

## Esame del 3/9/2025

Si consideri una memoria condivisa M e un insieme di N processi  $P = \{P_0 \dots P_{N-1}\}$ . Ognuno dei processi  $P_i$  scrive periodicamente un nuovo messaggio sulla memoria condivisa M. Le scritture avvengono in ordine circolare partendo da  $P_0$ , per cui un processo  $P_{(i+1)\%N}$  potrà scrivere un nuovo messaggio solo dopo che il processo  $P_i$  lo abbia scritto. Si consideri inoltre un altro insieme di processi  $Q = \{Q_0 \dots Q_{K-1}\}$  con  $K > N/2$ , i quali leggono i messaggi depositati su M. Ciascuno dei messaggi dovrà essere letto esattamente da  $N/2$  processi dell'insieme Q. Si schematizzi la soluzione del suddetto problema di sincronizzazione, usando solo semafori, fornendo lo pseudo-codice della procedura SCRIVI usata dal generico processo  $P_i$  e della procedura LEGGI usata dal generico processo  $Q_j$ .

### Soluzione

Semafori:

```
turno[N] = {1, 0 ... , 0} // passaggio del turno tra scrittori  
can_write = N/2; // possibilita; di scrittura di un nuovo messaggio  
can_read = 0; // possibilita' di lettura di un nuovo messaggio  
BINARY process_read[K] = {1,1,..., 1}
```

$P_i$  – SCRIVI

```
wait(turno[i]);  
for j = 0, j < N/2 , j++ wait(can_write);  
write to M;  
for j = 0, j < N/2 , j++ signal(can_read);  
signal(turno[(i+1)%N])  
for j = 0, j < k , j++ signal(process_read[j]);
```

$Q_j$  – LEGGI

```
wait(process_read[j]);  
wait(can_read);  
read from M;  
signal(can_write);
```

## Esame del 23/1/2025

Si consideri un insieme di  $N$  processi ( $P_0, \dots, P_{N-1}$ ) i quali comunicano utilizzando una memoria condivisa  $M$  costituita da  $N$  differenti slot  $M[0] \dots M[N-1]$ . Periodicamente il processo  $P_i$  scrive un nuovo messaggio in  $M[(i+1)\%N]$  il quale deve essere letto dal successivo processo  $P_{(i+1)\%N}$ . Prima di scrivere il proprio messaggio,  $P_i$  legge esattamente una volta l'ultimo messaggio scritto in  $M[i]$ , permettendo quindi la possibilità che  $M[i]$  possa ospitare un nuovo messaggio. Nel caso in cui lo slot  $M[(i+1)\%N]$  non possa ospitare un nuovo messaggio poichè l'ultimo messaggio scritto non è ancora stato letto, il processo  $P_i$  deve entrare in attesa di poter effettivamente scrivere il suo nuovo messaggio. Allo stesso tempo,  $P_i$  dovrà entrare in attesa nel caso in cui il nuovo messaggio che lui deve leggere non sia stato ancora depositato. Inoltre si supponga che in tutti gli slot della memoria condivisa  $M$  ci siano inizialmente messaggi validi da leggere. Si schematizzi la soluzione del suddetto problema di sincronizzazione, usando solo semafori, fornendo lo pseudo-codice della procedura LEGGI-SCRIVI usata dal generico processo  $P_i$ .

## Soluzione

Semafori:

$\text{can\_read}[N] = \{1, 1, \dots, 1\}$  ; // possibilità di lettura di un nuovo messaggio

$\text{can\_write}[N] = \{0, 0, \dots, 0\}$ ; // possibilità di scrittura di un nuovo messaggio

$P_i$  – LEGGI-SCRIVI

wait( $\text{can\_read}[i]$ );

read from  $M[i]$ ;

signal( $\text{can\_write}[i]$ )

wait( $\text{can\_write}[(i+1)\%N]$ )

write to  $M[(i+1)\%N]$

signal( $\text{can\_read}[(i+1)\%N]$ )

## Esame del 14/7/2025

Si consideri una memoria condivisa M fatta da N slot e un insieme di N processi  $P = \{P_0 \dots P_{N-1}\}$ . Ognuno dei processi  $P_i$  scrive periodicamente un nuovo messaggio sull'i-esimo slot  $M[i]$  della memoria condivisa. Le scritture avvengono in ordine casuale. Si consideri inoltre un altro insieme di processi  $Q = \{Q_0 \dots Q_{K-1}\}$ . Ognuno dei processi  $Q_j$  legge periodicamente un nuovo messaggio scritto sulla memoria condivisa M. La lettura di ogni processo  $Q_j$  avviene secondo la regola del buffer circolare, leggendo quindi inizialmente da  $M[0]$  ed andando poi sui successivi slot della memoria M. Ogni messaggio scritto da un generico processo  $P_i$  nello slot  $M[i]$  deve essere letto esattamente da tutti i processi nell'insieme Q. Allo stesso tempo, ogni processo  $P_i$  potrà scrivere il suo x-esimo nuovo messaggio dopo che tutti gli altri processi in P abbiano scritto il loro (x-1)-esimo messaggio. Si schematizzi la soluzione del suddetto problema di sincronizzazione, usando solo semafori, fornendo lo pseudo-codice della procedura SCRIVI usata dal generico processo  $P_i$  e della procedura LEGGI usata dal generico processo  $Q_j$ .

## Soluzione

Semafori:

```
round[N] = {N, N, ..., N}
can_write[N] = {K, K, ..., K}
can_read[N] = {0, 0, ..., 0}
readers_roud[K] = {0, 0, ..., 0}
```

$P_i$  – SCRIVI

```
wait(round[i], N)
wait(can_write[i], K);
write to M[i];
signal(can_read[i], K)
for j = 0, j < N ; j++ signal(round[j]);
```

int index = 0; //per process global variable

$Q_j$  - LEGGI

```
wait(can_read[index]);
read from M[index];
signal(can_write[index]);
index = (index+1)%N;
If (index == 0) {
    for i = 0, i < K ; i++ signal(readers_roud[i]);
    wait(readers_roud[j], K)
}
```

## Esame del 16/6/2025

Si descriva la tecnica di gestione della memoria basata su partizioni dinamiche. Si consideri inoltre uno scenario in cui 4 processi {P1, P2, P3, P4} di taglia in Megabyte pari a {2, 3, 1, 3} vengano generati agli istanti di tempo {0, 4, 2, 1} ed abbiano, a partire dal loro caricamento in memoria, turnaround di durata indipendente gli uni dagli altri espressi in secondi come {4, 3, 7, 5}. Si determini il tempo massimo per il completamento di tutti i processi su un sistema di gestione della memoria a partizioni dinamiche, in cui lo spazio destinato ad ospitare i processi sia pari a 5 Megabyte e sia adottato il binding degli indirizzi a tempo di caricamento. Si considerino trascurabili i tempi di tutte le attività di gestione effettuate a livello di sistema e si supponga che il caricamento in memoria avvenga secondo una regola di accodamento di tipo FIFO (First-in First-out).

### Soluzione

Riordiniamo i processi in base al loro tempo di generazione per rispettare la regola FIFO  
{P1, P4, P3, P2}

I tempi di generazione dei processi risultano essere  
{0, 1, 2, 3}

Le durate (dal caricamento risultano essere)  
{4, 5, 7, 3}

Le richieste ordinate di utilizzo di memoria (nuove partizioni) in Megabyte saranno quindi  
{2, 3, 1, 3}

P1 e P4 non subiscono ritardi di caricamento.

P1 e P4 impongono un delay al caricamento di P3 pari al  $\min\{4, 6\}$

Quindi all'istante 4 la memoria avrà i suoi 5 Megabyte nel seguente stato (Busy o Free) {P3, X, P4, P4, P4}

Per poter caricare P2 dovremo attendere il completamento di P4 all'istante 6. A quell'istante P3 sarà ancora attivo fino all'istante 11 che supera l'istante di completamento di P2, ovvero 9.

Il tempo totale per l'esecuzione risulta quindi essere pari a 11.

## Esame del 8/2/2022

Si descriva lo scheduler di CPU multi-level feedback-queue. Inoltre, si consideri uno scenario in cui all'istante  $T_0$  esistano 2 processi  $P_1$  e  $P_2$ .

Il processo  $P_1$  é CPU-bound di durata infinita, mentre il processo  $P_2$  é I/O-bound ed interagisce infinite volte con un dispositivo  $D$  prima di completare la sua esecuzione. Il CPU-burst di  $P_2$  sia di 5 millisecondi. Nel caso lo scheduler multi-level feedback-queue abbia 2 livelli di priorità, si determini motivando la risposta quale sia il massimo valore di time-slice che garantisca un tempo di attesa di  $P_2$  non superiore a 10 millisecondi.

Si assuma che la latenza di esecuzione dello scheduler di CPU sia nulla.

### Soluzione

Considerando una strutturazione classica dello scheduler multilevel-feedback queue si può supporre che

- TS sia il time-slice al livello di priorità 0
- $2 \times TS$  sia il time-slice al livello di priorità 1

Il tempo di risposta del dispositivo  $D$  é di fatto arbitrario (poiché non ci sono indicazioni o ipotesi relative ad esso all'interno del testo)

Esistendo solo 2 processi ( $P_1$  e  $P_2$ ) se questi sono entrambi non-blocked l'assegnazione della CPU avviene in modo alternato ad ogni specifico livello di priorità

Il tempo di attesa massimo é quindi determinabile in base al massimo utilizzo del time-slice da parte dei processi in quel livello di priorità, pertanto se  $P_2$  ritorna nello stato ready in quel livello di priorità quello che determina il tempo di attesa é il residuo di utilizzo del time-slice da parte di  $P_1$ .

Il rientro nello stato ready da parte di  $P_2$  avviene in un istante arbitrario, quindi nel caso peggiore otteniamo di dover attendere per tutto il time-slice, che nel caso di priorità 1 é pari a  $(2 \times TS)$  ne deriva quindi che

$$(2 \times TS) < 10 \text{ quindi } TS < 5$$

Tale disequazione determina anche che transiti wait->ready da parte di  $P_2$  avverranno solo quando  $P_2$  si troverà a livello di priorità 1, che é esattamente il livello massimo, per il quale si ha la predetta alternanza di esecuzione di processi non-blocked

## Esame del 16/9/2021

Si descriva il metodo di allocazione dei file indicizzato. Si consideri un file system con metodo di allocazione dei file indicizzato a 2 livelli. Il record di sistema che tiene traccia di ogni file ha 6 indici diretti e 4 indici indiretti. Il dispositivo di memoria di massa che ospita il file system ha blocchi di taglia pari a 2048 byte. Si calcoli la taglia massima, espressa in byte, di un file gestibile su tale file system considerando i due scenari dove

1) l'indice che identifica un blocco all'interno del dispositivo abbia taglia pari a 8 byte,

oppure

2) abbia taglia pari a 4 byte

## Soluzione

La porzione del file gestita tramite indici diretti e' indipendente dalla taglia dell'indice (dipende esclusivamente dalla taglia del blocco del dispositivo), in particolare tale porzione P del file ha taglia massima pari a

$$P = 6 \times 2048 \text{ byte}$$

Per la porzione del file gestita tramite indici indiretti la taglia dipende dalla taglia degli indici di secondo livello poiche' il loro numero e' funzione della taglia del blocco.

Abbiamo quindi due scenari

1) indici di taglia pari a 8 byte – in questo caso ciascuno dei 4 blocchi del dispositivo che mantengono indici di secondo livello ne ha  $2048/8 = 256$  permettendo una taglia della porzione P' di file gestibile pari a  $P' = 4 \times 256 \times 2048 \text{ byte}$ , per un totale della taglia del file di  $P + P' = 6 \times 2048 + 4 \times 256 \times 2048 \text{ byte}$

2) indici di taglia pari a 4 byte – in questo caso ciascuno dei 4 blocchi del dispositivo che mantengono indici di secondo livello ne ha  $2048/4 = 512$  permettendo una taglia della porzione di file gestibile di  $P' = 4 \times 512 \times 2048 \text{ byte}$ , per un totale della taglia del file di  $P + P' = 6 \times 2048 + 4 \times 512 \times 2048 \text{ byte}$

## Esame del 16/9/2021

Si descriva lo scheduler di CPU round-robin. Inoltre, si consideri uno scenario in cui all'istante  $T_0$  siano attivati  $N$  processi  $P_1 \dots P_N$ . Il processo  $P_1$ , di durata infinita, è I/O bound e ha CPU-burst pari a 10 millisecondi. Esso interagisce con un dispositivo  $D$  che ha tempo di risposta pari a 15 millisecondi. Tutti gli altri processi sono CPU-bound di durata infinita. Si determini il tempo di attesa massimo che il processo  $P_1$  potrà avere considerando

1) il caso di time-slice dello scheduler pari a 5 millisecondi oppure

2) il caso di time-slice dello scheduler pari a 20 millisecondi.

Si ipotizzi che il tempo per l'esecuzione dello scheduler e per la gestione del dispositivo a livello del software del sistema operativo sia nullo.

## Soluzione

Il tempo di attesa del processo corrisponde al tempo che intercorre tra il raggiungimento dello stato Ready e l'effettiva schedulazione in CPU (passaggio nello stato Running)

1) Nel primo caso, ogni volta che  $P_1$  eseguirà una interazione con il dispositivo, esso verrà reinserito nella ready-queue esattamente al termine di un time-slice, poiché il tempo di risposta del dispositivo è esattamente un multiplo del time slice. In tal caso,  $P_1$  avrà tutti gli altri  $N-1$  processi registrati prima di esso nella ready-queue poiché in quell'istante l'ultimo processo Running transita nello stato ready. Quindi il tempo di attesa massimo di  $P_1$  sarà  $(N-1) \times 5$  millisecondi. Questo valore copre anche lo scenario di attesa della CPU prima ancora di eseguire la prima interazione col dispositivo  $D$ , o nel caso di preemption applicata a  $P_1$ , lecita a causa del fatto che il CPU-burst di  $P_1$  ha lunghezza superiore al time-slice.

2) Nel secondo caso, ogni volta che  $P_1$  eseguirà una interazione con il dispositivo, esso verrà reinserito nella ready-queue quando il processo attualmente in CPU avrà ancora 5 millisecondi di tempo per eseguire (ovvero  $20 - 15$ ). In tal caso,  $P_1$  avrà  $(N-2)$  processi avanti ad esso nella ready-queue, poiché uno dei processi è; nello stato running, sperimebtabto quindi un tempo di attesa pari a  $20 \times (N-2)$  millisecondi. Il massimo del tempo di attesa può comunque essere  $20 \times (N-1)$  nel caso in cui l'assegnazione iniziale della CPU non preveda un ordinamento dei processi nella ready-queue in cui  $P_1$  si inserisce non come ultimo processo. Il caso di preemption è da escludere poiché il CPU burst di  $P_1$  (pari a 10 millisecondi) è inferiore al time-slice (pari a 20 millisecondi).