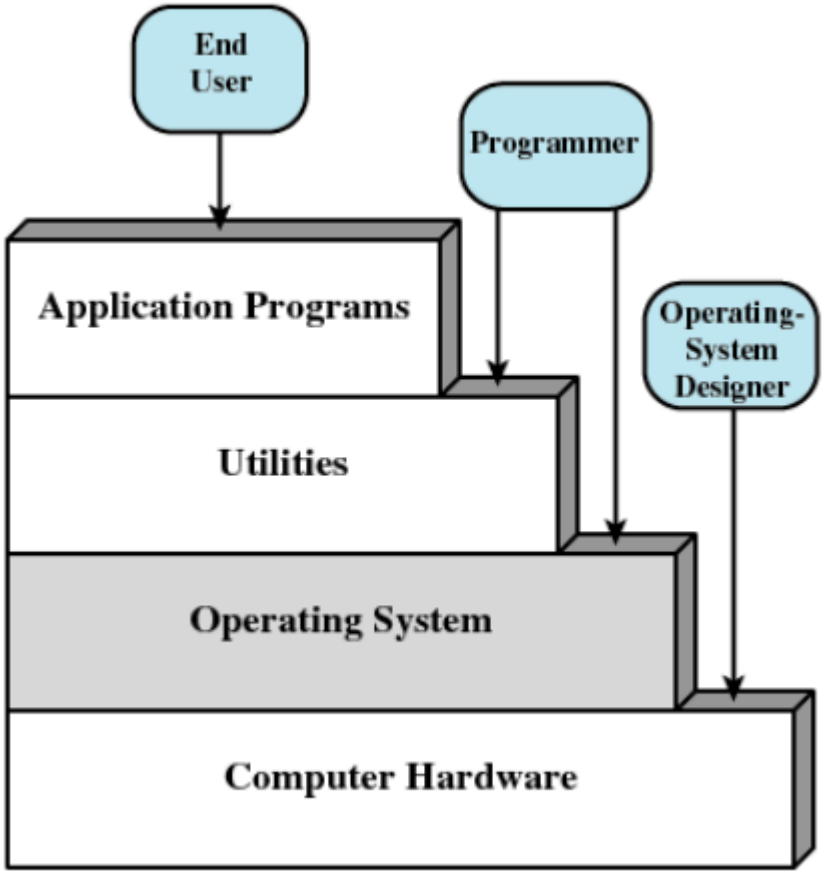
# 6 - Introduzione ai SO

Il sistema operativo deve rendere ***conveniente*** l'uso del calcolatore ai potenziali utenti, deve essere ***efficiente*** nell'utilizzo del calcolatore e delle sue parti costruttive e deve avere la ***capacità di evolversi*** rispetto alle sue evoluzioni hardware e delle esigenze degli utenti

## Sistema operativo come interfaccia

L'obbiettivo principale di un sistema operativo è rendere ***trasparente*** all'utente la parte hardware che usa programmi applicativi




## Servizi offerti dal SO

Il sistema operativo offre diversi servizi all'utente finale:

- **Creazione di programmi**: compilatore, debugger come utilità offerte al programmatore, non sono parte del SO ma sono accessibili tramite esso
- **Esecuzione di programmi**: caricamento in memoria dei programmi, inizializzazione dei dispositivi I/O etc.
- **Accesso ai dispositivi di I/O**: l'utente/programmatore non considera il set di istruzioni e i segnali dei dispositivi

- **Accesso controllato ai file**: comprendere i formati di file (pdf, word etc.), meccanismi di protezione e associazione dei file indirizzi di memoria
- **Accesso al sistema**
- **Rilevazione e correzione degli errori** hardware o generati dai programmi in esecuzione (come overflow)
- **Contabilità e statistiche d'uso delle risorse e dei tempi di risposta** (al fine di migliorare le prestazioni)

# SO come gestore delle risorse (efficienza del SO)

Il sistema operativo è fatto di istruzioni, contenuto in memoria RAM nel suo principio, nel tempo però ci si è resi conto che alcune parti del sistema operativo vengono invocate molto raramente o meno frequentemente;
Definiamo **kernel** (al momento) come parte del sistema operativo residente in memoria centrale che contiene le funzioni usate più di frequente
Il sistema operativo:

- Dirige la CPU nell'utilizzo delle altre risorse del sistema e nella temporizzazione delle esecuzione dei programmi
- Decide quando un programma in esecuzione può usare una risorsa (persino il processore)

## Batch multi programmati

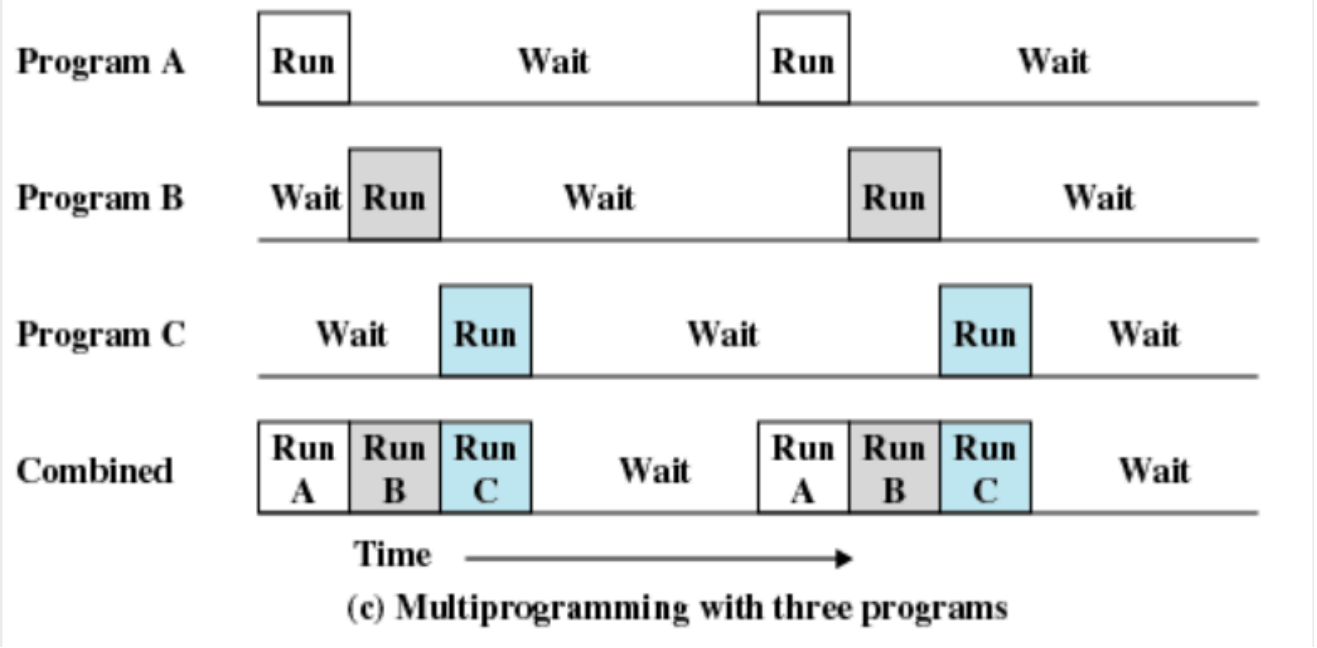
Prima esisteva la mono programmazione, cioè la possibilità di eseguire un unico programma sul calcolatore (ipotizzando sia single core)

| | |
|---|---|
| Lettura di un record | 0.0015 sec. |
| Esecuzione di 100 istruzioni | 0.0001 sec. |
| Scrittura di un record | 0.0015 sec. |
| TOTALE | 0.0031 sec. |

$$\text{Percentuale di utilizzo CPU} = \frac{0.0001}{0.0031} = 0.032 = 3.2\%$$

Poi è nata la multi programmazione, dove sono presenti più programmi in memoria e il suo obbiettivo è limitare l'inattività del processore, quando un job effettua un operazione di I/O la CPU può essere impegnata da un altro processo, in modo da non farla fermare mai.
Il singolo core di prima viene sfruttato in più modi.

### Multi-tasking




Il multi-tasking è l'estensione logica della multi programmazione, la CPU esegue più lavori commutando le loro esecuzioni con una frequenza tale da permettere a ciascun utente di usufruire di tempi di risposta rapidi. Normalmente un processo, durante la sua esecuzione, impegna la CPU per un breve periodo di tempo prima di terminare o richiedere delle operazioni di I/O;
Tali operazioni possono essere interattive (come l'inserimento a tastiera) ma durante questa operazione interattiva il sistema commuta rapidamente la CPU ad un altro processo

## Mono programmazione vs Multi programmazione

Con la *mono-programmazione* può essere mandato in esecuzione solamente un processo per volta rendendo inutilizzata la CPU nel caso in cui il processo stia effettuando operazioni di I/O (acquisizione da tastiera, condivisione di dati in **uscita** verso una stampante ecc...). Il tempo e la percentuale di utilizzo della CPU con questo modello risultano molto elevati.

*Formula che calcola la percentuale di utilizzo della CPU* :
**TO** = tempo totale operazioni(sec.)
**TI** = tempo esecuzione di *n* istruzioni (sec.)

$$\frac{TO}{TI}$$

Attraverso la *multi-programmazione* si possono mandare in esecuzione più processi per volta in base alle caratteristiche hardware della CPU (numero di core). L'obiettivo principale è quello di limitare l'inattività del processore quando un processo effettua un'operazione di I/O impegnandolo con un altro processo, questo farà si che sia possibile elaborare i task in maniera seriale. Ci sono delle difficoltà che si presentano con la multiprogrammazione, quest'ultimi riguardano la ***gestione della memoria*** e delle risorse disponibili poiché potranno essere usate da tutti i processi anche in più istanti diversi ed infine è necessario anche un algoritmo che decida quale **job** mandare in esecuzione (*schedulazione*)

# Processo = Job = Task

Ma cos'è un processo?
Partiamo in ordine:

- **Programma**: serie di istruzioni che operano sui dati
- **Dati**: alcuni sono immediatamente noti (come le variabili), altri sono noti a runtime (o ad esecuzione)
  Un processo è un **programma in esecuzione** che deve essere portato in memoria RAM, capendo quale risorse usa (tra cui la CPU).
  Il processo ha il proprio **contesto di esecuzione**, informazioni necessarie al sistema operativo per gestire il processo come:
- Contenuto dei registri della CPU
- Priorità
- Stato di esecuzione
- Stato di attesa su un dispositivo di I/O

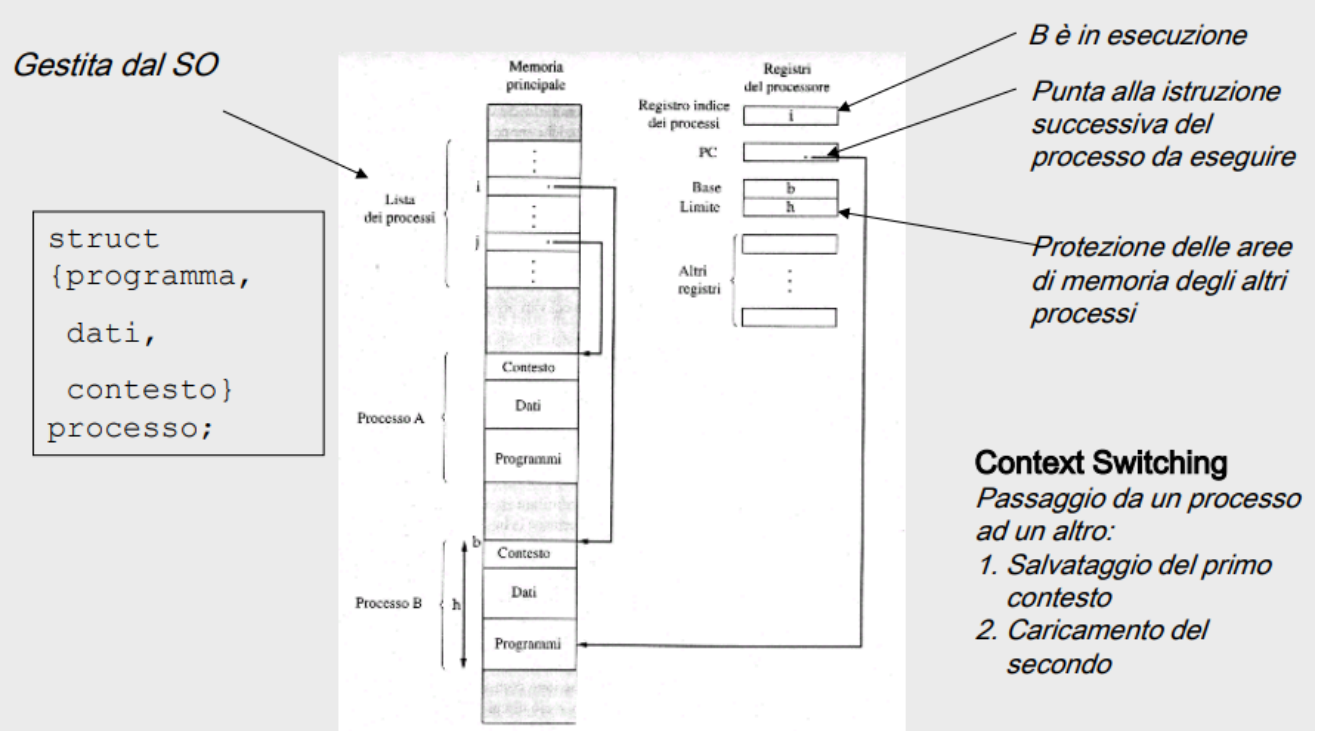
## Implementazione di un processo

Ogni processo è dunque formato da una struttura contenente una serie di componenti (programma, dati, contesto), quest'ultimo per essere eseguito deve essere caricato in RAM, considerando un sistema multi-programmato, tipico dei calcolatori moderni, è possibile che in memoria siano caricati altri processi pronti per essere eseguiti o già in esecuzione.
Nell'istante in cui un processo viene mandato in esecuzione all'interno dei registri della CPU (che sono di numero limitato) devono essere trasferiti tutti i dati necessari al processo che sta per essere eseguito per funzionare, tra questi possiamo trovare:

- PC che punta all'istruzione successiva del processo da eseguire
- L'indice del processo all'interno della *lista dei processi*
- Registri che contengono l'indirizzo *Base* e *Limite* necessari per proteggere le aree di memoria degli altri processi, quest'ultimi infatti indicano il primo e l'ultimo indirizzo delle celle (tutte contigue) di memoria in cui è contenuto il processo

Questi dati sono definiti come il contesto di un processo e consentono anche di riprendere l'esecuzione di un processo che è stato sostituito da un altro da dove si era fermato.
L'operazione di passaggio da un processo ad un altro è chiamata **context switching** e i passaggi principali che deve eseguire sono:

1. Salvataggio del primo contesto
2. Caricamento del secondo




# Gestione della memoria

Il sistema operativo deve assolvere 5 compiti per le memorie:

- **Isolamento dei processi**: i programmi non si interfacciano tra di loro nella memoria centrale, hanno il loro spazio riservato
- **Allocazione e gestione automatica della memoria**: la gerarchia delle memorie deve essere trasparente all'utente
- **Supporto alla programmazione modulare**: in un programma si porta unicamente il main e le funzioni vengono chiamate dinamicamente, caricandole quando se ne ha bisogno. La parte modulare di un programma non ha mai una dimensione variabile ma un processo sì
- **Protezione e controllo dell'acceso**: gestione di aree di memoria condivise tra i processi
- **Memorizzazione a lungo termine**: il file system in breve

Alcune necessità sono soddisfatte da:

- **Memoria virtuale (memoria RAM + parte di Hard Disk)**: i programmi indirizzano la memoria con riferimenti logici ignorando gli aspetti fisici, quando un programma è in esecuzione solo una sua parte risiede effettivamente in memoria centrale. In modo semplice, è il modo in cui il sistema operativo permette di poter far eseguire processi che occupano più memoria di quella presente fisicamente nel sistema (sempre nei limiti possibili).
- **File system**: implementa la memorizzazione a lungo termine

# Schedulazione

La schedulazione è la definizione di un ordine con cui i processi devono essere eseguiti.
La politica di allocazione delle risorse deve considerare i seguenti fattori:

- **Equità**: tutti i processi appartenenti ad una stessa classe, o con richieste simili, o stesso costo, devono avere la stessa possibilità di accesso alla risorsa
- **Tempo di risposta differenziale**: il sistema operativo discrimina tra classi che hanno bisogno di risorse diverse e tempi diversi (come i processi con forte uso di I/O vengono schedulati per primi)
- **Efficienza**: massimizzare il throughput, minimizzare il tempo di risposta