE, PCI, SCSI, USB), oppure ad una interfaccia PCMCIA richiedono un I/O driver.

I compiti dell'I/O driver sono essenzialmente quattro:

- avviare l'operazione di I/O offrendo una interfaccia semplificata che nasconda il più possibile le caratteristiche hardware interne del dispositivo;
- forzare la terminazione dell'operazione di I/O se essa non è terminata entro un tempo prefissato; ciò viene realizzato tramite un meccanismo di time-out che invia al NUCLEO un apposito segnale di interruzione quando l'intervallo di tempo è scaduto;
- gestire i segnali di interruzione emessi dal dispositivo risvegliando il processo bloccato in attesa della terminazione dell'operazione di I/O;
- analizzare l'esito dell'operazione di I/O inviando eventuali messaggi di errore nel caso in cui essa non sia andata a buon fine.

Poiché ogni dispositivo di I/O ha caratteristiche peculiari e condizioni di errore proprie (ad esempio, un gestore di stampante deve essere in grado di gestire segnali di fine carta nel contenitore della stampante!), i moderni NUCLEI includono centinaia di gestori distinti.

Allo stesso tempo, l'incessante innovazione tecnologica fa sì che appaiono ogni giorno sul mercato nuovi dispositivi di I/O che devono essere integrati nei NUCLEI esistenti (un esempio è costituito dai lettori DVD apparsi da poco tempo sul mercato dei personal computer).

Per quanto detto prima, un gestore di dispositivo di I/O non può essere realizzato come un processo di sistema poiché deve essere in grado di gestire un apposito segnale di interruzione; esso deve essere invece integrato nel NUCLEO. In conseguenza, il NUCLEO deve essere predisposto ad accogliere nuovi gestori di dispositivi.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

11.5 Sincronizzazione tra CPU e dispositivo di I/O

Non è possibile in questo breve corso dare ulteriori dettagli sulla struttura interna dei driver di I/O. Vale la pena tuttavia accennare al fatto che la sincronizzazione tra CPU e dispositivo di I/O può essere svolta in due modi diversi:

- Polling: la CPU interroga periodicamente la porta di I/O per verificare se l'operazione di I/O è terminata. Il ciclo di polling prevede solitamente il rilascio della CPU (una funzione del NUCLEO simile a quella usata per realizzare la chiamata di sistema `sched_yield()`) in modo da consentire ad altri processi di usare la CPU.

- Interruzioni: il driver di I/O avvia l'operazione e pone il processo nello stato di bloccato sull'evento "interruzione proveniente dal dispositivo". Il processo verrà posto successivamente nello stato di pronto non appena si sarà verificata l'interruzione richiesta. Quando ciò avviene, il driver di I/O riprende l'esecuzione ed analizza il codice di terminazione dell'operazione di I/O prima di ridare il controllo al processo interessato.

Il primo approccio è preferibile al secondo quando il tempo di risposta del dispositivo di I/O è breve rispetto al tempo richiesto per effettuare la commutazione di processi: le attuali stampanti dotate di buffer di notevli dimensioni sono solitamente gestite con la tecnica del polling.
Il secondo approccio è usato quando vi sono molti byte da trasferire: i driver di disco fanno uso del DMAC e di interruzioni per segnalare alla CPU la fine dell'operazione di I/O.

11.6 Uso di cache nei driver per dischi

La caratteristica più importante di un sistema operativo è l'efficienza ed, in particolare, l'efficienza del NUCLEO. Diverse categorie di utenti sottomettono vari sistemi operativi commerciali ad una serie di test impietosi basati sull'uso di programmi di prova (i cosiddetti programmi *benchmark*) e stabiliscono classifiche non sempre imparziali. Dato che le interfacce verso l'utente

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

risultano oggi abbastanza simili tra loro e che le tecniche di compilazione hanno raggiunto un livello difficilmente migliorabile, l'area in cui si può sperare di raggiungere migliori prestazioni è quella del NUCLEO.

In effetti, i moderni calcolatori spendono oltre il 50% del tempo nello stato Kernel, ossia eseguendo programmi del NUCLEO, per cui risulta cruciale tentare di migliorare le prestazioni di tale componente del sistema operativo.

Ciò premesso, è ragionevole affermare che il collo di bottiglia degli attuali calcolatori è costituito dalla lentezza dei dispositivi di I/O, ed in particolare dei dischi rispetto alle capacità di elaborazione della CPU. Mentre un disco è in grado di trasferire poche centinaia di byte in alcuni millisecondi, la CPU è in grado di eseguire decine di migliaia di istruzioni nello stesso intervallo di tempo!

La risposta dei progettisti di NUCLEI è stata di ricorrere ad un uso intensivo di memorie di tipo cache. Una *cache* è una zona di memoria RAM destinata a contenere una piccola parte dei dati contenuti nel disco. Tipicamente, per un disco avente una capacità di qualche Gigabyte si fa uso di una cache di qualche Megabyte, ossia di una zona di memoria avente una dimensione di due ordini di grandezza inferiore a quella del disco.

Quando viene richiesta una lettura o scrittura sul disco, il NUCLEO verifica se l'informazione richiesta non sia già disponibile nella cache. Nel caso affermativo, la preleva dalla cache evitando di effettuare un indirizzamento del disco, e quindi con un notevole risparmio di tempo. Se gli indirizzamenti ai settori del disco fossero distribuiti in modo uniforme, le cache non avrebbero ragione di esistere poiché la probabilità di trovare il dato richiesto nella cache sarebbe dell'ordine di uno su mille.

Fortunatamente, la maggior parte dei programmi utenti effettua indirizzamenti al disco in modo locale, per cui è molto probabile che il programma utente continui ad operare per qualche tempo sugli stessi blocchi di dati prima di richiederne altri. La località in termini di programmi significa che il processo non utilizza simultaneamente tutti i sottoprogrammi ma solo alcuni di essi. La località in termini di dati significa che il processo non utilizza simultaneamente tutte le strutture di dati ma solo su alcune di esse.

L'idea vincente che ha portato all'introduzione della cache è quella di conservare nella RAM blocchi di dati *anche quando nessun processo sta attualmente operando su tali dati*. Supponiamo che un processo abbia aperto un file ed

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis

abbia quindi effettuato scritture sul file. In questo caso, i blocchi di dati modificati verranno conservati nella cache anche dopo che il processo ha chiuso il file.

In effetti, il conservare nella cache i blocchi di dati indirizzati più recentemente, aumenta notevolmente la probabilità di non dovere effettuare indirizzamenti al disco, e quindi diminuisce il tempo medio di accesso al disco e migliorano le prestazioni dell'intero sistema.

L'utente può verificare con mano la presenza di una cache del disco rieseguendo due volte lo stesso comando e misurandone la durata. Eseguendo ad esempio:

```
time ls -R /usr/src/linux
```

due volte di seguito, si noterà come la seconda esecuzione del comando `ls` sia molto più rapida della prima per via dei dati ormai presenti nella cache.

Tra i vari compiti del NUCLEO si aggiunge quindi quello di gestire le cache dei dischi e di stabilire una strategia di svuotamento automatico della cache.

In effetti, ogni volta che un settore di disco non si trova nella cache, esso viene letto da disco ed inserito nella cache. Per evitare il riempimento totale della cache, il NUCLEO deve provvedere a riutilizzare alcune parti della cache; a tale scopo, rilascia blocchi di dati nel seguente ordine:

1. quelli letti da disco ma non modificati; tra quelli sceglie quelli non indirizzati da più tempo;

2. se non è stata rilasciata sufficiente memoria dalla cache, quelli letti e modificati; in questo caso, è necessario scrivere su disco la versione aggiornata prima di riutilizzare l'area di memoria.

È interessante osservare che l'uso di cache migliora a tale punto le prestazioni del disco che i dischi più moderni includono spesso una propria cache inclusa nell'unità di controllo del disco che viene gestita interamente in hardware.

Le notevoli prestazioni di tali dischi vanno lette tenendo presente che i tempi medi d'accesso dichiarati non si riferiscono solo alla parte meccanica ma tengono presente anche i benefici indotti dalla cache locale.

scarica appunti ingegneria GRATIS: www.riccardogalletti.com/appunti_gratis