Concorrenza e Mutua Esclusione:
Dentro i processi essitono zone in cui si interagisce con risorse (es. stampante) con le quali bisogna interagire con mutua esclusione = se mando una pagina a stampare, non può entrare un altro e stampare le sue 4 righe in mezzo alle mie.
Se le risorse condivise da più processi gli accessi devono essere gestiti
Più Processi = Meno Tempo in CPU per Processo = Più probabilità che ogni processo venga interrotto nei suoi punti critici

Concorrenza affetta più contesti:

- Applicazioni multiple
- Applicazioni Strutturate = in modo che ci sia competizione e collaborazione tra moduli
- Sistema Operativo = che rappresenta l'insieme dei processi che cooperano x il funzionamento della macchina

Terminologia:

- Operazione atomica = gruppo di istruzioni che non può essere diviso o interrotto a metà, questo provoca una perdita di performance perché c'e' bisogno di sincronizzazione , ovvero un set di istruzioni prima e dopo l'azione atomica
- Sezione Critica = sezione di istruzioni che vanno fatte quando usiamo risorse condivise
- DeadLock = Quando i processi toccano risorse mutuamente in possesso dell'altro , bloccano la loro esecuzione e se questo accade a tutti i processi contemporaneamente abbiamo il deadlock = il blocco completo del sistema
- Race Condition = più processi aggiornano una variabile e avviene la sovrascrizione
- Starvation = I processi vorrebbero finire l'esecuzione ma rimangono bloccati, mentre altri processi continuano l'esecuzione tranquillamente. A differenza del deadlock che riguarda tutto il sistema, questo riguarda solo un sottogruppo di risorse che sono affamate.

Multiprogrammazione:
Output dei processi deve essere indipendente dalla velocità di esecuzione degli altri processi
La velocità relativa di esecuzione è data dai quanti di tempo che la CPU

concede ai processi ed è diversa per ogni processo, dipende da vari fattori.

- Interliving = ogni processo fa un numero di istruzioni per quanto su una CPU
- Overlappin = processi eseguono istruzioni contemporaneamente su CPU diverse = Concorrenza

Difficoltà nella concorrenza:

- Condivisione delle risorse globali
- Allocazioni ottimali di risorse
- Caos nel debug dell'app
    - Se monoprocessore = seguo il processo sequenzialmente fino al bug
    - Se multiprocessore = su ogni processore avrò processo diverso con velocità diverse che condividono variabili e non posso sapere il bug da chi è provocato , da chi e quando.

Competizione:
Per tempo di processore, per clock, per accesso a risorse, per i/o device, per memoria
Cooperazione per Comunicazione : ***

Entro in sezione critica per evitare che gli altri interferiscano.
Gli altri sono quindi in attesa x quella sezione , e quando io finisco c'e' competizione e solo uno di quelli in attesa entrerà nella sezione.

Mutua Esclusione a Livello Hardware:
1)Disabilitazione degli Interrupt : garantisce la mutua esclusione
Svantaggi : efficienza diminuisce, non funzione in un architettura multiprocessore

2)Confronto due celle di memoria x vedere se hanno lo stesso valore e swappo i valori delle celle = atomicamente
Il primo processo che entra nella cpu entra nella sezione critica con bolt = 0 , viene cambiato a 1 nel compareSwap fintanto che ha finito le sue istruzioni, gli altri processi sono bloccati nel while. Quando il primo sbloccherà e metterà nuovamente bolt a 0 allora il primo tra gli altri in attesa che effettuerà il compareSwap entrerà nella sezione critica, questo meccanismo non è deterministico può essere uno tra tutti i processi.

Vantaggi & Svantaggi di istruzioni come Compare&Swap:

- si può applicare a singolo processore che a multiprocessore

- busy-waiting
- starvation
- deadlock = es. Processo A - comp&swap x accedere alla risorsa Ra , poi comp&swap x accedere alla risorsa Rb
- Processo B - comp&swap x accedere alla risorsa Rb , poi comp&swap x accedere alla risorsa Ra
- I due si mettono uno in attesa dell'altro e sono entrambe bloccati.

Se ho deadlock => Starvation
Se ho starvation => non è detto che io abbia deadlock

Semaforo:
Un intero condiviso tra più processi o thread che permette di sincronizzarsi.
Su quest'intero ci si può mettere in attesa ovvero:
Se l'intero è = 0 ,io processo faccio certe cose, finite le cose lo incremento a 1 dove di nuovo qualcun'altra se lo setterà a 0.

Semaforo = variabile con valore intero, su cui sono definite tre operazioni:
1. Inizializzazione = normalmente ad un valore non negativo,
2. SemWait = attesa sul semaforo che decremento il valore = lo mette chi è dentro l'incrocio (es. 0)
3. SemSignal = operazione che si fa in uscita per riaumentare il valore. (es. 1)

Non esiste modo per cambiare questi semafori ma solo ed esclusivamente implementare queste operazioni.
Si creano con i semafori delle code di blocco che si gestiscono con:
SemWait = valore negativo = mi metto in coda di blocco in attesa
SemSignal = incrementa valore negativo = permette a dei processi di entrare in stato ready

La struttura deve essere messa in maniera tale da non creare deadlock

<u>Semaforo Binario:</u>
Il valore del semaforo può avere due valori = 0 e 1
Se sem = 1 = VERDE, processo passa e lo mette a 0 per bloccare gli altri processi

Se altro processo vede sem = 0 = ROSSO, si mette in blocco (che evita il busy-waiting che il compare&swap faceva occupando la CPU), fintanto che non ci sarà una SemSignal per svegliarlo e portarlo in ready
E a quel punto o sem è verde oppure altro processo ha fatto prima e di nuovo è rosso.