**SISTEMI OPERATIVI**

**Che cos’è un sistema operativo?**

Il sistema operativo è il software più importante che possiamo trovare nel calcolatore la cui funzione principale e gestire le risorse della CPU, memoria principale e delle periferiche e inoltre ha il compito di controllore e di non permettere ai programmi in esecuzione di accedere a delle informazioni di altri programmi in esecuzione e ai dati del sistema operativo. Questo software gira in modalità Kernel.

**Le modalità Kernel e Utente🡪**  2 modalità di esecuzione della CPU.

**Kernel🡪** In modalità kernel gira il sistema operativo ed è una modalità in cui può eseguire qualsiasi istruzione si possa eseguire sul calcolatore. Inoltre **comprende la gestione dei processi, la gestione della memoria virtuale e del file system, i driver delle periferiche**. Queste sono le astrazioni introdotte dal sistema operativo.

**Utente🡪** In modalità Utente girano tutto il resto dei software e in questa modalità non si possono eseguire tutte le istruzioni come invece si poteva fare in modalità kernel per esempio la gestione della memoria o la gestione dei dispositivi di I/O non si può fare.

La gestione delle risorse di un sistema operativo include **il multiplexing**. Il multiplexing permette l’esecuzione protetta e simultanea di più programmi.

1)**Multiplexing space🡪** in questo caso la memoria principale viene suddivisa in più programmi in esecuzione, in modo che ognuno presente in memoria allo stesso tempo sia in attesa di usare al proprio turno la CPU.

2)**Time multiplexing🡪** quando una risorsa è condivisa temporalmente i programmi fanno a turno per usarla. Il compito del sistema operativo è vedere per quanto tempo usa la risorsa e chi è il prossimo ad usare la risorsa.

**MULTIPROGRAMMAZIONE**

L’obiettivo della multiprogrammazione è usare al massimo la CPU evitando tempi morti dovuti alla lentezza dell’I/O rispetto alla CPU. La soluzione pensata fu quella di partizionare la RAM in tanti pezzi assegnando un diverso JOB a ogni partizione. Mentre un JOB rimane fermo in attesa del completamento dell’I/O un altro usa la CPU. Potrebbe succedere che un JOB richieda alla sistema operativo di usare un dispositivo di I/O in questo caso allora la CPU viene assegnata ad un nuovo JOB. Questo è il vantaggio dei sistemi multi-programmati.

**CONTEXT SWITCH🡪** Il trasferimento della CPU da un programma all’altro viene chiamato cambio di contesto. Viene eseguito dal sistema operativo che deve salvare lo stato dei registri del programma che esce dalla CPU e caricare all’interno di quest’ultima il nuovo programma.

**SYSTEM CALL**

La system call è una chiamata di sistema e serve per ottenere servizi dal sistema operativo. Un programma utente fa una system call e quest’ultima entra nel kernel e fa una richiesta al sistema operativo

**SISTEMI INTERATTIVI (TIME SHARING)**

L’obiettivo di questi sistemi è di garantire tempi di reazione rapidi ai programmi interattivi, evitando che uno monopolizzi la CPU. La CPU viene assegnata a turno a vari programmi in esecuzione e questo turno ha una durata di un time-slice. Se ci sono più programmi che devono usare la CPU un dispositivo hardware chiamato timer si occupa di dire alla CPU attraverso un interrupt che un altro programma ha bisogno della CPU.

**TIME SLICE🡪** Il time slice è la durata di tempo di occupazione da parte di un programma della CPU è viene impostato un valore molto piccolo. Però time slice troppo brevi causano un overhead cioè un costo di gestione molto grande rispetto all’esecuzione del programma. Questo accade perché con time slice troppo brevi vengono effettuati molti context switch e quindi si passa una frazione di tempo troppo grossa a fare i cambi di contesto.

**GESTIONE MEMORIA**

La memoria ha al suo interno più processi/programmi ma come possono convivere tutti insieme? Dobbiamo fare attenzione a dove andiamo ad allocare i processi/programmi e vedere se in quella zona di memoria che abbiamo deciso di usare per allocare il programma non è già occupata**. L’idea è utilizzare gli indirizzi relativi (o virtuali) e poi fare traduzione in indirizzi fisici quando si sa dove è allocato il programma**. La traduzione viene effettua a tempo di esecuzione poiché si ha una massima flessibilità. La traduzione da indirizzo logico a indirizzo fisico viene controllata dalla **MMU**.

**PROTEZIONE MEMORIA🡪** Devo inoltre assicurarmi che il processo A e il processo B non vadano a scriversi nelle loro relative zone di memoria e per assicurarci che ciò non accada usiamo due registri che ci fornisce la nostra architettura.

**BASE🡪** indica dove si trova la prima istruzione del processo.

**LIMIT🡪** indica dove si trova la fine della zona di memoria del processo.

Questi due registri possono essere modificati solamente in modalità kernel. Il controllo del non superamento del registro limit viene effettuato dalla **MMU**.

**Interrupt🡪** Un interrupt è un segnale da parte di una periferica che indicano alla CPU il verificarsi di eventi esterni. In questo modo il processore deve salvare lo stato (context switch) dell’esecuzione del programma fino all’arrivo dell’interrupt e poi inizierà l’operazione di I/O. Al contrario di quanto succede con la richiesta di un operazione di I/O da parte di un processo se l’operazione di I/O è causato da un interrupt la CPU è libera da quel processo è viene avvertita che I/O è terminato tramite un interrupt. **Le operazioni per la comunicazione con i controller dei dispositivi di I/O sono privilegiate. Quindi totalmente invisibili al programma.** In questo modo solo il sistema operativo può comunicare con i dispositivi di I/O garantendogli protezione ai rispetto ai programmi utente.

Cosa succede alla CPU?

1) Appena arriva l’interrupt la CPU completa l’ultima istruzione che era in corso.

2)Come accade nel context switch salva lo stato d’esecuzione del processo interrotto.

3)Si salta nella zona di memoria dove è contenuto il gestore della periferica.

4)Dopo aver gestito l’interrupt viene ripristinato lo stato del programma interroto.

**Interrupt annidati🡪** Può capitare che subito dopo la richiesta di un primo interrupt ce ne subito dopo un’altra, come possiamo gestirli? La soluzione è impostare un registro chiamato **PSW** che indica che ci stiamo occupando già di un interrupt.

**SET ISTRUZIONI ISA PRIVILEGIATE**

Queste istruzioni sono eseguibili solo in modalità kernel.

**RETINT🡪** Riabilita gli interrupt e ripristina il programma interrotto.

**DISINT🡪** Disattiva gli interrupt.

**ENABINT🡪** Abilita gli interrupt.

Se queste istruzioni non sono eseguite in mode kernel si incorre in una **TRAP.**

**TRAP**

La Trap è un’interruzione provocata dall’esecuzione di una determinata istruzione di un programma in esecuzione. Il motivo dell’interruzione è causato per esempio da un’istruzione non eseguibile in mode utente.

**DIFFERENZA TRA INTERRUPT** **E TRAP**

La differenza tra Interrupt e Trap è che il primo è sincrono quindi è causato dal programma in esecuzione il secondo è asincrono rispetto al programma.

**PROCESSO**

Un processo è una attività di elaborazione guidata da un programma la cui velocità di esecuzione dipende da quanti processi condividono la CPU e la memoria.

**VELOCITA’ DI ESECUZIONE DEI PROCESSI**

In un sistema multi-programmato non si possono dare garanzie sulla velocità dei processi poiché non si può prevedere quali parti del codice verranno eseguite e non possiamo prevedere quando la CPU verrà interrotta da una richiesta di I/O.

**SISTEMI REAL-TIME🡪** Solo in questi sistemi sono garantite delle informazioni sulla velocità di esecuzione di un processo.

**CREAZIONE PROCESSI**

1)Inizializzazione del sistema

2)Esecuzione di una system call di creazione di un processo

3)Richiesta dell’utente di creare un processo

4)Inizio di un JOB in modalità BATCH.

All’avvio di un sistema operativo vengono creati parecchi processi. Alcuni sono processi attivi ossia interagiscono con gli utenti ed eseguono lavoro per loro. Altri processi lavorano in background e sono chiamati demoni.

**TERMINAZIONE PROCESSO**

1)Quando un programma ha finito correttamente la sua esecuzione.

2)Quando si verifica una condizione di errore rilevata dal programma ed esso decide di terminare.

3)Quando si verifica una condizione di errore che causa una TRAP.

4)Quando un processo A ne richiama un altro e quindi il processo A deve terminare.

**STATI DI UN PROCESSO**

Il sistema operativo deve tenere traccia dello stato di tutti i processi.

I processi possono essere in due stati: **ATTIVO** e **WAITING.**

Nello stato attivo il processo è in un punto del programma in cui non sta aspettando l’esecuzione di I/O al contrario nello stato di waiting il processo sta aspettando l’esecuzione di un I/O.

Nello stato attivo il processo si divide in due ulteriori fasi in quella di **RUNNING** e **READY.**

1)**Running🡪** Il processo in questo caso viene eseguito dalla CPU.

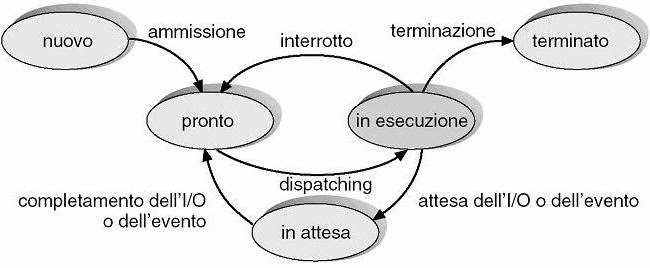
2)**Ready🡪** Il processo in questo caso è pronto ma non è eseguito dalla CPU perché quest’ultima è occupata.

Il passaggio dallo stato di ready allo stato di running avviene dopo un certo quanto di tempo.

**Chi sceglie il processo che deve passare dalla fase di ready alla fase running? Lo** **SCHEDULER.**

**Chi attiva il context switch? Il DISPATCHER.**

Entrambi sono componenti del sistema operativo



**La terminazione può avvenire da un qualunque stato però solo dallo stato running ci sarà una terminazione volontaria.**

**NUOVO🡪** è uno stato molto transitorio. Il processo sta qui il tempo che serve per allocare le risorse.

**TERMINATO🡪**in questo stato il processo è terminato ma il sistema operativo ne tiene ancora traccia perché possono servire informazioni sulla terminazione.

**IN CORSO🡪** il processo può essere eseguito, in esecuzione o bloccato in attesa di un evento.

**STATI DEL PROCESSO P DOPO L’EVENTO**

1)Il processo viene creato. (dallo stato new in cui sta per poco tempo va allo stato ready).

2)Il processo viene scelto dallo scheduler. (da stato ready a stato waiting).

3)Il processo chiama una read().un’operazione di I/O. (da stato ready a stato waiting).

4)Il processo passa nello stato di ready una volta che riceve l’interrupt.

5)Il processo riviene scelto dallo scheduler.

6)Il processo termina.

**PCB**

Il sistema operativo mantiene una scrittura chiamata PCB (Process control block). Essa è una tabella dei processi che contiene delle caratteristiche dello stato dei processi.

I contenuti tipici sono:

1)Registri CPU (per esempio PC, PSW, SP).

2)Dati dello scheduler (processo che ha la priorità, parametri dello scheduler).

3)Dati per accounting (tempo uso della CPU, tempo da quando il processo è iniziato).

4)Informazioni utili per protezione (ID utente, ID group).

Lo stato dei registri viene salvato nel PCB (Process control block) e ripristinato in occasione del context switch. In breve si può vedere come una tabella ma i PCB in diversi sistemi operativi sono delle strutture dati infatti possono essere nodi di liste che costituiscono la coda dei processi pronti.

**MODELLO UTILIZZO CPU🡪 CPU=1-pn** Ogni processo ha una certa attività continua di CPU (CPU burst) oppure un’attività continua di I/O (I/O burst).

p= proporzione di tempo di tutti i processi quando sono in I/O.