

Fondamenti di Informatica  
per la Sicurezza  
a.a. 2007/08

## Cenni di sistemi operativi

**Stefano Ferrari**



UNIVERSITÀ DEGLI STUDI DI MILANO  
DIPARTIMENTO DI TECNOLOGIE DELL'INFORMAZIONE

Stefano Ferrari ★ Università degli Studi di Milano

Fondamenti di Informatica per la Sicurezza ◇ Cenni di sistemi operativi ◇ a.a. 2007/08 - p. 1/39

### Motivazioni

Il sistema operativo (SO) è un insieme di programmi che:

- gestiscono l'hardware;
- forniscono un'interfaccia semplificata all'utente ed ai programmi utente verso l'hardware.

Il SO è utile perché:

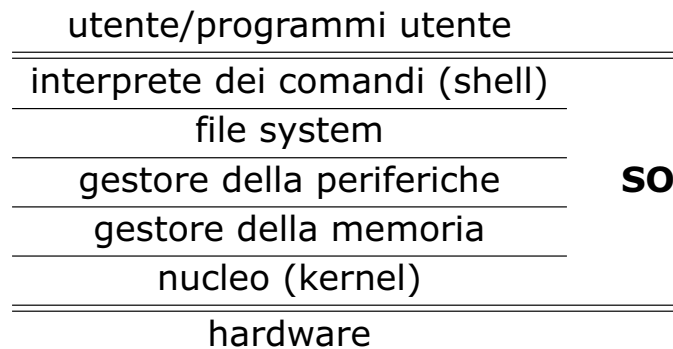
- l'uso diretto dell'HW non è agevole;
- i programmi dovrebbero essere riscritti se si cambiasse l'HW;
- la gestione della macchina è trasparente per l'utente.

Stefano Ferrari ★ Università degli Studi di Milano

Fondamenti di Informatica per la Sicurezza ◇ Cenni di sistemi operativi ◇ a.a. 2007/08 - p. 2/39

## Modello a buccia di cipolla

Il modello di SO più diffuso ha una struttura modulare a strati (**modello a buccia di cipolla**):



Ogni strato fornisce un'astrazione (**macchina virtuale**) agli strati più esterni.

## Kernel

Il **kernel**:

- è lo strato a contatto diretto con la CPU (se si cambia HW, solo il kernel va modificato);
- gestisce l'uso della CPU da parte dei programmi;
- permette ad ogni processo (programma+dati)/utente di considerarsi unico utilizzatore della CPU.

## Gestore della memoria

---

Il **gestore della memoria**:

- fornisce uno spazio virtuale di indirizzamento;
- protegge dati/istruzioni;
- maschera la collocazione fisica (e.g., swap);
- controlla la sovrapposizione degli spazi di memoria associati ai vari programmi.

## Gestore delle periferiche

---

Il **gestore delle periferiche**:

- fornisce una visione semplificata delle periferiche;
- fornisce ai programmi delle primitive ad alto livello per l'uso delle periferiche;
- gestisce i conflitti nell'uso delle periferiche (ogni programma "vede" un insieme di periferiche dedicate).

## File system

### Il file system:

- è uno strato dedicato ad una classe di periferiche peculiari;
- organizza i dispositivi di memoria di massa;
- gestisce la corrispondenza tra locazione fisica e identificatore del file;
- gestisce i permessi di uso dei file.

## Shell

### L'interprete dei comandi (shell):

- è il modulo accessibile all'utente (tramite le periferiche);
- interpreta i comandi dell'utente;
- si occupa del caricamento e dell'esecuzione dei programmi.

## Rete

La connessione di **rete** non è stata considerata: quando il modello è stato formulato, la rete veniva vista come una funzionalità aggiuntiva esterna al SO.

La rete può essere considerata semplicisticamente come una periferica particolare.

Tuttavia, l'appartenenza ad una rete di calcolatori è un aspetto che investe i vari strati del SO.

## Tipi di SO

I SO si possono classificare a seconda delle loro caratteristiche:

- monoutente monoprogrammato;
- monoutente multiprogrammato;
- multiutente multiprogrammato.

Fra i sistemi multiprogrammati sono diffusi:

- time-sharing (i processi usano alternativamente la CPU per un **quanto** di tempo);
- real-time.

Inoltre, ci sono sistemi dedicati.

## Parallelismo

L'architettura di von Neumann prevede un'esecuzione sequenziale dedicata.

Alcune operazioni, però, possono essere eseguite in parallelo.

Il parallelismo può essere individuato a diversi livelli:

- dati (esempio: cambiare la luminosità a tutti i pixel dell'immagine);
- istruzioni ( $a=b+c$  e  $z=1$ );
- programmi (lettore mp3 e word processing).

## Parallelismo: motivazioni

I principali motivi a supporto del parallelismo sono:

- **efficienza**: mentre un processo attende un input, può lasciare l'uso della CPU ad un altro;
- **interattività**: l'interazione con l'utente richiede brevi, ma frequenti intervalli di utilizzo della CPU;
- **sincronizzazione/cooperazione**: si può semplificare la descrizione di molti programmi utilizzando attività concorrenti, aumentando anche l'efficienza della CPU (e.g., modello produttore-consumatore).

## Processo

Un processo (entità dinamica) è un programma (entità statica) in esecuzione.

Il dinamismo è espresso da:

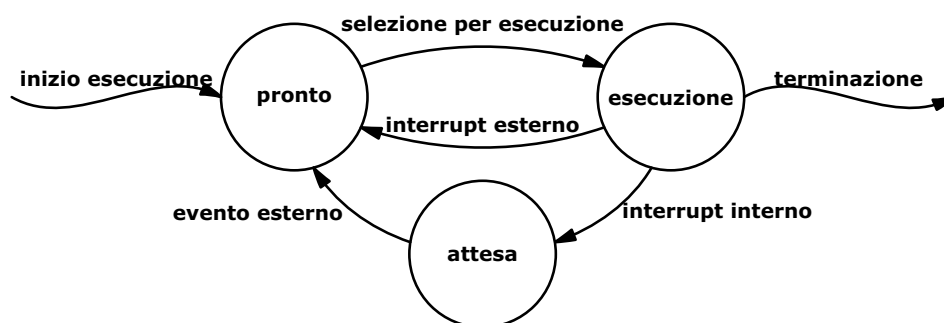
- evoluzione dei dati;
- flusso di esecuzione.

Ad ogni programma possono corrispondere più processi (figli): ognuno afferrisce a diverse sezioni del programma (processo padre).

## Stati di un processo (1)

Un processo si trova in uno dei seguenti tre stati:

- pronto — **ready** (coda dei processi pronti)
- esecuzione — **running** (uso della CPU)
- attesa — **waiting** (code delle periferiche)



## Stati di un processo (2)

Il processore viene utilizzato a turno dai processi in esecuzione, ciclando fra i tre stati:

- la transizione da **pronto** a **esecuzione** avviene quando il processo prende uso della CPU (**scheduler**);
- transita da **esecuzione** a **pronto** quando lo scheduler gli toglie l'uso della CPU (e.g., quando scade il suo quanto di tempo, evento esterno da gestire);
- passa da **esecuzione** a **attesa** quando il processo deve attendere un evento (e.g., carattere da tastiera);
- passa da **attesa** a **pronto** quando l'evento atteso si è verificato.

## Classificazione dei processi

I processi si dividono in due categorie:

- **I/O bound**, caratterizzati da frequenti operazioni di I/O (e.g., programma di scrittura);
  - lunghi periodi di attesa di eventi esterni intervallati da brevi elaborazioni;
- **CPU bound**, che fanno calcoli (e.g., simulazione);
  - lunghi periodi di calcolo intervallati da brevi operazioni di I/O.



## Multiprogrammazione

La multiprogrammazione è vantaggiosa perché:

- sfrutta i tempi morti;
  - mentre un processo attende un carattere da tastiera, la CPU può continuare ad eseguire un programma di puro calcolo;
- esegue lo switch tra processi già in memoria;
  - l'operazione di cambio di contesto (salvataggio dei valori dei registri del processo che lascia la CPU e ripristino di tali valori per il processo che subentra) è relativamente veloce perché avviene per processi che sono già in memoria centrale.

## Politiche di scheduling

L'obiettivo dello **scheduling** è la massimizzazione dell'utilizzo della CPU.

I più diffusi algoritmi di scheduling sono:

- **round robin**: ad ogni processo è assegnata la CPU per un quanto di tempo prefissato (sufficientemente grande rispetto al tempo di cambio del contesto);
- **priorità statica**: priorità fissata alla creazione dei processi in base alle loro caratteristiche (alta priorità per i processi interattivi, bassa priorità per i processi CPU bound);
- **priorità dinamica**: la priorità può essere modificata durante l'esecuzione dei processi.

## Vantaggi della concorrenza

L'esecuzione concorrente di processi ha diversi vantaggi, tra i quali:

- aumento del tempo di utilizzo effettivo della CPU;
- suddivisione dei processi su più CPU o su più calcolatori;
- condivisione (controllata) delle stesse risorse.

## Problemi della concorrenza

Le interazioni tra processi concorrenti possono causare due problemi:

- **starvation (morte per inedia):** processi di priorità più elevata possono ritardare indefinitamente l'esecuzione di un processo a bassa priorità;
- **deadlock (blocco infinito):** se due processi richiedono entrambi una risorsa (o un risultato) occupata dall'altro, nessuno dei due processi può terminare.

**Esempio:** un idraulico ed un elettricista devono rifare gli impianti di una casa, ma entrambi esigono che sia prima sistemato l'altro impianto.

## Processi concorrenti

La concorrenza tra processi assume la forma di:

- **competizione** per le risorse, quando i processi devono svolgere compiti differenti, con lo scopo di portare a termine il loro compito il prima possibile;
- **cooperazione**, quando i compiti che devono portare a termine sono tra loro correlati (per esempio, spooling di stampa); in questo caso si presentano problemi di:
  - **sincronizzazione** delle attività: sono necessari dispositivi per controllare l'ordinamento degli eventi (semafori);  
**Esempio:** non versare l'acqua se non c'è sotto il bicchiere;
  - **comunicazione**: serve per lo scambio di dati tra i processi (memoria condivisa, messaggi).

## Gestione della memoria centrale

Il **gestore della memoria** è il modulo di sistema operativo che fornisce ad ogni processo uno **spazio di indirizzamento virtuale**.

Per ottimizzare l'utilizzo della memoria, il gestore della memoria consente:

- il caricamento di un programma a partire da un qualsiasi indirizzo;
- il caricamento in memoria (fisica) solo di porzioni di programma o dati;
- la condivisione di istruzioni da parte di processi differenti (generati dallo stesso programma).

## Spazio di indirizzamento

La memoria disponibile per un programma può essere descritta in termini di:

- **spazio logico**: è lo spazio di indirizzamento virtuale in cui gli indirizzi delle celle partono da 0;
- **spazio fisico**: è lo spazio di indirizzamento del dispositivo di memoria.

## Rilocabilità del codice

La rilocazione della memoria:

- è realizzata dal gestore della memoria;
- consiste in un mappaggio degli indirizzi da logici a fisici;
- permette di eseguire un processo in qualsiasi zona della memoria fisica;
- definisce lo **spiazzamento** di un processo: indirizzo fisico della cella 0 dello spazio logico.

## Rilocazione

La rilocazione può essere:

- **statica**, quando lo spiazzamento viene calcolato al momento del linking;
- **dinamica**, quando:
  - i programmi vengono caricati in memoria con gli indirizzi calcolati da 0;
  - il contenuto di un particolare registro (**registro base**) viene aggiunto ad ogni istruzione.

## Rilocazione dinamica

La rilocazione dinamica:

- è necessaria in sistemi multiprogramma;
- richiede la disponibilità di apposito HW;
- può essere effettuata separatamente su dati e istruzioni mediante due registri base (diversi processi possono avere lo stesso registro base istruzioni).

## Memoria virtuale

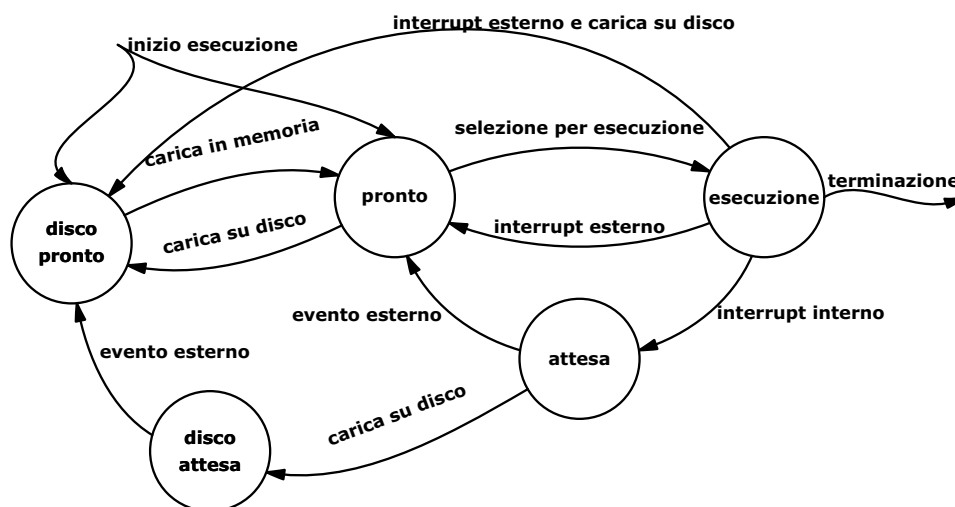
La memoria è una risorsa preziosa.

Generalmente, si dispone di meno memoria di quanta ne sarebbe necessaria ai processi attivi.

La **memoria virtuale** è una tecnica che consente di ampliare lo spazio di indirizzamento usando una parte della memoria di massa (**swap**) per coprire la mancanza di memoria centrale.

Se un processo richiede più memoria di quanta ne sia disponibile al momento, il gestore della memoria salva i dati di uno dei processi in attesa nell'area di swap, liberando così la memoria da essi occupata.

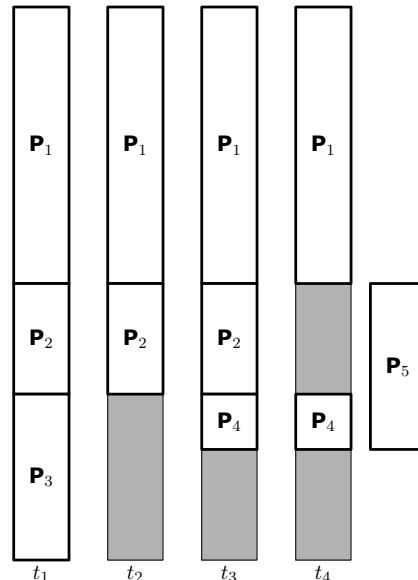
## Stati di un processo con swapping



## Frammentazione

Il gestore della memoria assegna la memoria disponibile ai programmi che devono essere eseguiti.

Si può verificare una situazione in cui vi è sufficiente memoria disponibile, ma non essendo **contigua** non può essere utilizzata.



## Paginazione

Si suddivide la memoria (sia fisica che logica) in blocchi di ugual dimensione (**pagine**).

Ad ogni processo si allocano solo un numero intero di pagine.

In questo modo:

- si guadagna la possibilità di allocare zone non contigue di memoria;
- si può attenuare il problema della frammentazione della memoria.

## Segmentazione

La **segmentazione** è una tecnica basata sulla suddivisione della memoria usata da un programma a secondo il tipo di informazione contenuta:

- segmento codice;
- segmento dati;
- segmento pila.

Ogni segmento viene trattato in modo indipendente dal gestore della memoria.

Permette la condivisione dello stesso segmento codice da parte di più processi.

## File system

Presenta all'utente e ai programmi una visione semplificata ed omogenea dei dispositivi per la memoria di massa:

- attraverso un'organizzazione logica (directory, volumi);
- mediante operazioni sui dati memorizzati, quali:
  - il recupero;
  - la cancellazione;
  - la modifica;
  - la copia.



## Dati e metadati

---

Oltre ai dati ed i programmi degli utenti, la memoria di massa deve memorizzare anche informazioni che riguardano il file system stesso:

- aree libere o allocate;
- directory.

## Allocazione

---

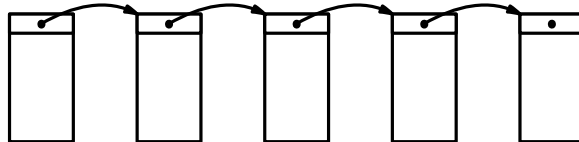
L'allocazione della memoria di massa condivide i problemi di allocazione della memoria centrale.

Per esempio:

- frammentazione;
- allocazione contigua (supporti write-once);
- allocazione a blocchi (supporti riscrivibili).

## Allocazione a lista concatenata

Un file è una concatenazione di blocchi.

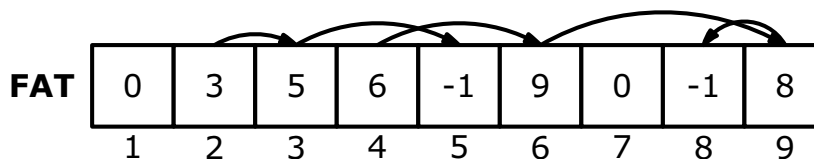


Ciò comporta che:

- una porzione di ogni blocco è impegnata dalla posizione fisica del blocco seguente;
- l'accesso al blocco  $N$ -esimo richiede l'accesso a  $N - 1$  blocchi;
- il danneggiamento di un blocco impedisce di recuperare il resto del file.

## File Allocation Table

La File Allocation Table (**FAT**) è una mappa di allocazione dei blocchi della memoria di massa.



Ogni elemento della mappa contiene uno fra i seguenti valori:

- l'identificatore di blocco vuoto;
- la posizione del prossimo blocco del file;
- l'identificatore di ultimo blocco.

## Svantaggi della FAT

L'approccio basato su FAT comporta i seguenti svantaggi:

- per essere efficiente, la FAT deve stare in memoria;
- l'accesso al blocco  $N$ -esimo richiede  $N$  accessi in memoria centrale (più uno in memoria di massa);
- la dimensione della FAT cresce più che linearmente con la dimensione del dispositivo.

**Esercizio:** Quanto occupa la FAT di un disco da 20 GiBi, con blocchi da 2 KiBi?

- Il disco dato ha  $10 \cdot 2^{20}$  blocchi: la FAT avrà  $10 \cdot 2^{20}$  elementi.
- Ciascun elemento richiede almeno 24 bit, quindi la FAT occuperà non meno di 30 MiBi.

## Allocazione indicizzata

Con l'allocazione indicizzata, ogni file è descritto da un blocco detto **blocco indice**.

Il blocco indice contiene l'elenco dei blocchi che compongono il file.

L'approccio può essere gerarchizzato (**indicizzazione multilivello**):

- nei file piccoli, il blocco indice punta ai blocchi dati;
- nei file grandi, il blocco indice punta a blocchi indice di livello inferiore.

Solo i blocchi indice dei file aperti devono stare in memoria.

## File system di rete

---

I sistemi distribuiti presentano problemi aggiuntivi nei seguenti campi:

- denominazione dei file;
- accesso remoto trasparente all'utente;
- consistenza dei dati;
- sicurezza.