Sistemi Operativi

Ionut Zbirciog

10 October 2023

1 Sistema Operativo

Un sistema operativo è un software essenziale che gestisce l'hardware di un computer e fornisce servizi fondamentali per l'esecuzione di programmi applicativi. Uno dei ruoli principali di un sistema operativo è mettere a disposizione l'hardware attraverso le chiamate di sistema. Esistono vari tipi di sistemi operativi, ognuno progettato per scopi specifici, tra cui:

- **Sistemi Operativi per Mainframe:** Progettati per gestire sistemi informatici di grande scala, come mainframe aziendali.
- **Sistemi Operativi per Server:** Ottimizzati per fornire servizi e risorse su reti e su Internet.
- **Sistemi Operativi per Personal Computer:** Utilizzati su computer desktop e laptop per l'uso quotidiano.
- **Sistemi Operativi per Smartphone e Computer Palmari:** Progettati per dispositivi mobili.
- **Sistemi Operativi per IoT e Sistemi Operativi Embedded:** Utilizzati in dispositivi embedded e nell'Internet delle Cose.

Ciò che tutti i sistemi operativi hanno in comune sono due aspetti chiave:

- **Extended Machine:** Forniscono un'estensione delle funzionalità e un'astrazione dell'hardware, consentendo ai programmi applicativi di interagire con l'hardware in modo uniforme.
- **Resource Machine:** Gestiscono la protezione e la condivisione equa delle risorse hardware tra i processi in esecuzione.

2 Processo

Un processo è un'entità fondamentale in un sistema operativo. Può essere definito come un programma in esecuzione. I processi rappresentano un'astrazione a livello utente che permette l'esecuzione di programmi. Ad ogni processo è associato uno spazio di indirizzamento, che contiene il codice del programma, i dati e altre informazioni necessarie per l'esecuzione.

Ogni processo ha un identificatore univoco chiamato **PID (Process ID)**, che viene utilizzato per identificarlo in modo univoco nel sistema.

Il layout di base di un processo include le seguenti sezioni di memoria:

- **Stack:** Contiene i puntatori ai dati e viene utilizzato per la gestione delle chiamate di funzioni e delle variabili locali.
- **Data:** Contiene le variabili del programma, incluse le variabili globali e i dati inizializzati.
- **Text:** Contiene il codice eseguibile del programma.
- **Gap:** Fornisce spazio per l'allocazione e la deallocazione dinamica della memoria durante l'esecuzione del processo.

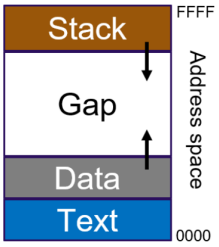

Figure 1: Layout di base di un processo

Il ciclo di vita di un processo comprende diverse fasi, e le informazioni sui processi attivi vengono mantenute in una tabella dei processi attivi. Un processo può essere creato (allocato nella lista dei processi), terminato (deallocato dalla lista dei processi), messo in pausa (spesso i processi sono in uno stato di sospensione) e ripreso. Inoltre, un processo può creare un altro processo, stabilendo una relazione padre-figlio, che definisce come i processi vengono generati.

Il possesso di un processo è assegnato a un utente identificato da un UID (User ID). Un processo figlio ha lo stesso UID del processo padre. Gli utenti possono appartenere a gruppi identificati da un GUID (Group ID). Inoltre, esiste un processo speciale noto come superuser o root, che ha privilegi elevati nel sistema.

3 File

Un file è un'astrazione di un dispositivo di memorizzazione, come un disco. I file possono essere letti e scritti specificando una posizione e una quantità di dati da trasferire. I file sono organizzati all'interno di directory, e le directory stesse possono contenere un elenco di identificatori per i file che contengono. Le directory e i file sono organizzati in una gerarchia, con la radice ("/") come directory principale. In un sistema UNIX, esistono due tipi di percorsi per identificare un file:

- **Percorso Assoluto:** Inizia dalla radice ("/") e specifica la posizione completa del file, ad esempio, "/home/username/file.txt".
- **Percorso Relativo:** È specifico alla directory corrente e si riferisce ai file relativamente alla directory in cui ci si trova, ad esempio, "../documents/report.pdf".

I dischi nei sistemi UNIX si trovano spesso in /mnt/disc o altre posizioni specifiche. In sistemi Windows, i dischi sono associati a lettere dell'alfabeto, come "C:" per il disco principale.

I diritti di accesso a un file sono controllati tramite tuple di tre bit per proprietario, gruppo e altri utenti. I bit di diritti di accesso includono "r" per la lettura, "w" per la scrittura e "x" per l'esecuzione. Ad esempio, "-rwxr-x–x" indica che il proprietario ha il permesso di eseguire, modificare e leggere il file, il gruppo può leggerlo ed eseguirlo, mentre altri utenti possono solo eseguirlo.

La prima lettera di una riga di diritti di accesso può essere "b" o "c" in caso di file speciali di blocco o carattere, ad esempio, per i dischi o le porte seriali.

I file e le pipe sono due tipi di astrazioni di file. Le pipe sono strutture dati che consentono ai processi di comunicare attraverso un canale FIFO (First In First Out).

Terminologia importante legata ai file include il percorso (path), la directory (folder), la directory di lavoro (working directory), il descrittore di file (file descriptor), i file speciali di blocco/carattere e le chiamate di sistema per la gestione dei file.

Termini importanti

- Path
- Folder/Directory
- Working Directory
- File Descriptor
- Block/Character special files
- Pipe

4 Chiamate di Sistema

Le chiamate di sistema rappresentano l'interfaccia che il sistema operativo offre alle applicazioni per richiedere servizi.

Poiché le chiamate di sistema sono specifiche del sistema operativo e dell'hardware, è essenziale che siano efficienti. Per rendere più agevole l'interazione con il sistema operativo, le chiamate di sistema sono spesso incapsulate in librerie C, come libc. Ogni chiamata di libreria corrisponde a una chiamata di sistema specifica. Ad esempio, la chiamata alla funzione `read(file, &buffer, nbyte)` consente di leggere dati da un file, specificando il puntatore al file, il buffer in cui scrivere i dati letti e la quantità di byte da leggere.

Quando un processo in modalità utente ha bisogno di un servizio di sistema, deve eseguire una **trap** o un'istruzione che effettua una chiamata di sistema. Questo passaggio consente al controllo di passare dalla modalità utente alla modalità kernel, dove il sistema operativo può gestire la richiesta. Dopo l'esecuzione di una chiamata di sistema, il controllo ritorna all'istruzione successiva a quella chiamata di sistema.

Le chiamate di sistema seguono una sequenza di passi:

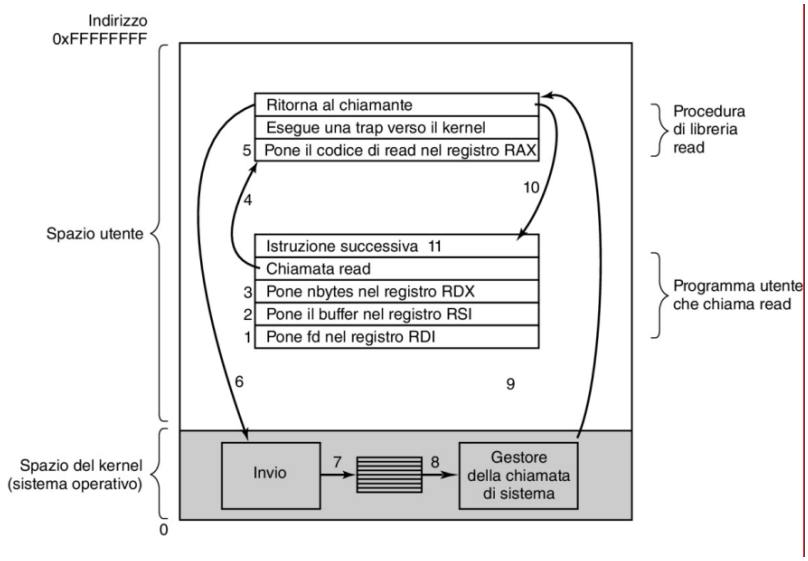

Figure 2: Chiamata di sistema

Esecuzione della chiamata di sistema read(fd, &buffer, nbytes).

1. Per prima cosa, il sistema operativo prepara i parametri mettendoli in registri appositi, secondo la convenzione di chimata System V. Da 1 a 3 il sistema mette i parametri in 3 registri.

2. Poi viene l'effettiva chiamata alla procedura di libreria (passaggio 4). Questa istruzione è la normale istruzione di chiamata di procedura usata per chiamare tutte le procedure.

3. La procedura di libreria, che può essere scritta in linguaggio assembly, colloca in genere il numero della chiamata di sistema in un resitro noto al sistema operativo, come il registro RAX (passaggio 5). Poi esegue un'istruzione trap (come l'istruzione x86-64 SYSCALL) per passare da modalità utente a modalità kernel.

4. Al passaggio 6, avviene l'esecuzione all'interno del kernel.

5. Il kernel esamina il numero di chiamata di sistema e la smista al gestore corretto.

6. Il gestore della chiamata di sistema esegue il lavoro richiesto.

7. Dopo il completamento del gestore, il controllo può ritornare alla procedura di libreria nello spazio utente. La chiamata di sistema può bloccare il programma chiamante se necessario, ad esempio, se il programma cerca di leggere da una tastiera senza input disponibile.

8. Questa procedura ritorna poi al programma utente nel modo consueto (passaggio 10)

9. Il programma passa alla successiva istruzione (passo 11).

Alcune chiamate di sistema comuni includono quelle per la gestione dei processi, dei file e del file system, nonché il controllo del tempo.

4.1 Chiamate di Sistema per la Gestione dei Processi

- `pid fork()`: Crea un processo figlio identico al processo genitore e restituisce il PID del processo figlio.
- `pid waitpid(pid, &statloc, options)`: Attende che un processo figlio specificato con il PID termini e restituisce lo stato di uscita del processo figlio.
- `s = execve(name, argv, environp)`: Sostituisce l'immagine centrale del processo con un nuovo programma specificato da `name`, passando gli argomenti in `argv` e l'ambiente in `environp`.
- `exit(status)`: Termina l'esecuzione del processo corrente e restituisce uno stato specificato da `status`.

4.2 Chiamate di Sistema per la Gestione dei File

- `fd = open(file, how, ...)`: Apre un file specificato da `file` in modalità lettura, scrittura o entrambe, restituendo un descrittore di file `fd`.
- `s = close(fd)`: Chiude un file aperto identificato dal descrittore di file `fd`.
- `n = write(fd, &buffer, nbytes)`: Scrive dati dal buffer `buffer` in un file identificato dal descrittore di file `fd`, scrivendo `nbytes` byte.
- `n = read(fd, &buffer, nbytes)`: Legge dati da un file identificato dal descrittore di file `fd` nel buffer `buffer`, leggendo un massimo di `nbytes` byte.
- `p = lseek(fd, offset, whence)`: Sposta il puntatore di lettura/scrittura in un file identificato dal descrittore di file `fd` in base all'offset specificato e alla modalità di spostamento `whence`.
- `s = stat(name, &buf)`: Ottiene informazioni sullo stato di un file specificato da `name` e le memorizza nella struttura `buf`.

4.3 Chiamate di Sistema per la Gestione del File System

- `s = mkdir(name, mode)`: Crea una nuova directory con il nome specificato da `name` e con i diritti di accesso specificati da `mode`.
- `s = rmdir(name)`: Rimuove una directory vuota con il nome specificato da `name`.
- `s = link(name1, name2)`: Crea un riferimento a un file specificato da `name1` con un nome alternativo specificato da `name2`.
- `s = unlink(name)`: Rimuove una voce dalla directory specificata da `name`, eliminando il file associato.

- `s = mount(special, name, flag)`: Monta un file system con una specifica opzione di montaggio identificata da `flag` sul punto di montaggio specificato da `name`.
- `s = unmount(special)`: Smonta un file system identificato da `special` dal punto di montaggio.
- `s = chdir(dirname)`: Cambia la directory di lavoro corrente del processo in quella specificata da `dirname`.
- `s = chmod(name, mode)`: Modifica i bit di protezione di un file specificato da `name` in base ai diritti di accesso specificati da `mode`.
- `s = kill(pid, signal)`: Invia un segnale specificato da `signal` a un processo identificato dal PID specificato da `pid` (nota: questa chiamata di sistema non termina il processo, ma invia un segnale ad esso).
- `s = time(&seconds)`: Restituisce il tempo trascorso in secondi dal 1 gennaio 1970 e lo memorizza nella variabile `seconds`.

5 Struttura di un Sistema Operativo

La struttura di un sistema operativo può variare a seconda dell'approccio adottato. Un'approccio comune è il modello monolitico, in cui l'intero sistema operativo è eseguito come un unico programma in modalità kernel. In questo modello, il sistema operativo è scritto come una raccolta di procedure collegate all'interno di un unico grande programma binario eseguibile. Questo approccio fornisce elaborazioni efficienti, ma può rendere il sistema operativo pesante e complesso. Inoltre, un errore in una parte del sistema può influenzare altre parti, causando crash sistematici.

Una struttura più modulare prevede la suddivisione del sistema operativo in tre strati principali: user mode, kernel mode e hardware. Il meccanismo di TRAP funge da interfaccia tra questi livelli. Questo modello permette di caricare dinamicamente componenti aggiuntive come driver di dispositivi o file system senza dover riavviare o ricompilare l'intero sistema operativo.

Inoltre, i sistemi moderni consentono di utilizzare librerie condivise e DLL (Dynamic Link Libraries) per condividere codice tra applicazioni e ridurre la duplicazione del codice in memoria.

6 Altri Comandi

Alcuni comandi utili nei sistemi operativi includono:

- `man (comando)`: Per accedere al manuale dei comandi.
- `pwd`: Per visualizzare la cartella corrente.
- `ls`: Per elencare il contenuto della cartella.
- `cd`: Per cambiare la directory corrente.
- `chmod`: Per cambiare i diritti di accesso ai file.