Esame di Sistemi Operativi
AA 2017/18
23 Gennaio 2018

Giorgio Grisetti

Istruzioni

Scrivere il proprio nome e cognome su ogni foglio dell'elaborato. Usare questo testo come bella copia per le risposte, utilizzando l'apposito spazio in calce alla descrizione dell'esercizio.

Esercizio 1

Sia data la seguente tabella che descrive il comportamento di un insieme di processi.

processo	tempo di inizio	CPU burst	IO burst
P1	0	5,1	10,10
P2	1	5,5	10,10
P3	3	1,7	5,1
P4	4	3,3	1,1

Domanda Si illustri il comportamento di uno scheduler di tipo SJF ed uno scheduler di tipo FIFO, utilizzando il grafico a disposizione nelle Figure 1 e 2. Ciascun processo alterna CPU burst ed IO burst, secondo la tabella illustrata sopra.

Soluzione Lo scheduler di tipo SJF produrra' la traccia di esecuzione dei processi illustrata in Figura 1. Lo scheduler FIFO, invece, la traccia raffigurata in Figura 2. In entrambe le figure avremo in verde i cicli di *cpu burst* ed in rosso l'*I/O burst*; le caselle azzurre indicano l'arrivo di un nuovo processo.

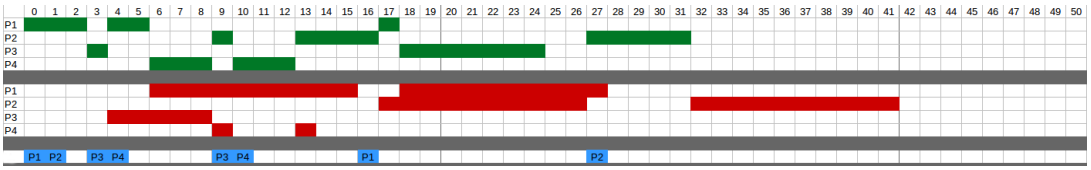

Figure 1: SJF Scheduler

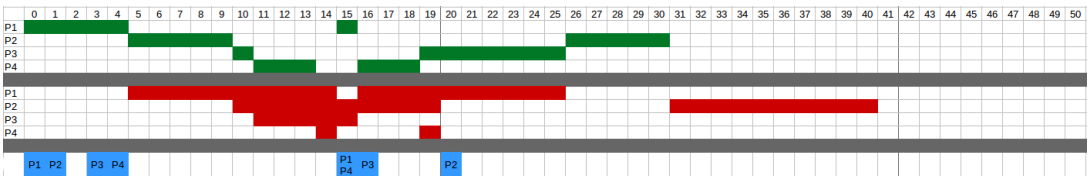

Figure 2: FIFO Scheduler

Per lo scheduler FIFO si assume che quando in un ciclo arrivano due processi contemporaneamente, allora venga servito quello con il `pid` minore - e.g.CPU-clock 15 della Figura 2.

Nell'allegato `so_exam_20180123_scheduling_01_solution.ods` è possibile trovare una versione espansa e più chiara delle Figure 1 e 2.

Esercizio 2

Sia data la seguente traccia di accesso alle pagine di memoria:

1 2 3 7 5 2 3 8 9 1 1 4 8 16 49.

Si assima di avere un TLB di 4 elementi, gestito con politica di rimozione LRU. Si assuma che un ciclo di fetch duri T_{fetch} ed un accesso al TLB impieghi T_{TLB}.

Domanda Calcolare il tempo di accesso medio alla memoria.

Soluzione Dalla traccia del problema avremo $N = 15$ totali accessi alle pagine di memoria. Data la politica di rimozione delle pagine, avremo il seguente tempo totale di accesso:

$$
\begin{aligned}
T_{TOT} = & \overbrace{4\,(2T_{Fetch} + T_{TLB})}^{\text{pages 1 2 3 7}} + \overbrace{(2T_{Fetch} + T_{TLB})}^{\text{page 5}} + \overbrace{(T_{Fetch} + T_{TLB})}^{\text{page 2}} + \overbrace{(T_{Fetch} + T_{TLB})}^{\text{page 3}} + \\
& + \overbrace{(2T_{Fetch} + T_{TLB})}^{\text{page 8}} + \overbrace{(2T_{Fetch} + T_{TLB})}^{\text{page 9}} + \overbrace{(2T_{Fetch} + T_{TLB})}^{\text{page 1}} + \\
& + \underbrace{(T_{Fetch} + T_{TLB})}_{\text{page 1}} + \underbrace{(2T_{Fetch} + T_{TLB})}_{\text{page 4}} + \underbrace{(T_{Fetch} + T_{TLB})}_{\text{page 8}} + \underbrace{(2T_{Fetch} + T_{TLB})}_{\text{page 16}} + \underbrace{(2T_{Fetch} + T_{TLB})}_{\text{page 49}}
\end{aligned}
$$

Quindi il tempo di accesso medio è dato da $T_{mean} = {}^{T_{TOT}}/_{N}$.

Esercizio 3

Cosa stampa il seguente programma?

```
#define N 3
int v[N];

int main () {
    for (int i=0; i<N; ++i)
        v[i]=i;

    for (int i=0; i<N; ++i) {
        int child_pid=fork();
        if (! child_pid) {
            for (int j=0; j<N; ++j) {
                v[j]=i;
            }
            break;
        }
        int retval;
        wait(&retval);
    }

    printf("[");
    for (int i=0; i<N; ++i)
    printf("%d ", v[i]);
    printf("]\n");
}
```

Soluzione L'output del programma è il seguente:

```
[0 0 0 ]
[1 1 1 ]
[2 2 2 ]
[0 1 2 ]
```

Esercizio 4

In uno schema produttore/consumatore abbiamo N processi produttori in grado di fornire dati ogni 10ms, ed M processi consumatori in grado di consumare dati a 17ms. I produttori/consumatori comunicano attraverso un buffer condiviso gestito come una coda circolare.

Domanda Qual è il rapporto ottimale produttori/consumatori tali che la coda non si riempia, ed i processi consumatori aspettino il minimo tempo possibile?

Soluzione Primariamente, indichiamo con $R_{prod} = {}^{N}/_{10}$ e $R_{cons} = {}^{M}/_{17}$ i rate di produzione e consumo di una risorsa. L'ottimo sarà dato dalla relazione:

$$R_{prod} = R_{cons} \rightarrow \frac{N}{10} = \frac{M}{17} \tag{1}$$

L'ottimo sar quindi $M = \frac{17}{10}\ N = 1.7\ N$ - o equivalentemente $N = \frac{10}{17}\ M$

Esercizio 5

Si consideri l'implementazione di un file system che gestisce i files mediante lista concatenata di blocchi (vedi progetto), ed un file system che invece utilizza un albero di indici memorizzati negli inode.

Domanda Illustrare brevemente i vantaggi dell'uno e dell'altro nell'eseguire le seguenti operazioni:

- accesso sequenziale
- accesso indicizzato
- operazioni su file di testo

Soluzione

1. **Accesso Sequenziale**: in questo caso, il file system che usa la *lista concatenata* sarà favorito, garantendo una maggiore velocità dell'operazione. Ciò poiché non bisogna effettuare alcuna ricerca per trovare il blocco successivo, poiché esso sarà semplicemente il blocco `next` nella lista.
2. **Accesso Indicizzato**: questa operazione - contrariamente alla precedente - risulta essere molto onerosa per il file system che usa la *linked list*. Infatti, per ogni accesso, bisognerà scorrere tutta la lista finchè non viene trovato il blocco desiderato. La ricerca tramite `inode` risulterà molto più efficiente.
3. **Accesso su file di testo**: per la natura del tipo di file (testo), la *linked list* risulterà più efficiente ancora una volta. Questo poiché i file di testo sono memorizzati in maniera sequenziale sul disco, riportandoci al caso 1.

Esercizio 6

Cos'è la legge di Amdahl? Cosa descrive tale legge?

Soluzione La legge di Amdahl permette di valutare il guadagno di performance derivante dal rendere disponibili più core computazionali ad una applicazione che ha componenti sia *seriali* che *parallele*. Indicando con $\mathcal{S}$ la porzione seriale dell'applicazione e con N il numero di core a disposizione, si avrà quindi la seguente relazione:

$$\texttt{GAIN} \leq \frac{1}{\mathcal{S} + \frac{1-\mathcal{S}}{N}} \tag{2}$$

E' bene notare che

$$\lim_{N \to \text{inf}} \texttt{GAIN} = \lim_{N \to \text{inf}} \frac{1}{\mathcal{S} + \frac{1-\mathcal{S}}{N}} = \frac{1}{\mathcal{S}} \tag{3}$$

Ovviamente il `GAIN` dipende anche da come è implementato nel dettaglio il sistema multi-core.

Esercizio 7

Si considerino i seguenti costrutti di sincronizzazione.

- spinlock
- mutex
- semafori

Domanda Quali sono le loro caratteristiche? Quando usare un mutex? Quando un semaforo? Quando uno spinlock?

Soluzione

- **Mutex**: i mutex sono usati per prevenire *race conditions* ed operare senza problemi durante sezioni critiche dei processi. I mutex mettono a disposizione un `lock` da acquisire prima di effettuare una operazione critica e da rilasciare una volta conclusa tale operazione. Il lock sostanzialmente una variabile binaria, manipolata tramite le operazioni atomiche di `acquire`/`release`.
- **Spinlock**: uno spinlock una particolare implementazione di un mutex tramite *busy-waiting* del processo - poiché per l'appunto il processo cicla a vuoto (spins) finché non acquisisce il lock. Tale implementazione risulta essere inefficiente per attese mediamente lunghe, dato che molti cicli di CPU vengono sprecati nel wait per il lock mentre potrebbero essere sfruttati da altri processi magari. Per operazioni molto brevi invece, gli *spinlock* risultano essere molto utili poiché non vi è context-switch durante la fase di wait.
- **Semafori**: un semaforo è costituito sostanzialmente da una variabile intera che può essere manipolata soltanto attraverso le operazioni di `wait` e `signal`; esse rispettivamente *decrementano* e *incrementano* il contatore all'interno del semaforo. Ovviamente, l'implementazione del semaforo è tale per cui solo una operazione per volta può essere effettuata.
  E' bene notare che un semaforo binario (solo 0 e 1 ammessi dal contatore) costituisce effettivamente un *mutex* e può essere utilizzato come tale se il sistema non fornisce quest'ultimi. Comunque, in generale, un semaforo può spaziare in un definito range di valori. Essi sono quindi molto utilizzati per regolare l'accesso a delle risorse (`wait` per richedere una risorsa e `signal` quando rilasciata/creata) o per sincronizzare processi.

Esercizio 8

Domanda In cosa consiste l'operazione di *mount* di un file system?

Soluzione Tramite tale operazione il sistema operativo viene informato che un nuovo file system pronto per essere usato. L'operazione, quindi, provvederà ad associarlo con un dato `mount-point`, ovvero la posizione all'interno della gerarchia del file system del sistema dove il nuovo file system verrà caricato. Prima di efferruare questa operazione di *attach*, ovviamente bisognerà controllare la tipologia e l'integrità del file system. Una volta fatto ciò, il nuovo file system sarà a disposizione del sistema (e dell'utente), come raffigurato nella Figura 3.

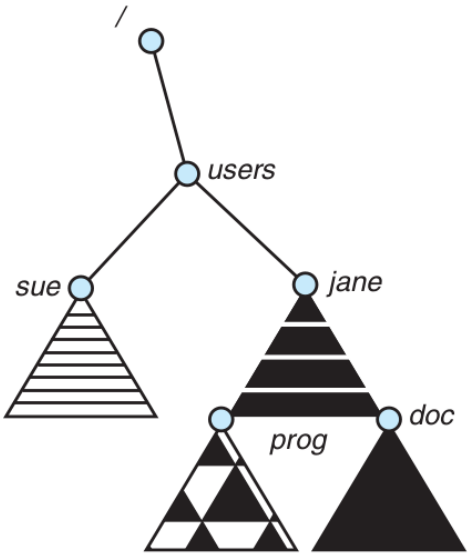

Figure 3: Il file system `jane` viene montato nella directory `/users/jane`

Esercizio 9

Domanda Quali sono i passi necessari per aggiungere una qualsiasi syscall in un sistema operativo provvisto di tale meccanismo?

Soluzione Una volta definita la syscall, va registrata nel sistema e aggiunta al vettore delle syscall del sistema operativo, specificando il numero di argomenti (ed il loro ordine) nell'altro vettore di sistema apposito.

Esercizio 10

Domanda Spiegare brevemente la differenza tra `open(...)` e `fopen(...)`.

Soluzione `fopen(...)` una funzione di alto livello che ritorna una stream, mentre `open(...)` una syscall di basso livello che ritorna un *file descriptor*. `fopen(...)`, infatti, nasconde una chiamata alla syscall `open(...)`.