

Quesito 1 (punti 4).

[A] Quale tra le seguenti affermazioni è corretta in relazione alla politica di ordinamento processi FCFS senza valutazione dell'attributo di priorità?

1. il tempo di attesa è sempre maggiore del tempo di risposta
2. il tempo di attesa è sempre minore del tempo di risposta
3. il tempo di attesa è sempre uguale al tempo di risposta
4. nessuna delle precedenti

[B] Quale tra le seguenti politiche di ordinamento, in generale minimizza il tempo medio di attesa dei processi?

1. FCFS
2. Round-Robin con valutazione dell'attributo di priorità dei processi
3. Round-Robin senza valutazione dell'attributo di priorità dei processi
4. Shortest Job First.

[C] Quale tra le seguenti affermazioni concernenti la politica di ordinamento Round-Robin è corretta?

1. il tempo di attesa è sempre maggiore del tempo di risposta
2. il tempo di attesa è sempre minore del tempo di risposta
3. il tempo di attesa è sempre uguale al tempo di risposta
4. nessuna delle precedenti

[D] Quale tra le seguenti politiche di ordinamento in generale minimizza il tempo medio di risposta dei processi?

1. First Come First Served (senza prerilascio, fino al completamento)
2. Round Robin
3. Shortest Job First (senza prerilascio)
4. Accodamento FIFO con priorità e prerilascio.

Quesito 2 (punti 8). Sopra un profondo dirupo che separa due sponde di un territorio di montagna un comando di guardia forestale ha teso un ponte di corde così stretto da impedire a due persone di affiancarsi per superarsi vicendevolmente. Tale ponte permette dunque per progetto esclusivamente il passaggio di persone in una sola direzione.

Proporre a quel comando di guardia forestale un algoritmo di controllo degli accessi a entrambi i lati del ponte, capace di garantire l'assenza di *deadlock* e di *starvation*, fornendo anche la relativa dimostrazione.

Quesito 3 (punti 6). Si consideri un elaboratore dotato di un sistema di memoria virtuale paginata e di un processore la cui frequenza di ciclo di *clock* sia 1 MHz. Si assuma che detto processore impieghi un ciclo di *clock* per eseguire istruzioni che non comportino riferimenti a pagine di memoria diverse da quella corrente. Si assuma inoltre che valgano le seguenti ipotesi:

- l'accesso a una pagina di memoria diversa da quella corrente comporti un costo temporale aggiuntivo di 1 μ s;
- ciascuna pagina di memoria sia composta di 1.000 B;
- il disco fisso ruota alla velocità di 6.000 rpm (rotazioni/minuto);
- il trasferimento tra disco e memoria avvenga a 1.000.000 B/s;
- il tempo medio per spostare la testina dalla posizione corrente fin sopra la traccia dove si trova il punto di lettura/scrittura su disco sia di 10 ms;
- un blocco su disco corrisponda in dimensione esattamente a una pagina di memoria virtuale;
- 98% delle istruzioni eseguite facciano riferimento alla pagina corrente;
- 80% delle pagine accedute (diverse dalla corrente) si trovi già in memoria;
- quando una nuova pagina debba essere caricata in memoria, nel 50% dei casi quella rimpiazzata sia stata precedentemente modificata.

Si calcoli il tempo medio effettivo di esecuzione di ciascuna istruzione su tale elaboratore assumendo che il sistema stia eseguendo un unico processo e che il processore rimanga inattivo (stato *idle*) durante i trasferimenti di dati.

Quesito 4 (punti 8). Uno studioso è interessato a valutare l'effetto algoritmico e prestazionale della concorrenza sulla realizzazione a programma del calcolo della successione di Fibonacci. Aiutare l'audace studioso specificando in un pseudo-linguaggio a piacere e con primitive di controllo di concorrenza ugualmente a piacere, ma coerenti con un qualche standard reale (per esempio C e `fork()`) un sistema multiprogrammato che effettui tale calcolo secondo il seguente principio: il processo padre riceve in ingresso l'intervallo di calcolo, calcola il primo termine ($fib_0 = 0$) e il secondo termine ($fib_1 = 1$) della successione, genera un processo figlio e gli passa l'indice di limite e i due termini appena calcolati, e poi stampa i valori calcolati; il processo figlio valuta l'indice e se necessario produce il termine successivo ($fib_2 = fib_1 + fib_0$), poi genera un processo successore e gli passa l'indice di limite e gli ultimi due termini della sequenza, e infine stampa il valore calcolato, e così via. (Ricordiamo che nella successione di Fibonacci si ha $fib_n = fib_{n-1} + fib_{n-2}$ per $n \geq 2$.)

Dimostrare che nella soluzione proposta i dati vengano passati in modo corretto tra il processo padre e il processo figlio e che i processi del sistema eseguano e terminino in modo ordinato e corretto.

Discutere l'utilità della multiprogrammazione per questo tipo di calcolo.

Quesito 5 (punti 6). Il progettista di un sistema operativo ha deciso di usare nodi indice (*i-node*) per la realizzazione del proprio *file system*, stabilendo che essi abbiano la stessa dimensione di un blocco, fissata a 512 byte. Il progettista ha poi deciso che un nodo indice primario contenga 12 campi di indirizzo di blocchi di disco e 2 campi puntatori a nodi indice di primo e secondo livello di indirizzazione rispettivamente. Sapendo che gli indirizzi di blocco sono espressi su 32 bit, si vuole allocare un *file* logicamente composto da 10.000 *record* da 80 byte ciascuno, imponendo che un *record* non possa essere suddiviso su due blocchi. Calcolare quanti blocchi verranno utilizzati per allocare il *file* dati e quanti per gestire la sua allocazione tramite nodi indice. Determinare inoltre l'occupazione totale in memoria secondaria risultante da tale strategia di allocazione.