Fondamenti di Informatica per la Sicurezza a.a. 2003/04

◇ *Cenni di sistemi operativi* ◇

Stefano Ferrari

Università degli Studi di Milano

Dipartimento di Tecnologie dell'Informazione

Sistema operativo: motivazioni

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

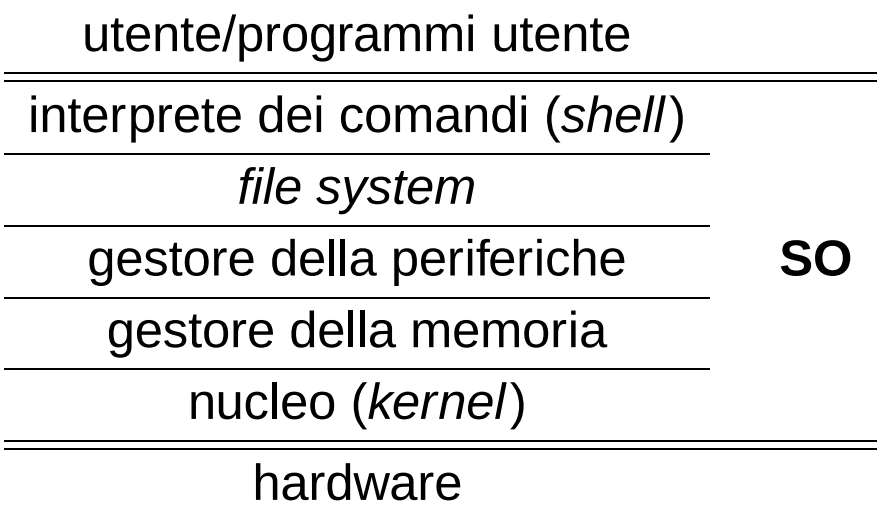
- gestiscono l'hardware
- forniscono un'interfaccia semplificata all'utente ed ai programmi utente verso l'hardware

Perché?

- l'uso diretto dell'HW non è agevole
- i programmi devono essere riscritti se si cambia l'HW
- trasparenza verso l'utente

Modello a buccia di cipolla

Il modello di SO più diffuso è basato su una struttura modulare conosciuta come *modello a buccia di cipolla*:



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

Modello a buccia di cipolla (2)

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
- ogni processo (programma+dati)/utente "vede" una CPU tutta per sè

gestore 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

Modello a buccia di cipolla (3)

gestore periferiche fornisce una visione semplificata delle periferiche

- i programmi usano le periferiche tramite primitive ad alto livello
- gestisce i conflitti nell'uso delle periferiche: ogni programma "vede" un insieme di periferiche dedicate

file system la memoria di massa è una periferica peculiare

- organizzazione della memoria di massa
- gestione della corrispondenza tra locazione fisica e identificatore del file
- permessi

Modello a buccia di cipolla (4)

shell è il modulo accessibile all'utente (tramite periferiche)

- interpretazione dei comandi
- caricamento ed esecuzione programmi

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

Tipi di SO

I SO si possono classificare a seconda delle loro caratteristiche:

- monoutente monoprogrammato
- monoutente multiprogrammato
- multiutente multiprogrammato
- time-sharing (i processi usano alternativamente la CPU per un *quanto di tempo*)
- real-time

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

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

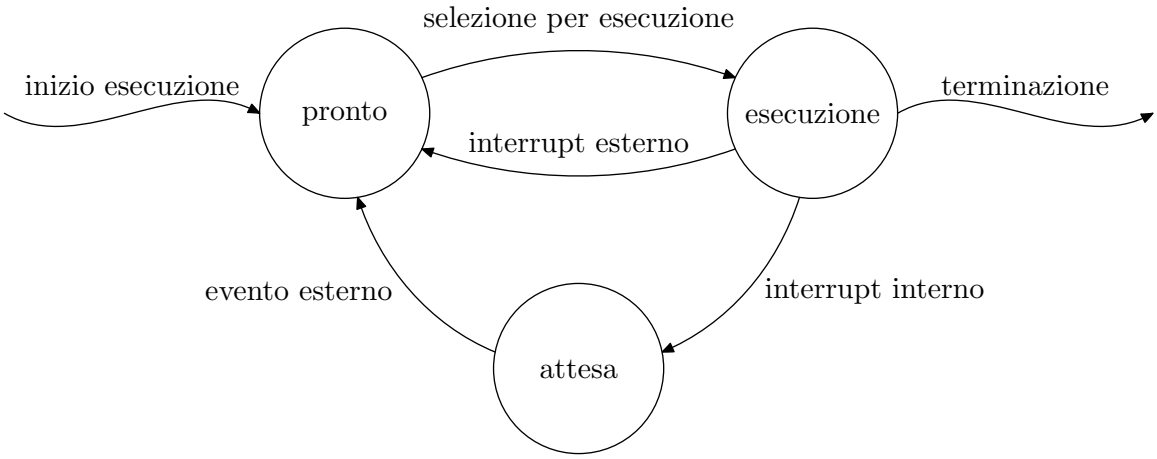
Un processo (entità dinamica) è un programma (entità statica) in esecuzione:

- evoluzione dei dati
- flusso di esecuzione

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

Stati di un processo

- 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 (*pronto*, *attesa*, *esecuzione*)
  - 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

Multiprogrammazione

- I processi si dividono in due categorie:

  I/O bound caratterizzati da frequenti operazioni di I/O (e.g., programma di scrittura)

  CPU bound il loro scopo è fare calcoli (e.g., simulazione)

- La multiprogrammazione è vantaggiosa:
  - sfruttamento di tempi morti
    mentre un processo attende un carattere da tastiera, la CPU può continuare ad eseguire un programma di puro calcolo
  - 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 relativamentre 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.

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 fissata alla creazione dei processi in base alle loro caratteristiche: altà priorità per i processi interattivi, bassa priorità per i processi *CPU bound*

priorità dinamica può essere modificata durante l'esecuzione dei processi

Concorrenza

Esecuzione concorrente:

- suddivisione dei processi su più CPU o su più calcolatori
- condivisione risorse controllata

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

- competizione per le risorse
- cooperazione
  - sincronizzazione
  - comunicazione

Gestione della memoria centrale

Il *gestore della memoria* è il modulo di sistema operativo che fornisce ad ogni processo uno *spazio di indirizzamento virtuale* (può anche essere maggiore della memoria fisicamente disponibile)

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

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

Rilocabilità del codice

- spazio logico: i programmi vengono scritti immaginando di disporre degli indirizzi da 0 in avanti
- spazio fisico: il gestore della memoria realizza un rimappaggio degli indirizzi così da poter eseguire il programma in qualsiasi zona della memoria:
  - rilocazione: operazione di rimappaggio degli indirizzi
  - spiazzamento: indirizzo fisico della cella 0 dello spazio logico

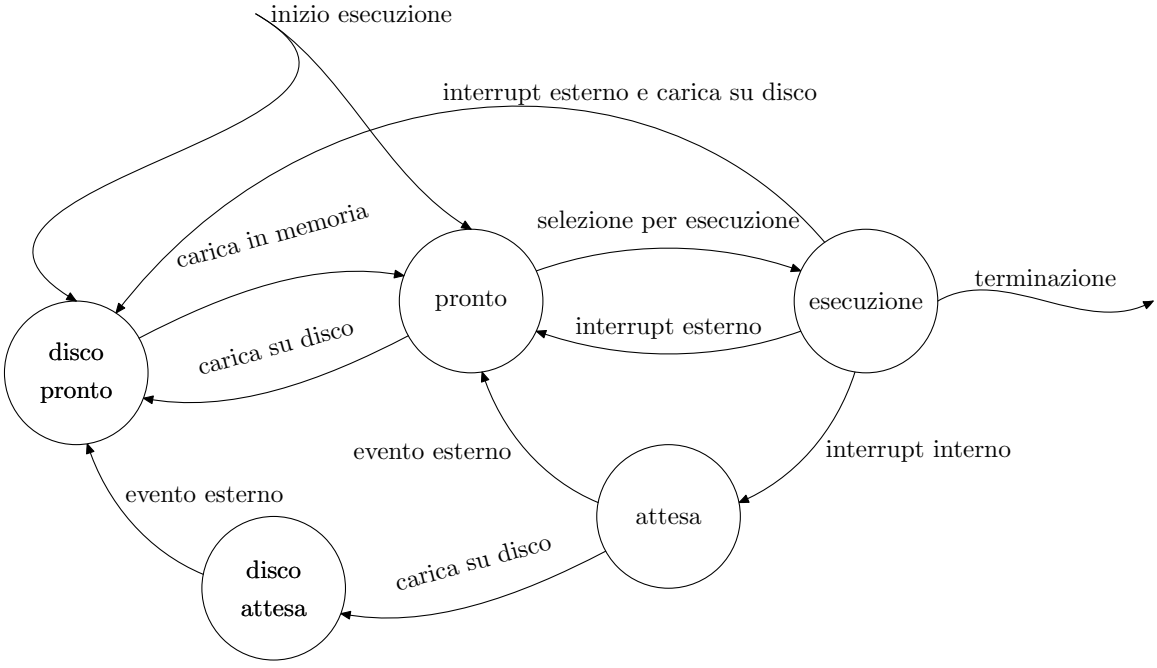
Rilocazione

- rilocazione statica: al momento del linking viene calcolato lo spiazzamento
- rilocazione dinamica:
  - i programmmi vengono caricati in memoria con gli indirizzi calcolati da 0
  - il contenuto del *registro base* viene aggiunto ad ogni istruzione
  - rilocazione di dati e di istruzioni: due registri base (diversi processi con lo stesso registro base istruzioni)

Nota la rilocazione dinamica richiede la disponibilità di apposito HW

Memoria virtuale

uso di spazio disco per coprire la mancanza di memoria centrale (*swap*)



Paginazione

Suddivisione della memoria (sia fisica che logica) in blocchi di ugual dimensione (pagine)

- frammentazione della memoria
- possibilità di allocare zone non contigue di memoria

Dispositivo HW: *Memory Managment Unit* (MMU)

- gestisce la tabella delle pagine e fa l'associazione tra indirizzo fisico e logico
- minimizza i *page fault*

Segmentazione

Suddivisione della memoria usata da un programma:

- segmento codice
- segmento dati
- segmento pila

- ogni segmento viene trattato in modo indipendente dal gestore della memoria
- 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
- e mediante operazioni
  - recupero delle informazioni
  - eliminazione
  - modifica
  - copia