Sistemi Operativi 1

Riccardo Cara

December 11, 2023

Contents

1 Introduzione

Un *sistema operativo* è un software che si occupa di gestire le risorse hardware e fornire software di base che cambiano in base allo scopo del dispositivo. il sistema operativo si occupa di:

- **gestire le risorse hardware** gestire le risose fisiche condivise per raggiungere equità e efficenza
- **virtualizzare delle risorse** astrarre le risorse hardware per renderle disponibili al software sottoforma di risorsa virtuale.
- **interfacciare hardware e software** permette agli utenti di interagire con le risorse hardware senza controllarle direttamente.

e viene diviso in due parti:

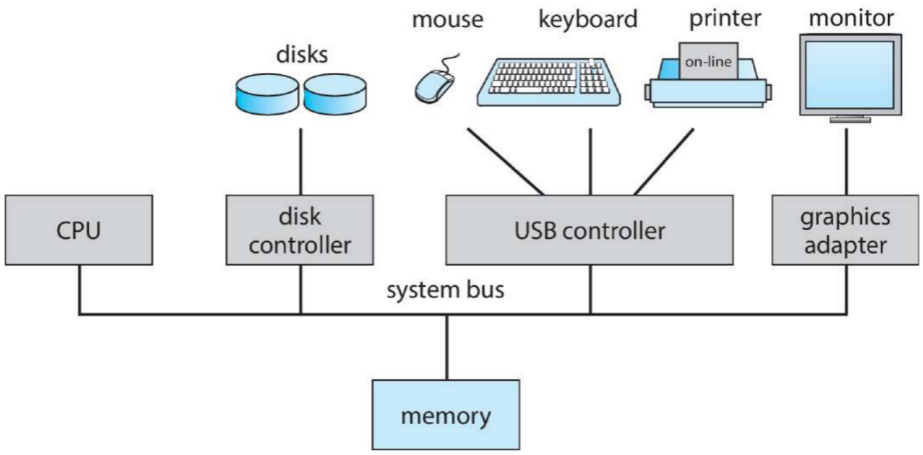
- **software di sistema**: tutti i software che non sono il kernel
- **kernel**: il nucleo dell'OS e si occupa di:
  - **interfaccia hardware** interfacciare i software con l'hardware
  - **gestione della memoria** si occupa di distribuire la ram (Random Access Memory)
  - **gestione dei processi** si occupa della gestione del tempo e quindi permette il multitasking
  - **gestione dispositivi**

  un software effettua delle system calls ("syscall" per chi ha studiato assembly), il kernel traduce le chiamate in serie di comandi e le invia alla cpu.

  Esistono diverse strutture di Sistemi operativi:

- **struttura semplice**: non c'è differenza tra User-mode e Kernel-mode (se ne parlerà più avanti) quindi i software e l'utente hanno accesso completo al computer, questo rende questo tipo di OS come l'MS-DOS poco sicuro.
- **Struttura a macro-kernel**: tutti i software del sistema operativo sono nel kernel e quindi operano in kernel-mode. Solo i software utente lavorano in user-mode
- **Struttura a micro-kernel**: il kernel si occupa solamente di gestire le funzionalità base, come la gestione della memoria, lo scheduling della CPU e la comunicazione tra più processi, il resto dei software del sistema operativo viene effettuato in user-mode
- **Struttura a kernel ibrido**: è un ibrido tra macro-kernel e micro-kernel,
- **struttura a livelli**: Il sistema operativo è organizzato ad anelli, in cui l'anello più piccolo è il kernel e ad ogni anello L i software che vi operano hanno a disposizione le
- **altri**

2 Un sistema computerizzato

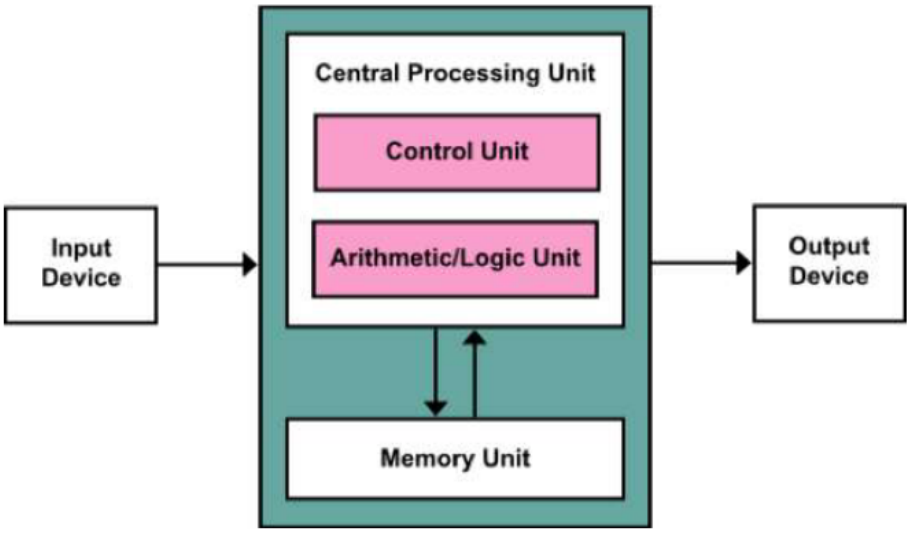


questo è lo schema di un computer, si può osservare che esso è composto da più componenti connessi tra loro tramite un bus di sistema che permette ai componenti del computer di comunicare tra loro. I componenti sono:

- *CPU* la parte che esegue le computazioni
- *memoria principale* immagazzina dati e istruzioni usati dalla CPU, è condivisa tra CPU e dispositivi I/O
- *dispositivi I/O* sono le periferiche, ovvero i dispositivi che permettono al computer di comunicare (monitor, stampante, mouse, tastiera,...)

2.1 Architettura di un computer

L'architettura di un computer è concettualmente identica tra Personal Computers, servers, smartphones ecc ... basato sul concetto di programma memorizzato, ovvero un computer che immagazzina i dati e le istruzioni dei programmi sullo stesso spazio di memoria. Prima di questa architettura, le istruzioni dei programmi e i dati erano salvati su spazi di memoria distinti, basti pensare a quando per cambiare programma si doveva cambiare scheda fisica. (non ne sono sicuro, forse era in un film, comunque il concetto è quello :D).



In questo tipo di architettura, la CPU esegue le operazioni in maniera sequenziale usando registri interni mentre la memoria contiene le istruzioni e i dati.

la CPU esegue tre passi in maniera ciclica:

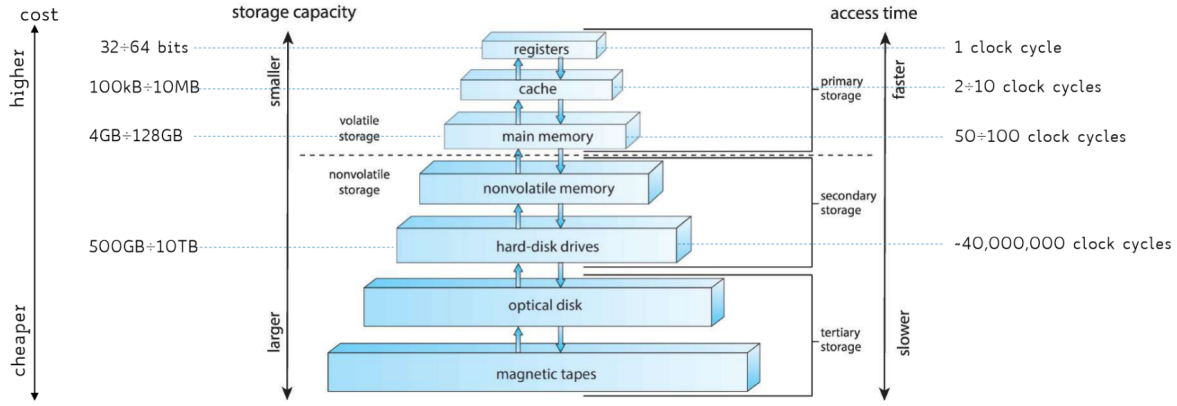
1. **Fetch**: raccoglie l'istruzione dal Program Counter (un registro di memoria speciale della cpu in cui è contenuta l'istruzione da eseguire)
2. **Decode**: interpreta l'istruzione fetchata
3. **Execute**: esegue l'istruzione fetchata

la CPU può essere a processore singolo o a processori multipli, la differenza più grande è che la cpu a processori multipli incrementa il throughput, ovvero, nello stesso periodo di tempo, la cpu a processori multipli da in output una mole di dati maggiore rispetto al processore singolo. In questo corso ci concentreremo sui sistemi a processore singolo.

2.2 Memoria

La CPU possiede delle istruzioni, l'insieme di istruzioni viene definito dal linguaggio macchina. Le istruzioni in linguaggio macchina sono composte da un operatore (op code), zero o più operandi rappresentanti registri di memoria o indirizzi di memoria.

In un Sistema computerizzato ci sono molte memorie, registri di memoria, cache, RAM, spazi di archiviazione(HDD, SSD) ecc. . . queste memorie hanno un sistema gerarchico a livello di costi, capacità di memoria e a livello di tempo impiegato per accedervi:



le memorie sono divise in due sottogruppi:

1. **registri e cache**: sono gestiti dall'architettura
2. **memoria principale, memoria non volatile, spazi di archiviazione**: sono gestiti dal sistema operativo

2.3 Bus di sistema

i bus di sistema sono 3:

- **Data Bus**: porta l'informazione
- **Address Bus**: determina dove l'informazione va inviata
- **Control Bus**: determina quale operazione viene effettuata

Se il control bus è condiviso tra memoria e dispositivi di I/O di cui si parlerà nella prossima sezione, si utilizzerà una linea speciale chiamata"**M/#io**" ed indica se la CPU vuole comunicare con la memoria o con un dispositivo I/O.

2.4 dispositivi I/O

i dispositivi input output, sono divisi in 2 parti, il dispositivo in se e il device controller, ovvero un chip che permette di controllare una famiglia di device controller:

- **SATA controller**: controlla i dischi SATA
- **IDE controller**: controlla i dischi IDE
- **usb controller**: controlla i dispositivi USB
- **PCI bus controller:** controlla i dispositivi connessi al bus PCI
- ...

il sistema computerizzato, comunica con gli elementi I/O tramite dei software chiamati driver: SATA driver, IDE driver, USB driver, PCI bus driver ecc...

2.5 registri

I dispositivi I/O hanno dei registri dedicati con cui comunicare con il sistema computerizzato.

- **registri di stato**: registro in cui viene salvato lo stato del dispositivo(attende l'input, occupato, errore, transazione completata, idle, ecc...)
- **registri di controllo/configurazione**: usato dalla CPU per configurare e controllare il dispositivo
- **registri dei dati**: usato per leggere o inviare dati dal/al dispositivo I/O

La CPU può comuicare con i dispositivi I/O in due modi:

- **Port-mapped I/O** i riferimenti al controller avvengono tarmite uno spazio di indirizzi I/O separato.
  - il registro di ogni device controller è mappato ad una porta (indirizzo) specifica durante il boot
  - necessita di istruzioni della CPU speciali (IN per leggere dal dispositivo I/O, OUT per scriverci)
  - non necessita di interpellare M/#IO per le istruzioni IN, OUT poichè sono solo per i dispositivi I/O e non confondibili con altre istruzioni per la memoria.
- **memory mapped I/O** i registri dei controller vengono mappati sugli stessi indirizzi usati per i registri di memoria.
  - non necessita di istruzioni speciali
  - le porte dei dispositivi I/O sono viste come normali indirizzi di memoria mappati nella RAM
  - per accedere ai registri dei dispositivi I/O vengono usate istruzioni simili a MOV
  - M/#IO indica sempre che gli indirizzi richiesti dalla CPU appartengono alla memoria

2.6 gestione I/O

Esistono due tipi di gestione dei dispositivi input e output. Uno è il *polling* l'altro è l'*interrupt driven*. Il polling è il controllo periodico della cpu sullo stato del dispositivo I/O. l'interrupt driven è un sistema di controllo in cui il dispositivo I/O invia un segnale di interrupt una volta che ha concluso un'azione. Le operazioni di gestione dei dispositivi I/O possono essere eseguiti dalla **CPU** che si occuperà di trasferire i dati oppure da un **DMAC** (Direct Memory Access Controller) solitamente accoppiato con il sistema di interrupt driven allo scopo di rimuovere la CPU dalla gestione dei dispositivi di I/O.

2.7 Servizi del Sistema Operativo

2.7.1 Kernel/User Mode

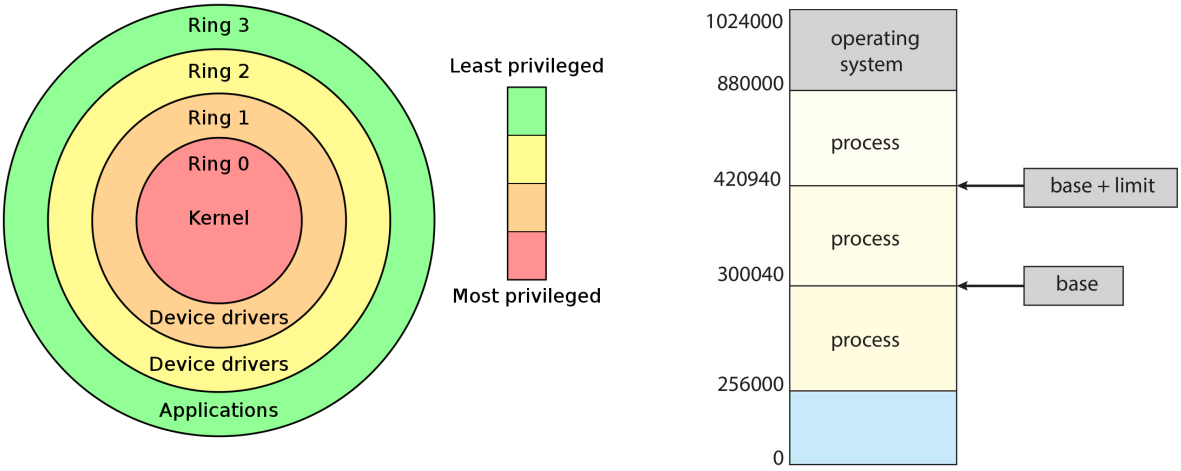
la CPU ha dei comandi che non dovrebbero essere eseguiti da programmi, come esempio HLT per arrestare il sistema o INT X per inviare un interrupt. Per evitare che un qualsiasi programma possa usufruire di questi comandi, essi necessitano di essere eseguiti in **Kernel mode**, la CPU infatti può essere impostata in **User mode** o in kernel mode:

- **Kernel mode** La CPU in kernel mode non ha restrizioni e può quindi eseguire qualsiasi istruzione
- **User mode** la CPU in User mode ha delle restrizioni, quindi **non** può:

– accedere agli indirizzi riservati ai dispositivi I/O
– passare a Kernel mode
– modificare il contenuto della memoria principale
– ...

la modalità cambia in base ad un bit speciale contenuto in un registro protetto, quando il bit vale 0 si è in Kernel mode, quando il bit vale 1 si è in User mode, un esempio di questo cambio di modalità è quando avviene una system call.

Esistono anche altri tipi di protezione, in architetture moderne viene utilizzato un tipo di protezione ad anelli in cui c'è un nucleo che equivale alla Kernel mode e ad ogni anello aggiunto equivalgono più restrizioni.



Oltre alle restrizioni sulle istruzioni eseguite è importante imporre delle restrizioni anche sull'uso della memoria, un programma infatti non dovrebbe poter modificare i contenuti nello spazio di memoria riservato ad un altro programma, come non dovrebbe neanche poter modificare nulla dello spazio di memoria riservato all'OS. Un metodo per proteggere lo spazio di memoria dai programmi è assegnare due registri ad ogni programma nel momento del loro avvio, un **registro base** ed un **registro limite** e controllare che ogni accesso da parte di quel programma alla memoria avvenga nel range di registri tra il registro base e il registro limite.

2.7.2 system calls

System Calls

una system call è una chiamata di sistema effettuata da un programma per richiedere uno dei servizi del sistema operativo, ad esempio stampa su schermo

esistono molte system calls, ma ricadono tutte in 6 categorie:

- **gestione dei file** ad esempio creazione ed eliminazione dei file, lettura e scrittura ecc...
- **controllo dei processi** ad esempio l'esecuzione, la creazione e la terminazione di un processo, l'allocazione e la liberazione di memoria
- **controllo dei dispositivi** ad esempio leggere e scrivere su un dispositivo, connetterlo e disconnetterlo ecc...

- **comunicazioni** ad esempio creazione ed eliminazione di connessioni, invio e ricezione di messaggi ecc...
- **mantenimento delle informazioni** ad esempio il cambio dell'ora, della data, impostare le informazioni dei file ecc...

quando un programma effettua una syscall (System call), la richiede al sistema operativo tramite un **API** ovvero un interfaccia in grado di far comunicare il programma con il sistema operativo. Una volta che il programma ha richiesto la syscall, l'API la convertirà in un interrupt richiamando l'IVT (Interrupt Vector Table) tramite cui verrà avviato il System Call Handler che contiene una tabella con tutte le Syscall possibili e chiama la funzione che implementa la Syscall richiesta. Un qualsiasi evento che richiede il passaggio tra user mode e kernel mode si chiama *trap di sistema*, degli esempi sono:

- **system calls**: appena citate
- **interrupt**: segnali inviati in maniera asincrona da una componente hardware all'os
- **exceptions**: gestione di errori dovuti ad eventi inattesi, svolti in maniera asincrona e innescati dai software

3 Gestione dei processi

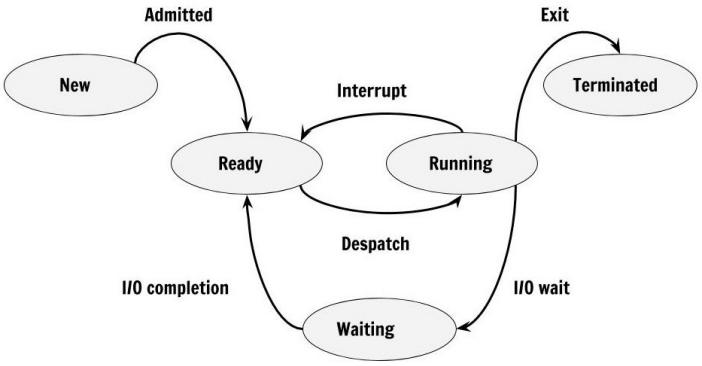
Definizione programmi e processi

Un programma è un file eseguibile che permette di eseguire dei compiti specifici, esso viene salvato in una memoria secondaria detta memoria di massa, come gli HDD o le SSD. Un processo è un azione che un programma esegue, eseguendo in modo sequenziale tutti i processi si conclude il compito del programma, i processi vengono salvati in memoria principale ed eseguiti dalla CPU.

Ogni programma possiede la stessa quantità di VAS (virtual address space) o spazio di indirizzi virtuale, in cui insiemi di indirizzi di memoria sono dedicati a specifici compiti:

- **Text**: contiene le istruzioni dell'eseguibile
- **Data**: contiene i valori statici (fissi) inizializzati all'avvio
- **BSS**: contiene i valori statici non inizializzati all'avvio
- **Stack**: contiene i dati utilizzati da una funzione durante la sua esecuzione
- **Heap**: contiene i dati dinamici ossia dati di dimensione variabile
- **Spazio libero**: utilizzato dallo stack e dall'heap come spazio di espansione

Un processo passa attraverso alcune fasi da quando viene avviato a quando viene terminato:



- **New**: il processo è stato appena inizializzato
- **Ready**: il processo è pronto per essere eseguito dalla CPU, aspetta di essere schedulato
- **Running**: il processo sta venendo eseguito dalla CPU
- **Waiting**: il processo sta aspettando che qualcosa accada, infatti alcuni processi possono aver bisogno di un evento per proseguire, ad esempio un input
- **Terminated**: il processo è stato concluso e viene terminato dall'OS

3.1 Process Control Block

Program Control Block

Il PCB è la struttura dati di un processo, ne traccia lo stato e la posizione in memoria. A più processi corrispondono più PCB.

Un PCB è contiene diverse informazioni, in base all'implementazione possono essere di più o di meno, ma in generale sono presenti:

- **Process state**: contenente lo stato del processo.
- **Process number**: contenente il numero identificativo del processo.
- **Stack pointer, Program counter e altri puntatori**: il puntatore al registro in cima allo stack, il puntatore che indica il processo a seguire, il puntatore al processo padre, il puntatore ai processi figli.
- **informazioni per lo scheduling**: contenente informazioni utili allo scheduling, come l'indicatore di priorità e il puntatore alla state queue.
- **informazioni per il memory management**: contenente informazioni per la gestione della memoria del processo
- **informazioni sullo stato I/O del processo**: auto esplicatorio

3.2 Processo padre e figlio

Processo padre e processo figlio

Un processo può avviare un altro processo, il processo creatore viene detto processo padre, il processo creato viene detto processo figlio.

Ogni processo ha un PID (Process IDentifier), ovvero un numero utile ad identificare un processo ed un PPID (Parent Process IDentifier) ossia un numero utile ad identificare il processo padre. Un ottimo esempio di processo padre e processo figlio è lo sched e l'init in un sistema operativo UNIX: lo scheduler, chiamato *sched* è il primo processo avviato dal sistema operativo, esso ha PID=0, lo scheduler avvia poi il processo *init* con PID=1. Il processo init avvia tutti i processi *daemon* ossia i processi in background e anche tutti i processi relativi al login degli utenti.In questo modo lo sched è il processo antenato, e padre di init, e i processi daemon e di login sono figli di init.

3.2.1 Creazione di un processo

fork() e spawn()

la funzione per creare un processo figlio, cambia in base all'OS:

- Su windows si utilizza la syscall spawn() per creare un processo figlio diverso dal padre, con istruzioni risorse e PCB diversi dal processo generatore (processo padre)
- Su Unix si utilizza la syscall fork() che crea un processo figlio identico al processo padre con le stesse risorse e istruzioni, ma con un PCB differente. Per ottenere lo stesso effetto di spawn(), si può utilizzare exec() dopo il fork(), da come si può intuire, viene creato un processo figlio identico al padre e con exec() si esegue un altro programma

SI OSSERVA CHE Nel codice, la funzione pid=fork(), crea un processo figlio identico al processo padre (le stesse linee di codice), la funzione ritorna 0 al processo figlio assegnando 0 alla variabile pid e ritorna il PID del figlio alla variabile pid del processo padre, in base al valore della variabile pid nel codice si esegue un blocco di codice diverso tramite un if.

3.2.2 Terminazione di un processo

Un processo può terminare se stesso utilizzando la syscall exit (nel codice la funzione "exit()" che ritorna un valore intero nel caso il processo padre stia utilizzando "wait()"), un processo può anche essere terminato dal processo padre o dall'OS nel caso venga utilizzato il comando kill. Quando un processo viene terminato e vengono ritornati i seguenti valori al processo padre in caso esso stia utilizzando "wait()": lo stato di processo terminato, il tempo di esecuzione, codice di esito dell'esecuzione, avviene anche la liberazione di tutte le risorse utilizzate dal processo (le finestre vengono chiuse, i dati vengono de-allocati ecc...).

processo orfano e processo zombie

Quando il processo padre non termina correttamente il processo figlio, possono avvenire 2 cose:

- **Processo orfano**: quando il processo padre viene terminato, ma il processo figlio no, esso viene chiamato processo orfano e viene adottato dall'OS che si occuperà di terminarlo.
- **Processo zombie**: quando il processo figlio tenta di terminare, ma il processo padre non sta usando wait() prende nome di processo zombie, che viene adottato dall'OS per essere poi terminato.

3.3 Comunicazione tra processi

Un processo può essere isolato, ossia può compiere il suo lavoro a prescindere da altri processi, oppure cooperativo, in cui ha bisogno di comunicare con un altro processo per influenzarlo o essere influenzato. Ci sono due modi per il quale più processi possono comunicare.

- **Memoria condivisa**: la memoria di un processo è inizialmente nell'address space di un processo, tramite system calls si permette l'accesso di quella memoria ad un altro processo.
- **Messaggi**: tramite delle system calls i processi si inviano dei "messaggi" contenenti i dati da comunicare, la comunicazione può essere diretta o indiretta.
  - **Diretta** il processo A deve sapere il nome del processo processo B prima di inviare il messaggio e viceversa
  - **Indiretta** i processi comunicano tramite delle porte o un mailbox, non hanno necessità di sapere il nome del processo

  è inoltre necessario specificare la tipologia di queue per i messaggi, vi sono 3 possibili queue:
  - **Zero capacity**: il progaramma 1 invia un messaggio e il programma 2 deve averlo ricevuto prima che il programma 1 ne invii un altro
  - **Bounded capacity**: il processo 1 può inviare un insieme di messaggi limitati al processo 2 a prescindere se esso li abbia ricevuti
  - **Unbounded capacity**: come il Bounded capacity, ma illimitato

  le differenze principali sono che:

- il metodo della memoria condivisa è più lento da inizializzare, ma una volta inizializzato è il più veloce nell'eseguire la comunicazione, è infatti preferibile quando la comunicazione avviene su un singolo computer e la quantità dei dati e la frequenza sono alti.
- il metodo dei messaggi è preferibile quando si devono scambiare pochi dati tra due computers

3.4 Scheduling

Un computer è in grado di gestire più processi e la gestione di essi è affidata ad uno *scheduler* che pianifica in quale ordine i processi vanno eseguiti, questo avviene tramite un algoritmo di schedule, che può eseguire lo schedule in modo tale da favorire (far eseguire prima) processi che occupano la risorsa per meno tempo possibile o fare uno schedule in base a quale processo è stato chiamato prima, secondo uno stack. Per evitare che una task monopolizzi la risorsa, si implementa un timer, il quale allo scadere del tempo, genera un interrupt indicando allo scheduler di passare al prossimo processo. Siccome gli interrupt sono asincroni, potrebbero impedire l'esecuzione di un processo, sta all'hardware garantire che l'esecuzione di un processo non possa essere interrotto da un interrupt.

3.4.1 Process scheduling

Per ogni stato possibile di un processo, vi è un Process State Queue, in cui i PCB vengono messi in coda, quando l'OS cambia lo stato del processo, il PCB del processo viene spostato ad un'altra coda(queue) vi è una coda anche per ogni dispositivo I/O. Il numero di PCB che possono essere contemporaneamente nella running queue dipende strettamente dal numero di core della CPU.

Context Switch

Il context switch è la procedura che la CPU esegue per mettere interrompere temporaneamente un processo in esecuzione permettendo al processo in ready di essere eseguito

1. il processo 1 sta venendo eseguito
2. avviene una system call o un interrupt
3. lo stato del processo 1 viene salvato in PCB_1.
4. il processo 2, primo nello stato di ready viene eseguito
5. avviene una system call o un Interrupt
6. lo stato del processo 2 viene salvato in PCB_2
7. viene ricaricato ed eseguito il processo 1, cosa possibile poichè in PCB_1 si era salvato lo stato del processo 1

3.5 CPU Scheduler

lo scheduler è il processo che si occupa di "pianificare" i processi da eseguire dalla ready queue in base a degli algoritmi. il CPU scheduler può essere:

- **preemptive**: una volta avviato un processo, lo scheduler non può fare nulla se non aspettare che il processo termini
- **non preemptive**: una volta avviato il processo, lo scheduler può decidere se continuare ad eseguire il processo attuale oppure eseguirne uno nuovo

Quando lo scheduler passa il controllo della CPU al processo selezionato, viene utilizzato il dispatcher, un modulo dell'OS a cui è dedicata questa funzione.

3.5.1 tempo dello schedule

Un processo ha una durata di vita molto breve, si è già visto quali sono i suoi stadi, si indecranno adesso, i periodi di vita:

- **arrival time**: il tempo di arrivo di del processo nella ready queue
- **start time**: il tempo di avvio del processo
- **completion time**: il tempo in di fine esecuzione del processo
- **burst time**: il tempo impiegato dalla cpu per eseguire il processo
- **turnaround time**: tempo trascorso tra completion time e arrival time
- **wait time**: tempo trascorso tra turnaround e burst time