**Parte II - Capitolo 2 - Strutture Dei Sistemi Operativi**

Un sistema operativo offre un ambiente in cui eseguire i programmi e fornire i servizi. Ogni insieme di servizi offre funzionalità utili all'utente: interfaccia con l'utente (che può essere CLI, cioè da riga di comando, a lotti, cioè che le direttive siano codificate in un file eseguito in seguito, e la più diffusa GUI, cioè un sistema grafico di finestre dotate di puntatore), esecuzione di un programma, operazioni di I/O, gestione del file system, comunicazione, rilevamento degli errori. Altre funzioni che il sistema operativo deve svolgere, ma che non riguardano l'utente sono: assegnazione delle risorse, contabilizzazione dell'uso delle risorse, protezione e sicurezza.

Vi sono due modi fondamentali per gli utenti di comunicare con il sistema operativo. L'***interprete dei comandi*** è una funzionalità compresa nel kernel dei sistemi operativi. In alcuni ambienti come Windows e Unix l'interprete dei comandi è considerato un programma speciale; in altri, viene fornita una scelta tra molteplici interpreti dei comandi, definiti ***shell***. La funzione principale dell'interprete dei comandi consiste nel prelevare ed eseguire il successivo comando impartito dall'utente. A questo livello vengono usati molti comandi per la gestione dei file. I comandi possono essere implementati in due modi: o attraverso il codice di esecuzione del comando, o attraverso programmi speciali del sistema. Un'***interfaccia grafica con l'utente*** o ***GUI*** rappresenta la seconda modalità di comunicazione con il sistema operativo. Si tratta di uno strumento intuitivo (*user friendly*) che consiste in una scrivania di lavoro in cui, spostando il puntatore del mouse, si possono selezionare le icone, i file o le directory a cui si vogliono impartire comandi.

Le ***chiamate di sistema*** costituiscono l'interfaccia tra un processo e il sistema operativo. Tali chiamate sono generalmente disponibili sotto forma di linguaggio di programmazione C o assembly. Non bisogna essere a conoscenza di tutte le chiamate possibili che possono essere fatte al sistema, dato che, in genere, gli sviluppatori usano un'***interfaccia per la programmazione di applicazioni*** (API, application programming interface) che specifica un insieme di funzioni e dettaglia i parametri necessari per l'invocazioni delle stesse, insieme ai valori di ritorno. Il motivo principale per utilizzare una API e la portabilità: un programma sviluppato sulla base di una certa API girerà su qualunque sistema che la metta a disposizione. Il cosiddetto sistema di supporto all'esecuzione (*run time support system*) di un linguaggio di programmazione, fornisce nella gran parte dei casi un'***interfaccia alle chiamate di sistema*** rese disponibili dal sistema operativo, che funge da raccordo tra il linguaggio e il sistema stesso. Il chiamante non ha necessità di conoscere l'implementazione, ma è interessato solo a conoscere la API e i risultati dell'esecuzione della chiamata al sistema operativo. Vi sono tre metodi generali per passare parametri al sistema operativo. Il più semplice consiste nel passare i parametri in *registri*. Se i parametri sono maggiori dei registri utilizzati, i parametri vengono memorizzati in un *blocco* o tabella di memoria (metodo usato da Linux). Il programma può anche collocare i parametri in una *pila* da cui vengono prelevati dal sistema operativo.

Le chiamate di sistema sono classificabili in cinque categorie principali: controllo dei processi, gestione dei file, gestione dei dispositivi, gestione delle informazioni e comunicazione. Per quanto riguarda il ***controllo dei processi***, un programma deve potersi fermare in modo normale (end) o anormale (abort). Se si ricorre a una chiamata di sistema per terminare un programma in modo anormale, oppure se il programma incontra difficoltà e causa un'eccezione, talvolta si ha la registrazione in un file di un'immagine del contenuto della memoria (dump) e l'emissione di un messaggio d'errore. Un programma per la correzione degli errori detto *debugger* può analizzare queste informazioni per determinare le cause del problema. Un processo che esegue un programma può richiedere di caricare (load) ed eseguire (execute) un altro programma. È possibile inoltre determinare e reimpostare gli attributi di un processo (get process attributes, set process attributes). Per quanto riguarda la ***gestione dei file*** è innanzitutto necessario poter creare (create) o cancellare (delete) i file. Altre funzioni necessarie per gestire i file sono aprire (open) e usarlo, leggere (read), scrivere (write), riposizionare (reposition) e, alla fine, chiudere (close) il file. È possibile anche controllare e modificare gli attributi di un file (get file attributes, set file attributes) oltre ad altre chiamate di sistema come spostare (move) o copiare (copy). Per quanto riguarda la ***gestione dei dispositivi*** il sistema operativo deve occuparsi di controllare le diverse risorse che possono essere sia fisiche (es. nastri) o virtuali (es. file). In presenza di utenti multipli si può prescrivere una richiesta di dispositivo (request), che viene rilasciato (release) dopo l'uso. Una volta richiesto è possibile leggere (read), scrivere (write) ed eventualmente riposizionare (reposition). La ***gestione delle informazioni*** si occupa di trasferire le informazioni tra il programma utente e il sistema operativo. La maggior parte dei sistemi ha chiamate per ottenere l'ora (time) e la data (date) corrente, un'immagine della memoria (effettuare il dump). Un programma di tracciamento (trace) fornisce un elenco di chiamate di sistema in esecuzione. Le informazioni possono essere modificate (get process attributes, set process attributes). Per quanto riguarda la ***comunicazione*** tra i processi, esistono due modelli diffusi. Nel *modello a scambio di messaggi* i processi si scambiano messaggi per il trasferimento delle informazioni dopo aver aperto un collegamento tramite un nome di processo e tramite le chiamate get hostid e get processid. Il processo ricevente acconsente alla comunicazione tramite la chiamata di accept connection. I processi che gestiscono la comunicazione sono *demoni* specifici, cioè programmi di sistema realizzati per questo scopo. Questi programmi eseguono la chiamata wait for connection e sono chiamati in causa quando si stabilisce un collegamento. Nel *modello a memoria condivisa* i processi sono chiamati da shared memory create e shared memory attach per creare ed accedere alle aree di memoria possedute da altri processi.

Un'altra caratteristica importante di un sistema moderno è quella che riguarda la serie di ***programmi di sistema***, conosciuti anche come ***utilità di sistema***, che offrono un ambiente più conveniente per lo sviluppo e l'esecuzione dei programmi. In generale si possono classificare nelle seguenti categorie: gestione dei file (operazioni sui file e sulle directory), informazioni di stato (informazioni più o meno dettagliate sul sistema), modifica dei file (funzioni svolte da editor), ambienti d'ausilio alla programmazione (compilatori, assemblatori, programmi per la correzione di errori), caricamento ed esecuzione dei programmi, comunicazioni (meccanismi per il collegamento virtuale tra processi, utenti e calcolatori diversi).

Si passa ora ad affrontare i problemi di progettazione e realizzazione di un sistema. Il primo problema che si incontra riguarda la definizione degli scopi e delle caratteristiche del sistema stesso. I requisiti possono essere molto difficili da specificare anche se in generale si possono distinguere due gruppi principali: gli obiettivi dell'utente e gli obiettivi del sistema. Gli utenti desiderano un sistema che sia utile, facile da imparare e utilizzare, affidabile, sicuro e veloce. Requisiti analoghi sono richiesti da chi deve progettare il sistema: il sistema operativo deve essere facile da progettare, da realizzare e da manutenere, flessibile, affidabile, efficiente e senza errori. Ovviamente non esiste un'unica soluzione al problema data la vasta gamma di sistemi operativi eterogenei tra loro è facile capire che i requisiti sono diversi.

Un principio molto importante è quello che riguarda la distinzione tra meccanismi e criteri. I ***meccanismi*** determinano il modo in cui qualcosa deve essere eseguito. I ***criteri*** o ***politiche*** (***policy***) stabiliscono che cosa deve essere eseguito. Questa distinzione è molto importante ai fini della flessibilità, in quanto i criteri sono soggetti a cambiamenti di luogo e di tempo. I sistemi operativi basati su microkernel portano alle estreme conseguenza la separazione tra meccanismi e criteri, fornendo un insieme di funzioni fondamentali da impiegare come elementi di base, che consentono l'aggiunta di meccanismi e criteri più complessi.

La realizzazione dei sistemi operativi avveniva tramite linguaggio assembly; oggi si scrivono in linguaggi di alto livello come il C o il C++. I vantaggi di scrivere in linguaggio di alto livello sono gli stessi che si ottengono per i programmi applicativi: il codice si scrive più rapidamente, è più compatto, è più facile da capire e da mettere a punto. Inoltre risulta più facile da adattare ad un'altra architettura (*porting*). Gli eventuali svantaggi sono una minore velocità di esecuzione e una maggiore occupazione dello spazio di memoria.

Per funzionare correttamente il sistema operativo va progettato con estrema attenzione. Vediamo come è strutturato. I primi sistemi commerciali offrivano una ***struttura semplice*** (o ***struttura monolitica***) che non era ben definita, spesso nati come sistemi piccoli, semplici e limitati e solo in un secondo tempo accresciuti, superando il loro scopo originale. Un caso è l'MS-DOS in cui non vi è una netta separazione tra le interfacce e i livelli di funzionalità, che lo rende molto vulnerabile agli errori e agli attacchi dei programmi utenti. Questi limiti riflettono quelli dell'harware per cui era stato scritto (Intel 8088) che non distingueva tra modalità utente e di sistema e, quindi, non offriva protezione hardware. Anche il sistema UNIX è poco strutturato a causa delle limitazioni dell'hardware: esso consiste in due parti separate quali il kernel e i programmi di sistema. Il kernel è a sua volta diviso in una serie di interfacce utente e driver dei dispositivi espansi con l'evoluzione del sistema. In presenza di un harware appropriato, i sistemi operativi possono essere suddivisi in moduli più piccoli e gestibili. Uno dei metodi per rendere modulare un sistema operativo è detto ***metodo stratificato***, che rende il sistema suddiviso in un certo numero di livelli o strati: il più basso corrisponde all'hardware (strato 0), il più alto all'interfaccia con l'utente (strato n). Il vantaggio principale offerto da questo metodo è dato dalla semplicità di progettazione e dalle funzionalità di debug. Gli strati sono composti in modo tale che ciascuno di essi usi solo funzioni, operazioni o servizi che appartengono a strati di livello inferiore. La principale difficoltà risiede nella definizione appropriata dei diversi strati. È necessaria una progettazione accurata, poichè ogni strato deve poter sfruttare esclusivamente le funzionalità degli strati su cui poggia. Un ulteriore problema che si pone è che questa struttura tende ad essere meno efficiente (es. dispositivi di I/O). Man mano che il sistema operativo si è esteso, il kernel è cresciuto notevolmente, diventando difficile da gestire. Grazie agli studi della Carnegie Mellon University, fu realizzato il sistema operativo *Mach* che aveva il kernel strutturato in moduli secondo il cosiddetto orientamento a ***microkernel***. Seguendo quest'orientamento si progetta un sistema operativo rimuovendo dal kernel tutte le componenti non essenziali, che vengono realizzati come programmi utenti o di sistema. In generale un microkernel offre i servizi minimi di gestione dei processi, della memoria e di comunicazione. Uno dei vantaggi del microkernel è la facilità di estensione del sistema operativo: i nuovi servizi si aggiungono allo spazio utente e non comportano modifiche al kernel. Poichè è essenziale, se si devono eseguire modifiche al kernel, i cambiamenti sono ben circoscritti. Inoltre il sistema risulta più sicuro e affidabile (se un servizio è compromesso il resto del sistema operativo rimane intatto). I sistemi con microkernel possono incorrere in cali di prestazioni dovuti al sovraccarico indotto dall'esecuzione di processi utente con funzionalità di sistema. Molti sistemi operativi moderni, comunque, utilizzano i microkernel (Tru64 UNIX, Mac OS X). Il migliore approccio attualmente disponibile per la progettazione di sistemi operativi si fonda su tecniche di programmazione orientata agli oggetti per implementare un kernel modulare. Questa strategia a ***moduli*** è comune nelle implementazioni moderne di UNIX (Solaris), Linux e Mac OS X. La struttura di Solaris è formata da un kernel dotato di sette tipi di moduli caricabili: classi di scheduling, file system, chiamate di sistema caricabili, formati eseguibili, moduli STREAMS, varie, driver di dispositivi e del bus. Questa struttura permette di fornire servizi essenziali, ma anche di implementare dinamicamente alcune caratteristiche. Il sistema è molto flessibile, il modulo principale gestisce solo i servizi essenziali, oltre a poter caricare altri moduli e comunicare con loro. (Il Mac Os X di Apple sfrutta una struttura ibrida in cui il kernel è costituito dal microkernel Mach, che cura i processi essenziali, e dal kernel BSD, che si occupa dei servizi legati al file system e alla rete attraverso un'interfaccia a riga di comando).

Il ***debugging*** può essere genericamente definito come l'attività di individuare e risolvere errori nel sistema, i cosiddetti ***bugs***. Questa operazione viene effettuata sia sull'hardware che sul software e può comprendere una *regolazione delle prestazioni* che ha lo scopo di eliminare i colli di bottiglia che hanno luogo nei processi di sistema. Se un processo fallisce, la maggior parte dei sistemi operativi scrive le informazioni relative all'errore avvenuto in un ***file di log*** (***log file***) in modo da rendere noto agli operatori o agli utenti ciò che è avvenuto. Il sistema operativo può anche acquistare un'immagine del contenuto della memoria utilizzata dal processo, chiamata ***core dump***. Il ***debugger*** è lo strumento che permette di esplorare il codice e la memoria di un processo, incaricato di esaminare i programmi in esecuzione e i core dump. Un guasto nel kernel viene chiamato ***crash***. Come accade per i guasti dei processi, anche in questo caso l'informazione riguardante il guasto viene salvata in un file di log, mentre lo stato di memoria viene salvato in un'immagine del contenuto della memoria detto ***crash dump***. Per identificare eventuali colli di bottiglia dobbiamo monitorare le prestazioni del sistema. A tale scopo deve essere presente del codice che esegua misurazioni sul comportamento del sistema e mostri i risultati. Successivamente un programma di analisi può esaminare il file di log per determinare le prestazioni del sistema e identificare ostacoli e inefficienze. Un altro approccio di regolazione consiste nell'usare strumenti interattivi che permettono a utenti e amministratori di interrogare lo status dei vari componenti del sistema per individuare i colli di bottiglia.

DTrace

Il sistema operativo si deve configurare o generare per ciascuna situazione specifica. Questo processo è noto come ***generazione di sistemi*** (***SYSGEN***). I sistemi operativi sono normalmente distribuiti per mezzo di dischi o CD-ROM. Per generare un sistema è necessario usare un programma speciale che può leggere da un file o richiedere all'operatore le informazioni riguardanti la configurazione specifica del sistema o anche esplorare il sistema di calcolo per determinarne i componenti.

Dopo che il sistema operativo è stato scritto, bisogna predisporlo all'uso da parte dei dispositivi fisici. La procedure d'avviamento di un calcolatore attraverso il caricamento del kernel è nota come ***avviamento*** (***booting***) del sistema: nella maggior parte dei casi è presente un piccolo segmento di codice detto ***programma d'avvio*** o ***caricatore d'avvio*** (***bootstrap program*** o ***bootstrap loader***) che individua il kernel, lo carica in memoria e ne avvia l'esecuzione. L'esecuzione ha inizio in una locazione di memoria predefinita che contiene il registro delle istruzioni e il programma di avvio (è contenuto in una ROM, Read Only Memory). Il programma di avvio può eseguire diverse operazioni, tra cui la diagnosi della macchina per ottenere il suo stato. La scelta di custodire il sistema operativo nella ROM si addice ai piccoli sistemi, infatti, per i sistemi più grandi, subentra il problema della necessità di modificare il codice del programma di avvio. A questo proposito vengono sempre più spesso utilizzate i tipi di ***memoria a sola lettura programmabile*** (EPROM), che è appunto a sola lettura, ma può diventare rescrivibile qualora ricevesse un comando apposito. Tutte le forme di ROM sono anche dette ***firmware***, che sono un ibrido tra hardware e software. Per i sistemi operativi di grandi dimensioni il caricatore d'avvio è memorizzato nel firmware e il sistema operativo risiede su un disco. Il caricatore d'avvio trasferisce il blocco dove risiede il sistema operativo in memoria e lo esegue da quel ***blocco d'avvio*** (***boot block***). Un disco che contiene una partizione di avvio è detto ***disco d'avvio*** o ***disco di sistema*** (***boot disk***). Una volta caricato il programma di avvio completo, esso può caricare il kernel in memoria e dare inizio all'esecuzione, mandando in funzione il sistema (*running*).