È ottimale solo se i lavori da svolgere sono tutti subito disponibili (tempo di attesa nullo)

- Shortest remaining time Next

Simile al SJF ma è preemptive per risolvere il problema legato ai tempi di attesa non nulli
Il principio di brevità è applicato in qualunque momento sul "Remaining time"
Es: Eseguo che vale 5 e arriva a 2 (rimane 3)
Arriva B che vale 2.
A viene sospeso, è eseguito B e poi ripreso A

## Scheduling nei sistemi interattivi ROUND ROBIN

Versione preemptive di FCFS, la gestione è applicata basandosi su un quanto di tempo.
Time slice: tempo max che il job può usare la CPU prima che venga sospeso

Se la coda è
B → F - D - C - A
e finisce il time slice di B → F - D - C - A - B

Conseguenza: Con n processi e un quanto di q ms ogni processo avrà diritto a $\frac{1}{n}$ di CPU e attende al più (n-1)q ms

Quando un processo va in conflitto a un evento di I/O è tolto dalla coda e quando torna è inserito nella coda, se in testa o alla fine è a discrezione del progettista.
Metterlo in testa favorirebbe il sistema sbilanciato a favore dei processi che fanno molto I/O
Valori tipici q = 20-50 ms

# Scheduling a priorità

Regola di base: Si assegna al processo con più alta priorità la CPU

Assegniamo un'etichetta numerica che fissa la priorità di un processo.

Assegnamento priorità

- può essere dinamico o statico
- tende a favorire I/O bounded

* Un processo CPU-bounded tende ad usare sempre per intero il suo q mentre l'I/O-bounded tende ad usarlo parzialmente"

Creiamo un algoritmo dove la priorità p è calcolata come $\frac{1}{f}$ dove f è la frazione di uso della CPU da parte del processo

Se q = 50 ms e lo ha usato per 25 ms ci ha $p = 2$ ($f = \frac{1}{2}$, $\frac{1}{1/2} = 2$)

- SJF può essere visto come un banale algoritmo per priorità dove la p è decisa in base alla brevità.

Nei primi passi dell'algoritmo ovviamente non ci sono le misure di p i processi essere decisi con dei val di default e in pochi istanti la situazione si stabilizza.

E' un sistema dove si potrebbe scegliere di fare o meno la predizione.

Es: Esegue A che ha la p max, arriva B che ha priorità ancora più alta.

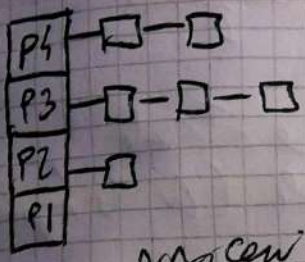
Sospendo A per eseguire B o completo prima finire A?

E' a discrezione del progettista

Se applicassimo solo la regola di base i processi a bassa priorità non verrebbero mai da CPU, ovvero situazione di starvation

Possiamo risolvere la starvation con l'aging
Un processo aumenta la sua priorità ogni volta che rimane ad aspettare nella coda
(il processo vecchio chiede di essere eseguito)
Così permettiamo ai processi a bassa priorità di poter usare ogni tanto la CPU
E' una tecnica meccanica.

Scheduling a code multiple (classi di priorità)




È un insieme di code, quella più in alto ha i processi a priorità superiore.
P4 punta a una coda contenente processi con priorità 4.

Come scelgo un processo?
La scelta si divide in 2 parti

- Quale coda prendere? La coda non vuota più alta
- Quale elemento di questa coda prendo? La tecnica è decidibile. Es: Applicare Round-Robin

Andiamo in contro prima ad uno scheduling verticale e poi uno orizzontale.
In alto stanno per lo più processi I/O-bounded e in basso i CPU-bounded
In base alla natura dei processi della coda scelta si può scegliere meglio quale scheduling orizzontale usare.

Es: Round Robin nella più alta
FCFS nella più bassa.

Nei livelli intermedi è molto ardua Round Robin ma con un quanto che cresce man mano che scendiamo.

~~Disordo sistema~~ Quando ripulisco lo scheduling verticale?

- Se c'è un nuovo candidato che va in una coda più alta
- Se la coda si svuota

Questo sistema può risolvere in modo elegante il problema della starvation.

Applichiamo a ogni coda una percentuale
la CPU si dedica per quella percentuale (di tempo) a quella coda.

| P1 | P2 | P3 | P4 | |
|---|---|---|---|---|
| 5 | 15 | 30 | 50 | % |

Su 10 secondi si dedica 5 su P4, 3 su P3 ecc...

È una soluzione migliore dell'aging

Come si omogenano le priorità nelle code multiple?

- Si usa un sistema a feedback

Si osserva come si comporta un processo, determinando il suo comportamento lo inserisco nella coda opportuna.

Se tende a usare parzialmente il q lo spostiamo sopra, se tende ad usarlo tutto lo spostiamo sotto.

# Shortest Process Next SPN

Applicazione di SJF ai sistemi interattivi

Necessità di un modo per calcolare la durata del prossimo burst di CPU.

Quindi applichiamo l'SJF basandoci sui burst precedenti

Come posso conoscere il burst prossimo di un processo?

Calcolo una stima basata sui burst precedenti

$$S_{n+1} = S_n (1-a) + T_n a$$

S: stima
T: tempo effettivo
$a = \frac{1}{2}$ (esempio)

Questa formula dà peso maggiore alle stime e tempi "vicini" nella ricorrenza

La nuova stima dipende dalla precedente stima e dal precedente tempo effettivo

Se a è prossimo a 1?
Do più peso a T
Vale il viceversa.

La migliore strategia sta nell'avere a né troppo alto né troppo basso.

La stima tende ad avvicinarsi al tempo reale, graficamente parlando

Tramite la formula e i dati ho calcolato

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $T_n$ | 6 | 4 | 6 | 4 | 13 | 13 | 13 |
| $S_n$ | 10 → 8 | 6 | 6 | 5 | 9 | 11 | |