Università degli Studi della Basilicata

Corso di Laurea in Scienze e Tecnologie Informatiche

Sistemi Operativi - A.A. 2019/2020

Simulazione di esame

27/01/2020

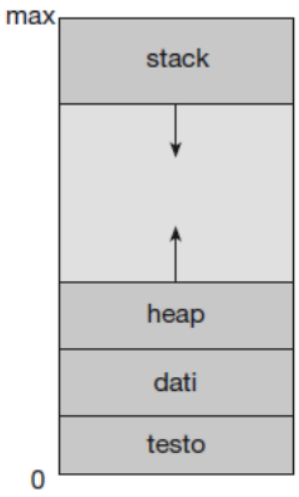
**Domanda 1 (max 5 punti)**

Descrivere il concetto di processo in un sistema operativo. Spiegare, inoltre, cosa sia un Process Control Block (PCB) utilizzando opportuni schemi grafici.

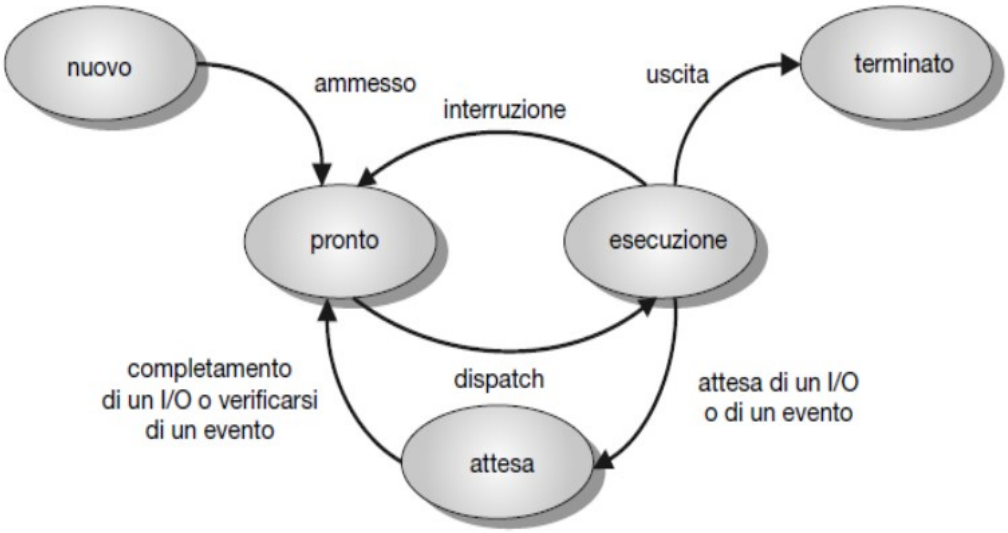
In un sistema operativo, un processo è un programma in esecuzione. La struttura di un processo in memoria è generalmente suddivisa in sezioni:

- Sezione di testo, contenente il codice eseguibile
- Sezione dati, contenente le variabili globali
- Heap, memoria allocata dinamicamente durante l'esecuzione del programma
- Stack, memoria temporaneamente utilizzata durante le chiamate di funzioni

La figura a lato mostra la tipica struttura di un processo in memoria




Un processo viene creato dal sistema operativo e attraversa diversi stati prima della sua terminazione. La figura seguente mostra gli stati in cui può trovarsi un processo

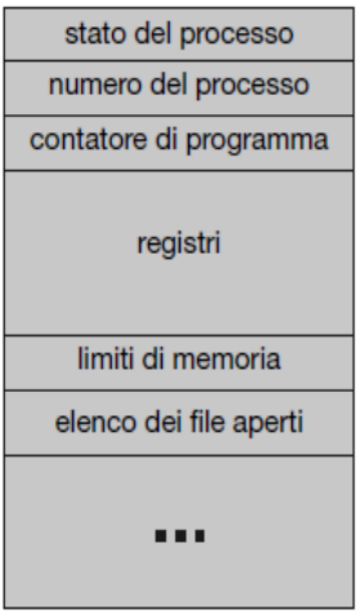


Il sistema operativo rappresenta con un process control block (PCB) ogni processo. Il PCB contiene molte informazioni connesse a uno specifico processo. Tali informazioni includono:

**Stato del processo** → nuovo, pronto, esecuzione, attesa, arresto

**Contatore di programma** che contiene l'indirizzo della successiva istruzione da eseguire per tale processo.

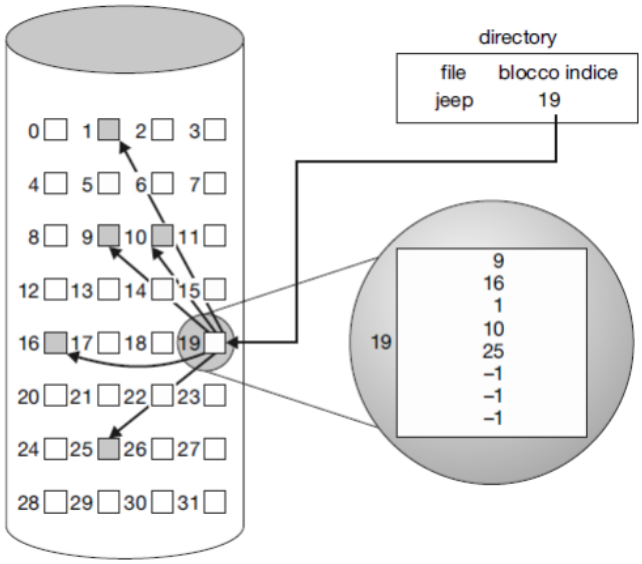
**Registri della CPU** → accumulatori, registri indice, puntatori alla cima dello stack (*stack pointer),* registri d'uso generale e registri contenenti i codici di condizione (*condition codes*)




**Domanda 2 (max 5 punti)**

Descrivere la modalità di allocazione indicizzata dei file in memoria secondaria utilizzando opportuni schemi grafici.

L'allocazione indicizzata è una modalità di assegnazione della memoria secondaria che fa uso di un blocco indice per raggruppare i puntatori ai blocchi contenenti i dati del file. La figura seguente mostra lo schema di allocazione con uso di blocco indice.




Il blocco 19 contiene i puntatori ai blocchi dei dati (9,16,1,10 e 25). Poiché il file jeep occupa 5 blocchi di dati, tre degli 8 puntatori del blocco indice sono inutilizzati.

Tra i vantaggi dell'uso dell'allocazione indicizzata troviamo:

- Facilità di creare, ridurre e far crescere la dimensione dei file
- Frammentazione esterna contenuta
- Possibilità di supporto per accesso diretto

Tra gli svantaggi annoveriamo:

- Overhead per file di ridotte dimensioni
- Difficoltà nella gestione di file di grossa dimensione

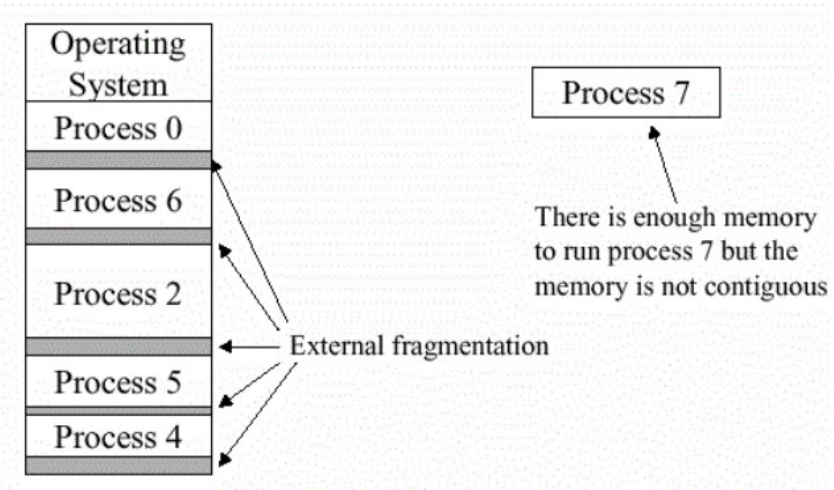
**Domanda 3 (max 5 punti)**

Spiegare la differenza tra frammentazione interna e frammentazione esterna utilizzando opportuni esempi.

Frammentazione interna ed esterna sono due problematiche relative all'allocazione di memoria (sia primaria che secondaria).

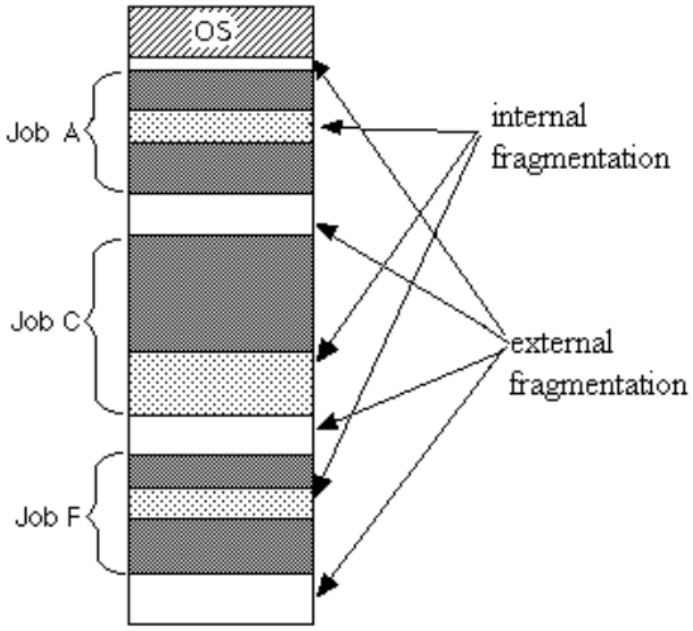
La frammentazione esterna si verifica quando la quantità totale di memoria allocabile è sufficiente per esaudire una richiesta, ma la memoria disponibile non è contigua e quindi non può essere allocata.

Si faccia riferimento all'esempio in figura. Anche se la somma totale dei blocchi di memoria disponibili è sufficiente a coprire il fabbisogno del Processo 7, tale processo non può essere allocato perché non ci sono porzioni contigue di memoria allocabile abbastanza grandi da contenere il processo 7. La memoria che risulta inutilizzabile si trova al di fuori (all'esterno) dei processi in esecuzione (che sono 0, 6, 2, 5, 4) e viene a crearsi quindi una frammentazione esterna.




La frammentazione interna si verifica quando un blocco di memoria assegnato ad un processo risulta più grande del necessario. In tal caso, la memoria risulta sprecata poiché, essendo interna allo spazio di memoria riservato ad uno specifico processo, non può essere utilizzata da altri processi.

La figura seguente illustra un esempio di frammentazione interna in memoria centrale. Inoltre, la figura mette in evidenza la differenza tra frammentazione esterna, descritta sopra, e frammentazione interna.




**Esercizio 1 (max 7,5 punti)**

Sia data la seguente successione di riferimenti alle pagine di memoria:

3, 1, 5, 9, 1, 2, 3, 1, 3, 2, 9, 2, 7, 3

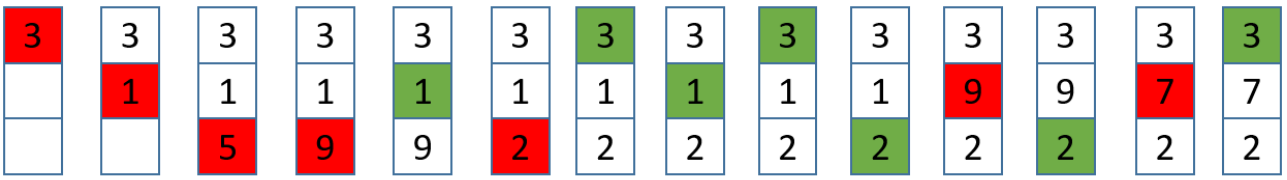
Si assuma

- di avere una tabella delle pagine di 3 elementi, gestita con politica Optimal Replacement
- che $T_{ma}$ e $T_{pf}$ siano rispettivamente i tempi di accesso in memoria e di gestione del page fault

1. Qual è il tempo di accesso effettivo in memoria per la situazione descritta?

2. Qual è la probabilità di avere un page fault?

L'algoritmo di sostituzione delle pagine Optimal Replacement consiste nel sostituire la pagina che non verrà usata per il periodo di tempo più lungo. Si tratta di un algoritmo che non può essere usato in pratica, ma che è molto utile per effettuare test comparativi. La figura seguente mostra l'evoluzione della tabella delle pagine per il caso di studio (page fault in rosso, page hit in verde).



Per calcolare il tempo di accesso effettivo, utilizzeremo la formula seguente.

$$\textit{tempo di accesso effettivo (EAT)} = \textit{n\_page\_hit} \times T_{ma} + \textit{n\_page\_fault} \times T_{pf}$$

Quindi nel caso di studio avremo EAT = 7 x $T_{ma}$ + 7 x $T_{pf}$

La probabilità di page fault sarà pari a 7 / 14 = 50%

**Esercizio 2 (max 7,5 punti)**

Sia S un semaforo inizializzato a 0 e si considerino due programmi in esecuzione concorrente composti dalle seguenti sequenze di istruzioni

| Programma A | Programma B |
|---|---|
| `wait(S)` | `print("x")` |
| `print("y")` | `signal(S)` |
| `wait(S)` | `print("z")` |
| `print("t")` | `signal(S)` |

Cosa verrà stampato? Motivare la risposta.

Iniziamo ricordando che un semaforo `S` è una variabile intera cui si può accedere, escludendo l'inizializzazione, solo tramite due operazioni atomiche predefinite: `wait()` e `signal()`

La `wait` e la `signal` sono così definite:

```
wait(S) {
    while(S <= 0)
        ; /* busy wait */
    S--;
}
```

```
signal(S) {
    S++;
}
```

Essendo A e B in esecuzione concorrente bisognerà simulare tutte le possibili alternative di scheduling. Inoltre, si ipotizza che le operazioni di `print` vengano eseguite immediatamente senza alcuna bufferizzazione.

Se A va per primo in esecuzione, rimarrà bloccato sulla `wait`. Per tale motivo, sicuramente `print("x")` verrà eseguita qunado B prenderà la CPU. Poi verrà eseguita la prima `signal`. Dopo tale esecuzione, S varrà 1 e quindi A potrà essere sbloccato. Tuttavia, non siamo sicuri che `print("y")` verrà eseguita prima di `print("z")`, poiché questo dipenderà dallo scheduling della CPU. L'unica cosa di cui siamo certi è che `print("t")` non potrà essere eseguita prima di `print("z")`. Per i motivi di cui sopra, il risultato della esecuzione di A e B potrà essere:

`xyzt`

oppure

`xzyt`