Esame di Sistemi Operativi
AA 2018/19
26 Marzo 2019
[soluzione]

Nome	Cognome	Matricola

Esercizio 1

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

Process	T_{start}	**CPU Burst**	**IO Burst**
P1	0	2	7
P2	2	5	2
P3	2	3	6
P4	4	6	1

Domanda Si assuma di disporre di uno *scheduler preemptive* Round Robin (RR) con quanto di tempo pari a $T_q = 10$. Si assuma inoltre che:

- i processi in entrata alla CPU dichiarino il numero di burst necessari al proprio completamento;
- 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 necessariamente** in esecuzione.

Si illustri il comportamento dello scheduler in questione nel periodo indicato, avvalendosi degli schemi di seguito riportati (vedi pagina seguente). Si supponga che i processi **si ripresentino con le stesse specifiche** una volta finito l'I/O.

Soluzione In base alle specifiche di cui sopra e supponendo che quando arrivino due processi contemporaneamente allo stesso istante di tempo venga scelto quello con `pid` minore, la traccia di esecuzione sara' quella riportata in Figura 1. Si noti che nessuno dei processi supera il quanto di tempo, quindi lo scheduler si comportera' come un First Come First Served (FCFS).

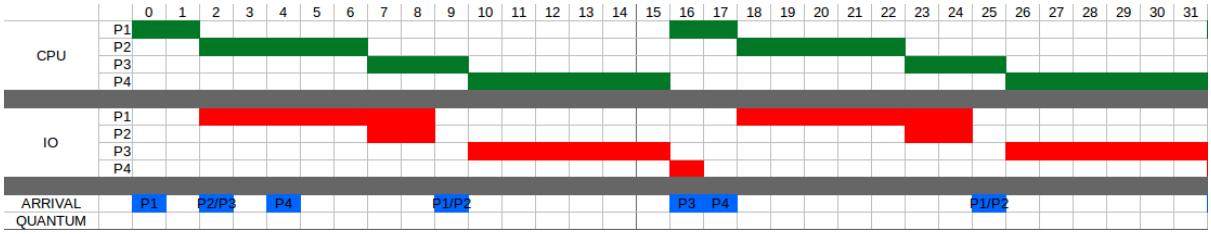


Figure 1: Traccia di esecuzione dei processi.

Nome	Cognome	Matricola

Esercizio 2

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

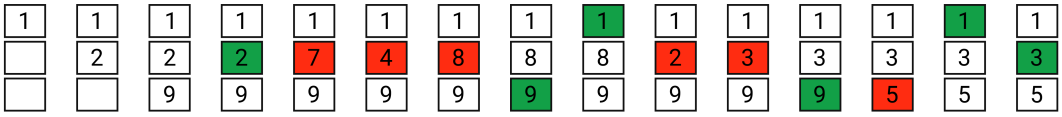
1 2 9 2 7 4 8 9 1 2 3 9 5 1 3

Si assima di avere un Translation Lookaside Buffer (TLB) di 3 elementi, gestito con politica *Optimal Replacement*. Si assuma che T_{fetch} e T_{TLB} siano rispettivamente i tempi di fetch e di accesso al TLB. Di quanto aumentano le prestazioni incrementando la dimensione del TLB a 4 elementi?

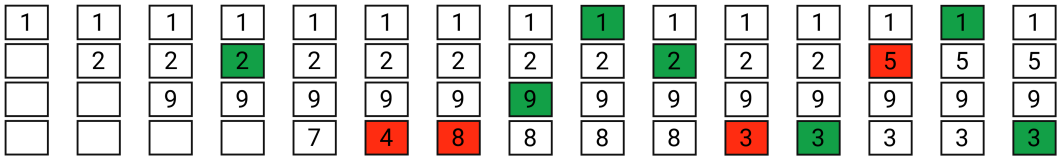
Soluzione Le tracce di accesso per entrambe le versioni del TLB sono riportate rispettivamente in Figura 2a e Figura 2b. Ponendo $T_{miss} = 2(T_fetch + T_{TLB})$ e $T_{hit} = T_fetch + T_{TLB}$, avremo:

$$T_{EAT[3]} = 9T_{miss} + 6T_{hit} \quad (1)$$
$$T_{EAT[4]} = 8T_{miss} + 7T_{hit} \quad (2)$$

La probabilita' di page fault passera' da $p_{f[3]} = 0.60$ a $p_{f[4]} = 0.53$, risultando in un miglioramento delle prestazioni del 7%.



(a) TLB con 3 entries



(b) TLB con 4 entries

Figure 2: Traccia delle pagine nel TLB in entrambi i casi.

Nome	Cognome	Matricola

Esercizio 3

Con riferimento agli algoritmi di *Page Replacement*, enumerare i 4 principali algoritmi usati per tale scopo, ordinandoli in base al loro *page-fault rate* (dal piu' alto al piu' basso). Si evidenzino anche gli algoritmi che soffrono dell'anomalia di Belady.

Soluzione Partendo dall'algoritmo con le peggiori performances in termini di *page-fault rate*, avremo:

#	**Algorithm**	**Belady**
1.	FIFO	si'
2.	Second Chance	si'
3.	LRU	no
4.	Optimal	no

Nome	Cognome	Matricola

Esercizio 4

Sia dato un sistema con paginazione con frame di dimensione 50 e TLB a 3 entries, gestito con politica di rimozione della pagine Last Recently Used (LRU). Si ipotizzi che un piccolo processo sia completamente contenuto nella prima pagina del sistema - i.e. da locazione 0 a 49. Si consideri quindi un array bidimensionale `float A[][]` con dimensioni R × C con R = 1 e C = 50, allocato come:

```
1
2     float** A = (float**)malloc(sizeof(float*)*R);
3     for (int r = 0; r < R; ++r)
4       A[r] = (float*)malloc(sizeof(float)*C);
5
```

Supponendo che le ultime pagine utilizzate in fase di allocazione siano gia' nel TLB, calcolare il numero di *page fault* dovute all'inizializzazione dell'array nelle seguenti modalita':

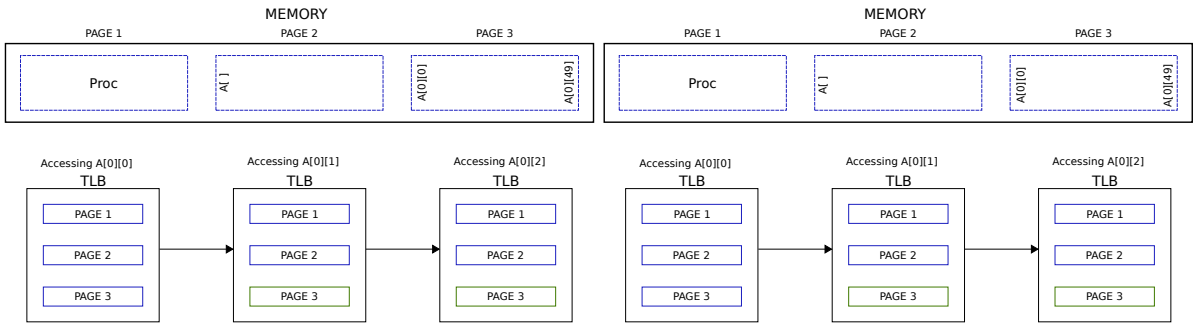
Modalita' 1:

```
1
2   for (int r = 0; r < R; ++r)
3     for (int c = 0; c < C; ++c)
4       A[r][c] = 0;
5
```

Modalita' 2:

```
1
2   for (int c = 0; c < C; ++c)
3     for (int r = 0; r < R; ++r)
4       A[r][c] = 0;
5
```

Soluzione In base alle specifiche del sottosistema di memoria e a come avviene l'allocazione della matrice `A[][]` il numero di pagine utilizzate in totale e' 3. Esse entrano tutte nel TLB, percio' in entrambe le modalita' il numero di page fault e' pari a 0 - poiche' quando la matrice viene allocata, le pagine vengano caricate nel TLB. Inoltre, per le specifiche dimensioni dell'array, le due modalita' di accesso si equivalgono. In Figura 3 e' riportata una breve illustrazione del sottosistema di memoria in entrambi i casi.



(a) Modalita' 1 (b) Modalita' 2

Figure 3: Illustrazione della memoria e del TLB in entrambe le esecuzioni.

Nome	Cognome	Matricola

Esercizio 5

Cos'e' un File Control Block (FCB)? Quali sono le informazioni contenute al suo interno?

Soluzione Il FCB e' una struttura dati che contiene tutte le informazioni relative al file a cui e' associato. Esempi di informazioni possono essere: permessi, dimensione, data di creazione, numero di inode (se esiste), ecc. . Inoltre il FCB contiene informazioni sulla locazione sul disco dei dati del file - ad esempio in un File System (FS) con allocazione concatenata il puntatore al primo blocco del file. In Figura 4 e' riportata una illustrazione di tale struttura.

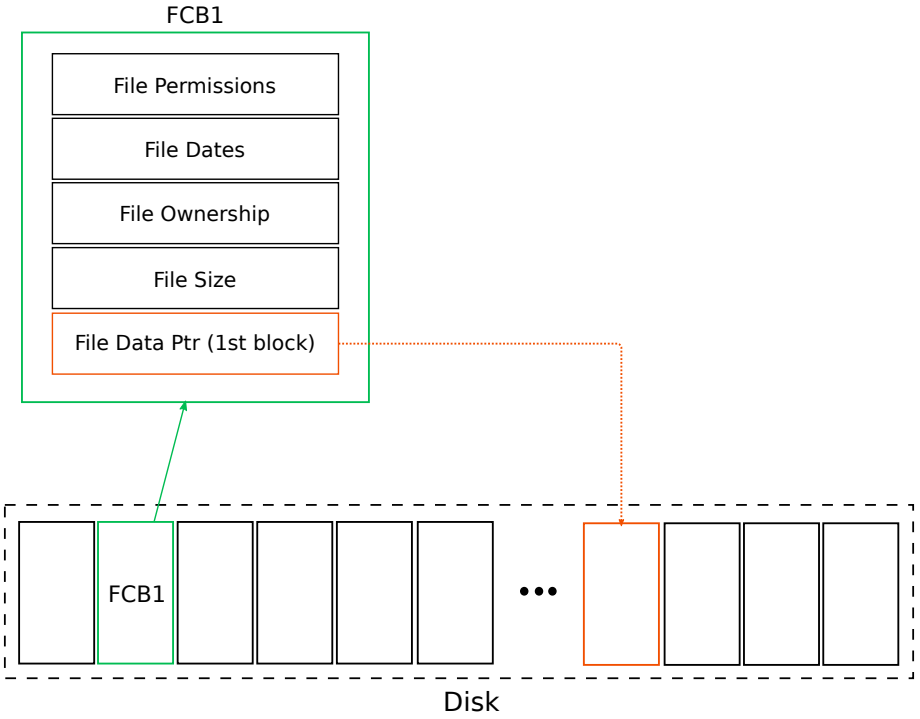


Figure 4: Esempio di FCB. La struttura contiene tutti gli attributi del file compresa la locazione dei dati - qui rappresentato dal puntatore al primo blocco della lista contente i dati, supponendo un FS con Linked List Allocation (LLA).

Nome	Cognome	Matricola

Esercizio 6

Siano dati i seguenti programmi:

P1:

```
#define SHMEM_SIZE 1
#define NUM_ROUNDS 10

char* sem_name_0;
char* sem_name_1;
char* resource_name;

int main(int argc, char** argv) {
  sem_name_0 = "s0";
  sem_name_1 = "s1";
  resource_name = "object";

  sem_t* sem_0 = sem_open(sem_name_0, O_CREAT, 0666, 0); //! full
  sem_t* sem_1 = sem_open(sem_name_1, O_CREAT, 0666, 1); //! empty

  int fd=shm_open(resource_name, O_RDWR|O_CREAT, 0666);
  if (fd < 0){
    exit(-1);
  }
  int ftruncate_result = ftruncate(fd, SHMEM_SIZE);
  if (ftruncate_result < 0) {
    exit(-1);
  }

  void * my_memory_area = mmap(NULL, SHMEM_SIZE, PROT_WRITE, MAP_SHARED,
  fd, 0);
  for (int i=0; i < NUM_ROUNDS; ++i) {
    sem_wait(sem_1);
    char* buffer=(char*) my_memory_area;
    sprintf(buffer, "%d", i);
    printf("p1|%s\n", buffer);
    sem_post(sem_0);
    sleep(1);
  }

  printf("p1|exit\n");

  int unlink_result=shm_unlink(resource_name);
  if (unlink_result<0) {
    exit(-1);
  }

  sem_close(sem_0);
  sem_close(sem_1);

  sem_unlink(sem_name_0);
  sem_unlink(sem_name_1);

  return 0;
}

```

P2:

```
int main(int argc, char** argv) {
  char* sem_name_0 = "s0";
  char* sem_name_1 = "s1";
  char* resource_name = "object";

  sem_t* sem_0 = sem_open(sem_name_0, O_CREAT, 0666, 0); //! full
  sem_t* sem_1 = sem_open(sem_name_1, O_CREAT, 0666, 1); //! empty

  int fd = shm_open(resource_name, O_RDONLY, 0666);
  if (fd < 0){
    exit(-1);
  }

  int SHMEM_SIZE = 0;

  struct stat shm_status;
  int fstat_result = fstat(fd, &shm_status);
  if (fstat_result < 0){
    exit(-1);
  }

  SHMEM_SIZE = shm_status.st_size;

  void* my_memory_area = mmap(NULL, SHMEM_SIZE, PROT_READ, MAP_SHARED, fd,
  0);

  while (1) {
    sem_wait(sem_0);
    int n = atoi((char*) my_memory_area);
    printf("p2|%d\n", n);
    sem_post(sem_1);
    if (4 == n)
      break;
  }

  printf("p2|exit\n");

  int unlink_result = shm_unlink(resource_name);
  if (unlink_result < 0) {
    exit(-1);
  }

  sem_close(sem_0);
  sem_close(sem_1);

  return 0;
}

```

Domanda Spigare brevemente cosa fanno i programmi. Inoltre, cosa stampano P1 e P2?

Soluzione I due programmi seguono uno schema "produttore/consumatore" (rispettivamente P1 e P2), condividendo dati attraverso una *Shared Memory*. Quest'ultima viene creata e configurata solo da P1, quindi se P2 viene eseguito prima di P1 esso andra' in errore. L'accesso alla Shared Memory e' salvaguardato da 2 semafori - condivisi tra i processi.

L'output dei due programmi e' riportato di seguito.

P1:

```
p1|0
p1|1
p1|2
p1|3
p1|4
p1|5

```

P2

```
    p2|0
    p2|1
    p2|2
    p2|3
    p2|4
    p2|exit

```

Si noti come, una volta che P2 abbia terminato la sua esecuzione, P1 rimanga bloccato a causa del semaforo.

Nome	Cognome	Matricola

Esercizio 7

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

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

Nome	Cognome	Matricola

Esercizio 8

Che relazione c'e' tra una syscall, un generico interrupt e una *trap*? Sono la stessa cosa?

Soluzione Ovviamente le tre cose non sono la stessa cosa.

Una syscall e una chiamata diretta al sistema operativo da parte di un processo user-level - e.g. quando viene fatta una richiesta di IO.

Un interrupt invece e' un segnale asincrono proveniente da hardware o software per richiedere il processo immediato di un evento. Gli interrupt software sono definiti *trap*. A differenza delle syscall, gli interrupt esistono anche in sistemi *privi di Sistema Operativo* - e.g. in un microcontrollore. Quando una syscall viene chiamata, una trap verra' generata (un interrupt software), in modo da poter richiamare l'opportuna funzione associata a tale syscall (attraverso la Syscall Table (ST)).