Cenni storici – 1

- Anni '50: i S/O hanno origine con i primi elaboratori a programma memorizzato
  - Modalità di esecuzione *batch* (a lotti) gestita da un operatore umano
    - **Tutto** l'elaboratore a disposizione di **un solo** programma per tutto il tempo della sua esecuzione
    - Immissione di un programma mediante interruttori binari frontali o schede perforate
    - Emissione dei risultati mediante *LED* luminosi a valore binario, testo stampato o schede perforate

Cenni storici – 2

- A partire dagli anni '60
  - Nuovi compilatori, nuovi linguaggi di programmazione, nuovi strumenti di sviluppo
- Ancora gestione a lotti
  - Immissione ed emissione di programmi e dati ancora molto laboriose (= molto costose e molto lente)
    - Un programma può essere sia ingresso che uscita di una esecuzione
  - L'operatore umano è ancora necessario per eseguire le operazioni di ingresso / uscita

Cenni storici – 3

- L'esecuzione di più **lavori** in modalità a lotti può essere facilmente gestita da un S/O **residente**
  - Più caricamenti seguiti da una fase ininterrotta di lavoro e dal ritrovamento dei rispettivi risultati
  - Ordine di esecuzione **predeterminato**
    - Quello di caricamento
    - Secondo il livello di privilegio del richiedente
- Operazioni di I/O fino a 1000 volte più lente dell'elaborazione
  - Esecuzione *off-line* (distinta dal tempo di esecuzione)
  - Sovrapposizione tra I/O ed elaborazione

Cenni storici – 4

- Sovrapposizione sempre più conveniente al crescere delle prestazioni dei dispositivi di I/O
  - Tecnica detta di *spooling* (*Simultaneous Peripheral Operation On-Line*)
  - Si effettua *spooling* quando l'emissione o la ricezione di dati avviene in parallelo all'esecuzione di (altri) lavori
    - Esempio.: invio di una richiesta di stampa, caricamento di un programma, invio di un messaggio di posta elettronica,
      senza interrompere il lavoro in corso

Cenni storici – 5

- Multi-programmazione
  - Desiderabile poter eseguire diversi lavori simultaneamente
    - In ambito mono-processore il parallelismo è solo simulato
  - Occorre controllare l'assegnazione dell'accesso alle risorse della macchina
    - Esempio: per quanti di tempo in modalità *time-sharing* sotto il controllo del S/O

Una definizione di S/O

- Un insieme di utilità progettate per
  - Offrire all'utente un'astrazione più semplice e potente della macchina *assembler* (**macchina virtuale**)
    - Più semplice da usare (p.es., senza bisogno di conoscenze di microprogrammazione ☺)
    - Più potente (p.es., usando la memoria secondaria per realizzare una più ampia memoria principale virtuale)
  - Gestire in maniera **ottimale** le risorse fisiche e logiche dell'elaboratore
    - Ottimalità è la minimizzazione dei tempi di attesa e la massimizzazione dei lavori svolti per unità di tempo

La nozione di processo

- Un processo è un programma in esecuzione e corrisponde a
  - L'insieme **ordinato** di **stati** assunti dal programma nel corso dell'esecuzione (sulla sua macchina virtuale)
    - Processo come "automa a stati"
  - L'insieme **ordinato** delle **azioni** effettuate dal programma nel corso dell'esecuzione (sulla sua macchina virtuale)
    - Processo come "attore" (operatore di azioni)

Realizzazione di processo

- Spazio di indirizzamento logico
  - La memoria della macchina virtuale che il processo può leggere e scrivere (*core*)
    - Memoria virtuale organizzata a pagine e/o segmenti
    - Programma eseguibile
    - Dati del programma
      - Organizzazione dell'informazione in forma di *file*
    - Aree di lavoro
    - Definizione del **contesto di esecuzione** ed area di salvataggio

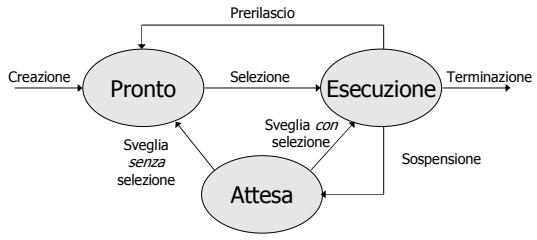
Caratteristiche di processi – 1

- In un sistema coesistono processi utente e di S/O
  - Possono cooperare tra loro ma hanno privilegi diversi
- I processi avanzano concorrentemente
  - Il S/O assegna loro le risorse necessarie secondo diverse politiche di ordinamento
    - A divisione di tempo
    - A livello di priorità (urgenza)
- I processi possono dover comunicare e sincronizzarsi tra loro
  - Il S/O deve fornire i meccanismi e i servizi necessari

Caratteristiche di processi – 2

- Un processo può creare processi "figli"
  - Esempio: un processo interprete di comandi (*shell*) lancia un processo figlio per eseguire il comando inviato da un utente
- I processi vengono
  - **Creati** per eseguire un lavoro
  - **Sospesi** per consentire l'esecuzione di altri processi
  - **Terminati** al compimento del lavoro assegnato
    - Un processo figlio che sopravvive alla terminazione del processo padre è detto "orfano" e può essere molto dannoso

Stati di avanzamento di processo



Gestore dei processi – 1

- Costituisce il cuore o nucleo del S/O (*kernel*)
  - Gestisce ed assicura l'avanzamento dei processi
    - Stato di avanzamento
      - In esecuzione, pronto per l'esecuzione, sospeso in attesa di un evento (una comunicazione, la disponibilità di una risorsa, …)
    - La scelta del processo del eseguire ad un dato istante si chiama ordinamento (*scheduling*)
    - Il gestore decide il cambio di stato dei processi
      - Per divisione di tempo
      - Per trattamento di eventi (p.es., risorsa libera / occupata)

Gestore dei processi – 2

- I compiti del nucleo di S/O sono
  - Gestire le transizioni di stato di attivazione dei processi
  - Gestire le interruzioni esterne causate da
    - Eventi di I/O
    - Situazioni anomale rilevate da altri processi o componenti del S/O
  - Consentire ai processi di accedere a risorse e di attendere eventi

Gestore dei processi – 3

- La politica di ordinamento deve essere equa (*fair*)
  - Processi pronti per eseguire devono avere l'opportunità di farlo
  - Processi in attesa di risorse devono avere l'opportunità di accederle
- I meccanismi e servizi di comunicazione e sincronizzazione devono essere efficaci
  - Il dato (o segnale) inviato da un processo mittente deve raggiungere il destinatario in un tempo breve e in modo sicuro

Definizione di risorsa

- **Risorsa** è qualsiasi elemento fisico (*hardware*) o logico (realizzato a *software*) necessario alla creazione, esecuzione ed avanzamento di processi
- Le risorse possono essere
  - Durevoli (p.es., CPU) o consumabili (p.es., memoria fisica)
  - Ad accesso divisibile od indivisibile
    - Divisibile se tollera alternanza con accessi di altri processi
    - Indivisibili se *non* tollera alternanza
  - Ad accesso individuale o molteplice
    - Molteplicità fisica o logica (virtualizzata)

La risorsa CPU

- Risorsa indispensabile per l'avanzamento di tutti i processi
- A livello fisico (*hardware*) corrisponde alla CPU
- A livello logico (gestione *software*) corrisponde ad una macchina virtuale offerta dal S/O

La risorsa memoria

- Risorsa ad accesso individuale se in scrittura, ad accesso molteplice se in lettura
- La gestione *software* la virtualizza (usando la memoria secondaria) attribuendone l'accesso ai vari processi secondo particolari politiche
- Se virtualizzata, diventa riutilizzabile e pre-rilasciabile
  - Altrimenti consumabile ed indivisibile
- Gestione velocizzata con l'utilizzo di supporto *hardware*

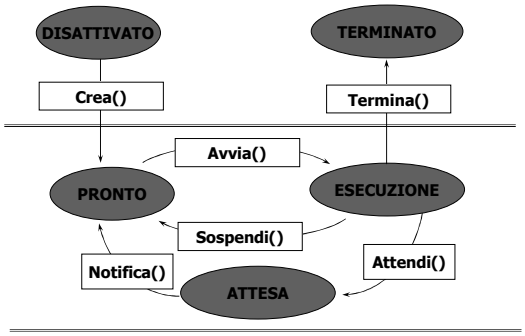
La risorsa I/O

- Risorse generalmente riutilizzabili, non prerilasciabili, ad accesso individuale
- La gestione *software* ne facilita l'impiego nascondendone le caratteristiche *hardware* e uniformandone il trattamento
- L'accesso fisico ha bisogno di utilizzare programmi proprietari e specifici (*BIOS*)

Caricamento del S/O

Il S/O può risiedere

- Permanentemente in ROM
  - Soluzione tipica di sistemi di controllo industriale e di sistemi *dedicati*
- In memoria secondaria per essere caricato (tutto o in parte) in memoria principale all'attivazione di sistema (*bootstrap*)
  - Adatto a sistemi di elevata complessità oppure adibiti al controllo (alternativo) da parte di più S/O
  - In ROM risiede solo il caricatore di sistema (*bootstrap loader*)



Stati – 1

DISATTIVATO

Il programma è in memoria secondaria. Un supervisore lo carica in memoria mediante una chiamata di sistema che crea una struttura di controllo di processo (*Process Control Block, PCB*)

PRONTO

Il processo, pronto per l'esecuzione, rimane in attesa del suo turno

ESECUZIONE

Il processore è stato attribuito al processo selezionato, la cui esecuzione avanza

Stati – 2

ATTESA

Il processo è sospeso in attesa di una risorsa attualmente non disponibile o di un evento non ancora verificatosi

TERMINATO

Il processo ha concluso regolarmente le sue operazioni e si predispone ad abbandonare la sua macchina virtuale

Transizioni – 1

Crea()

Assegna una macchina virtuale a un nuovo processo, aggiornando la lista dei processi pronti (*ready list*)

Avvia()

Manda in esecuzione il primo processo della lista dei pronti

Sospendi()

Il processo in esecuzione ha esaurito il suo quanto di tempo e torna in fondo alla lista dei pronti

Transizioni – 2

- **Attendi()**
  - Il processo richiede l'uso di una risorsa o l'arrivo di un evento e viene sospeso se la risorsa è occupata o se l'evento non si è ancora verificato
- **Notifica()**
  - La risorsa richiesta dal processo bloccato è di nuovo libera o l'evento atteso si è verificato. Il processo ritorna nella lista dei pronti
- **Termina()**
  - Il processo in esecuzione termina il suo lavoro e rilascia la macchina virtuale (e il *PCB* ad essa associato)

Strutture di rappresentazione

- Ogni processo è rappresentato da un **descrittore** (*Process Control Block*) contenente
  - Identificatore del processo
  - Contesto di esecuzione (stato interno) del processo
  - Stato di avanzamento del processo
  - Priorità (iniziale ed attuale)
  - Diritti di accesso alle risorse e privilegi
  - Puntatore al PCB del processo padre e degli eventuali processi figli
  - Puntatore alla lista delle risorse assegnate alla macchina virtuale del processo

Ordinamento di processi

- Diversi metodi sono utili per determinare quando porre un processo in stato di esecuzione in sostituzione di un altro (*switch*)
  - **Scambio cooperativo** (*cooperative* o *non pre-emptive switching*)
    - Il processo in esecuzione decide quando passare il controllo al processo successivo
      - Windows 3.1 ☹
  - **Scambio a prerilascio**: il processo in esecuzione viene rimpiazzata da
    - Un processo pronto a priorità maggiore (*priority-based pre-emptive switching*) → Sistemi detti "a tempo reale"
    - All'esaurimento del suo quanto di tempo (*time-sharing pre-emptive switching*) → Unix, Windows NT (misti!)

Dispatcher – 1

- L'avviatore di processi all'esecuzione (ma non il selettore, *scheduler*!) viene chiamato *dispatcher*
  - Deve essere molto efficiente perché gestisce *ogni* scambio
  - Deve salvare il contesto del processo in uscita, installare quello del processo in entrata (*context switch*) e affidargli il controllo della CPU
- L'efficienza del *dispatcher* si misura in
  - Percentuale di utilizzo della CPU
  - Numero di processi avviati all'esecuzione per unità di tempo
  - Durata di permanenza di un processo in stato di pronto

Dispatcher – 2

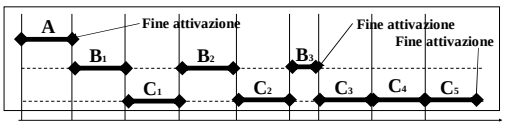
- I processi in stato di pronto sono accodati in una struttura detta lista dei pronti (*ready list*)
- La più semplice gestione della lista è con tecnica a coda (*First-Come-First-Served, FCFS*)
  - Il primo processo ad entrare in coda sarà anche il primo avviato all'esecuzione
  - Facile da realizzare e da gestire
  - La garanzia di esecuzione di altri processi (*fairness*) dipende dalla politica di scambio

Dispatcher – 3

- Le attività di un processo tipicamente comprendono sequenze di azioni eseguibili dalla CPU intervallate da sequenze di azioni di I/O
- I processi si possono dunque classificare in
  - *CPU-bound*
    - Comprendenti *poche* attività sulla CPU e di durata *molto lunga*
  - *I/O- bound*
    - Comprendenti molte attività, di breve durata, sulla CPU, intervallate da attività di I/O molto lunghe
- La tecnica FCFS penalizza i processi della classe *I/O-bound*

Dispatcher – 4

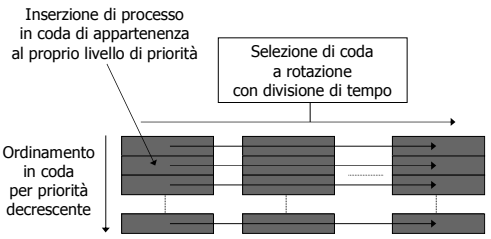
- Imponendo la suddivisione di tempo (*time-sharing*) sulla politica *FCFS* si deriva una tecnica di rotazione detta *round-robin*
- Vediamone l'applicazione su tre processi A, B e C con tempi di esecuzione 2, 5 e 10 ms. e quanto di tempo 2 ms.



Dispatcher – 5

- Ad ogni processo possiamo attribuire una priorità individuale, che denota il suo livello di privilegio
- Processi diversi possono poi essere categorizzati per attributi (p.es.: *CPU- bound*, *I/O- bound*)
- Possiamo allora istituire una coda per ciascuna categoria di processo, ed ordinarla a priorità
- Stabiliamo poi una politica di ordinamento *tra* code (p.es.: *round-robin*)
- Otteniamo una politica di ordinamento a livelli (p.es.: rotazione con priorità)

Politica a rotazione con priorità



Dispatcher – 6

- Possiamo anche facilmente (ed utilmente) definire una politica *duale* alla precedente
  - Istituiamo una coda per ogni livello di priorità attribuita ai processi
  - Selezioniamo la coda a priorità più elevata
  - Applichiamo la politica a rotazione (*round-robin*) sul processo selezionato
  - Otteniamo la politica a priorità con rotazione

Politica a priorità con rotazione