Esame di Sistemi Operativi
AA 2017/18
11 Settembre 2018

Nome	Cognome	Matricola

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

	Arrivo	**Priorita'**	**CPU burst 1**	**IO burst 1**	**CPU burst 2**	**IO burst 2**
P1	0	2	3	2	3	2
P2	1	1	2	4	2	4
P3	3	4	7	1	7	1
P4	2	3	5	1	5	1

Domanda Si assuma di disporre di uno scheduler preemptive con politica di selezione dei processi basata sulla *priorita'* di un processo - *Priority Scheduling*. Si assuma che:

- l'operazione di avvio di un processo lo porti nella coda di ready, ma **non** necessariamente in esecuzione
- il termine di un I/O porti il processo che termina nella coda di ready, ma **non** in esecuzione.

Si illustri il comportamento dello scheduler in questione nel periodo indicato, avvalendosi degli schemi di seguito riportati (vedi pagina seguente).

Soluzione Date queste premesse, la traccia di esecuzione dei processi e riportata nella Figura 1

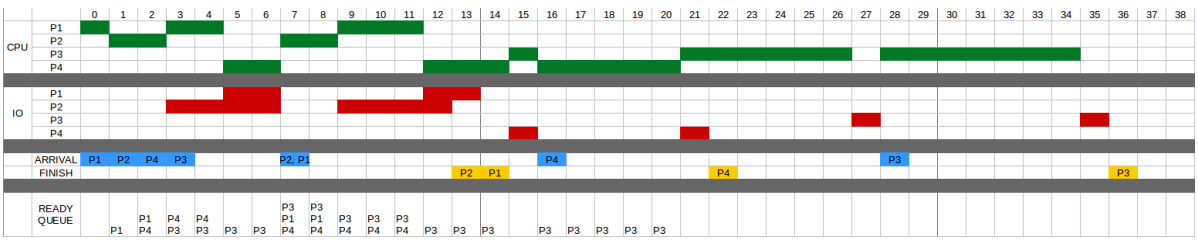


Figure 1: Traccia di esecuzione dei processi con *Priority Scheduling*. In verde sono indicati i cicli di `CPU burst` ed in rosso quelli di `I/O`. L'arrivo ed il termine di un processo sono indicati reispettivamente in blu e giallo.

Nome	Cognome	Matricola

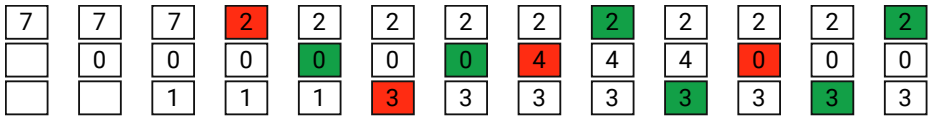
Esercizio 2

Sia data la seguente traccia di accesso alle pagine di memoria:
7 - 0 - 1 - 2 - 0 - 3 - 0 - 4 - 2 - 3 - 0 - 3 - 2.

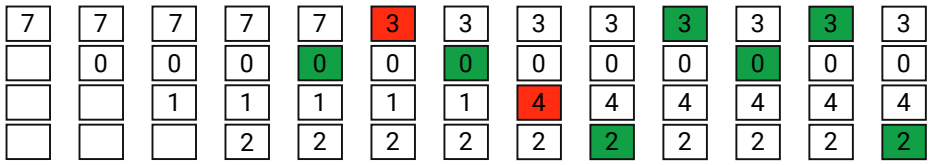
Si assima di avere un TLB di 3 elementi, gestito con politica *Optimal replacement*. Si assuma che un ciclo di fetch duri T_{fetch} ed un accesso al TLB impieghi T_{TLB}.

Domanda Di quanto aumentano le prestazioni con un TLB a 4 elementi?

Soluzione Le tracce di accesso sono riportate rispettivamente in Figura 2a e Figura 2b.



(a) Traccia con TLB di 3 elementi.



(b) Traccia con TLB di 4 elementi.

Figure 2: Tracce d'accesso con TLB a 3 e 4 elementi. In rosso sono evidenziate le sostituzioni, in verde gli *hit*.

Nel caso di TLB a 3 elementi il numero di hit e' pari a $N_{hit} = 6$, con una probabilita' di page fault $p_f = 0.54$; passando a 4 elementi si avra' $N_{hit} = 7$ e $p_f = 0.46$. Il guadagno prestazionale e' pari circa al 8%.

Nome	Cognome	Matricola

Esercizio 3

Quali risorse vengono usate per creare un thread? In cosa differiscono da quelle usate per la creazione di un processo?

Soluzione Creare un thread - sia esso kernel o user - richiede l'allocazione di una data structure contenente il register set, lo stack e altre informazioni quali la priorita', come riportato in Figura 3.

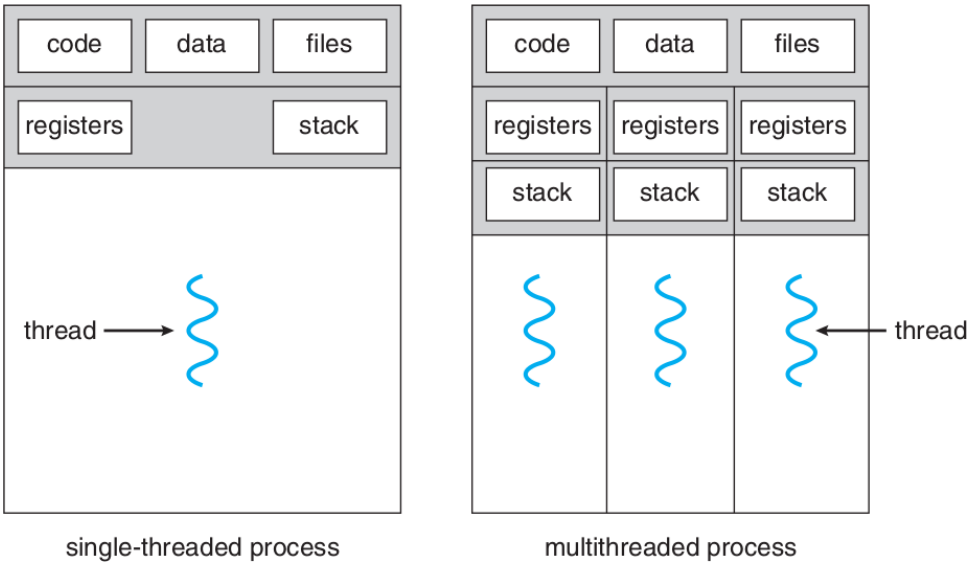


Figure 3: Processo single thread vs. multi-thread.

Creare un nuovo processo invece, richiede l'allocazione di un nuovo PCB (una data structure molto piu' pesante). Il PCB contiene tutte le informazioni del processo, quali `pid`, stato del processo, informazioni sull'`I/O`, il Program Counter e la lista delle risorse aperte dal processo. Inoltre il PCB include anche informazioni sulla memoria allocata dal processo in una speciale *memory-map*; allocare quest'ultima e' un'operazione molto time-consuming.

In definitiva, quindi, la creazione di un thread e' molto piu' semplice rispetto a quella di un nuovo processo.

Nome	Cognome	Matricola

Esercizio 4

Si consideri un file consistente in 100 blocchi e che il suo *File Control Block* (FCB) sia gia' in memoria. Siano dati due file-systems gestiti rispettivamente tramite allocazione a lista concatenata (*Linked List Allocation*) e allocazione indicizzata (*Indexed Allocation*)

Domanda Si calcolino le operazioni di I/O su disco necessarie per eseguire le seguenti azioni in entrambi i file-systems:

1. Aggiunta di un blocco all'inizio del file
2. Aggiunta di un blocco a meta' del file
3. Aggiunta di un blocco alla fine del file

Soluzione Nel caso di file-system con *Indexed Allocation*, ogni file conterra un index block, ovvero un blocco contenente i puntatori a tutti gli altri blocchi componenti il file. Cio' implica che per raggiungere un dato blocco bastera' effettuare una semplice indicizzazione. Nell'implementazione con *Linked List Allocation* invece, ogni blocco contiene il riferimento al precedente ed al successivo. In questo caso, per effettuare operazioni che non siano all'inizio del file bisognera' scorrere la lista fino al blocco desiderato ed in seguito eseguire l'operazione necessaria.

Date queste premesse, i risultati sono:

1. Linked Allocation: 1 `IO-ops`; Indexed Allocation: 1 `IO-ops`
2. Linked Allocation: 52 `IO-ops`; Indexed Allocation: 1 `IO-ops`
3. Linked Allocation: 3 `IO-ops`; Indexed Allocation: 1 `IO-ops`

Nome	Cognome	Matricola

Esercizio 5

Sia dato un sottosistema di memoria con paginazione, caratterizzato dalle seguenti dimensioni:

- frame 4 `KB`
- memoria fisica indirizzabile 512 `MB`

Domande

1. Quanti bit sono necessari per individuare una pagina in un indirizzo virtuale?
2. A quanto corrisponde la probabilita' di *page-fault* considerando un sistema in cui: un accesso al TLB impiega 2 `ns`; un ciclo di lettura/scrittura richiede 200 `ns`; il tempo di accesso medio e' di 240 `ns`.

Soluzione

1. Poiche' 4 `KB` richiedono 12 bit e 512 `MB` necessitano di 29 bit per l'indirizzamento, avremo che il numero di bit per individuare una pagina e' $29 - 12 = 17$ bit.
2. La formula per il calcolo del tempo di accesso medio alla memoria - *Effective Access Time* - e' data dalla seguente relazione:

$$T_{EAT} = p_{hit}(T_{TLB} + T_{RAM}) + p_{fault} \cdot 2(T_{TLB} + T_{RAM}) \tag{1}$$

   Sostituendo i dati della traccia nella (1) e risolvendo in $p_{fault} = 1 - p_{hit}$ avremo $p_{fault} = 0.1881$, ovvero $p_{hit} = 0.8119$.

Nome	Cognome	Matricola

Esercizio 6

Cosa stampa il seguente programma?

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int value;

void* runner(void* param);

int main(int argc, char *argv[]) {
  value = 10;

  pid_t pid;
  pthread_t tid;
  pthread_attr_t attr;
  pid = fork();

  if (pid < 0)
    return -1;

  if (pid == 0) {
    pthread_attr_init(&attr);
    pthread_create(&tid,&attr,runner,NULL);
    pthread_join(tid,NULL);
    printf("line C, value = %d\n",value); /* LINE C */
  } else {
    wait(NULL);
    printf("line P, value = %d\n",value); /* LINE P */
  }
  return 0;
}

void* runner(void* param) {
  value = 5;
  exit(0);
}
```

Soluzione Il thread `runner` dopo aver cambiato il valore di `value`, effettua brutalmente una `exit`, che portera' alla morte del processo `child`. Pertanto, il programma stampera' soltanto

```
line P, value = 10
```

Nome	Cognome	Matricola

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.

Nome	Cognome	Matricola

Esercizio 8

Che cos'e' una *syscall*? Come e' possibile aggiungere (in linea di massima) una nuova *syscall* in un OS previsto di tale meccanismo?

Soluzione Una *syscall* e' una chiamata diretta al sistema operativo da parte di un processo user-level - e.g. quando viene invocata la funzione `printf`.

Nel caso si volesse aggiungere una nuova syscall, essa, una volta definita, va registrata nel sistema e aggiunta al vettore delle syscall del sistema operativo, specificandone il numero di argomenti (ed il loro preciso ordine).

Nome	Cognome	Matricola

Esercizio 9

Sia data la seguente tabella di processi/risorse.

Allocated:

Processo	R1	R2	R3
P0	0	0	0
P1	2	0	0
P2	1	0	2

Maximum:

Processo	R1	R2	R3
P0	2	5	3
P1	2	0	1
P2	3	0	10

Available:

R1	R2	R3
0	5	8

Domande Si illustri l'evoluzione dell'algoritmo del banchiere nella gestione dei deadlock.

Soluzione In base alle specifiche, l'evoluzione del algoritmo e' la seguente:

```
*****************************************
op: 0
next_op: 1
p: 0
work:
SIZE: 3
0 5 8
needed:
SIZE: 3
2 5 3
reject

*****************************************
op: 0
next_op: 2
p: 1
work:
SIZE: 3
0 5 8
needed:
SIZE: 3
0 0 1
accept

*****************************************
op: 1
next_op: 2
p: 0
work:
SIZE: 3
2 5 8
```

```
needed:
SIZE: 3
2 5 3
accept

****************************************
op: 2
next_op: 3
p: 2
work:
SIZE: 3
2 5 8
needed:
SIZE: 3
2 0 8
accept

is_safe: 1
final_order:
SIZE: 3
1 0 2
```

Nome	Cognome	Matricola

Esercizio 10

Cos'e' la directory `/proc` e cosa contiene al suo interno?

Soluzione La directory `proc` un *virtual filesystem*, poich non contiene file reali, bens runtime system information - e.g. memoria di sistema, configurazione hardware, etc. Molte delle utilities di sistema, infatti, si riferiscono ai file contenuti in questa cartella - e.g. `lsmod` $\equiv$ `cat /proc/modules`.