Esame di Sistemi Operativi
AA 2018/19
14 Febbraio 2020
[soluzione]

Nome	Cognome	Matricola

Esercizio 1

Sia data la seguente tabella che descrive il comportamento di un insieme di processi periodici **real-time**.

processo	**tempo di inizio**	**deadline**	**periodo**	**CPU burst**
P1	0	4	6	1
P2	0	8	12	4
P3	0	12	14	3

Domanda Si assuma di disporre di uno scheduler preemptive Earliest Deadline First (EDF). Si assuma inoltre che:

- nessuno dei processi debba attendere il rilascio di una risorsa posseduta da un altro processo;
- 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.

Si illustri il comportamento dello scheduler in questione nel periodo indicato, avvalendosi degli schemi di seguito riportati.

Soluzione Poiche' i processi devono essere eseguiti in real-time, per prima cosa dobbiamo verificare se e' possibile rispettare le deadline con una sola CPU. Calcoliamo quindi la percentuale di utilizzo della CPU come

$$\mathrm{U}_{CPU} = \sum_{p} \frac{t_p}{d_p} = \frac{1}{4} + \frac{4}{8} + \frac{3}{12} = 1 \tag{1}$$

Dalla Equazione 1 si nota che $\mathrm{U}_{CPU} \leq 1$, quindi e' possibile effettuare lo scheduling rispettando tutte le deadline. In base alle specifiche richieste, la traccia di esecuzione dei processi sara' quella riportata in Figura 1.

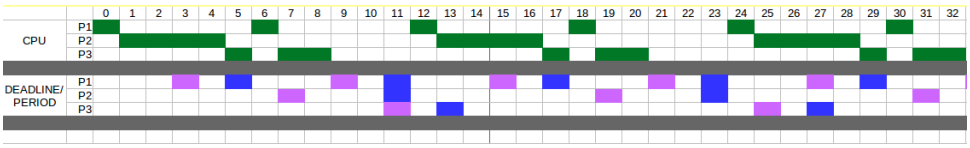


Figure 1: Traccia di esecuzione dei processi. Per ogni processo, in viola sono riportate le deadline e in blu il periodo.

Nome	Cognome	Matricola

Esercizio 2

Si consideri in sottosistema di memoria il caratterizzato dalle seguenti tabelle

Number	Base	Limit
0x00	0x000	0x002
0x01	0x002	0x006
0x02	0x008	0x001
0x03	0x009	0x003
0x04	0x00C	0x001
0x05	0x00E	0x001

Table 1: Segmenti

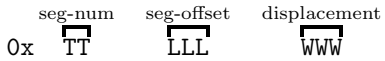
Page	Frame
0x000	0x010
0x001	0x00F
0x002	0x00E
0x003	0x00D
0x004	0x00C
0x005	0x00B
0x006	0x00A
0x007	0x009
0x008	0x008
0x009	0x007
0x00A	0x006
0x00B	0x005
0x00C	0x004
0x00D	0x003
0x00E	0x002
0x00F	0x001
0x010	0x000

Table 2: Pagine

Domanda Assumendo che le pagine abbiano una dimensione di 512 byte, che la tabella delle pagine consista di 512 elementi e che la tabella dei segmenti possa contenere 256 elementi, come vengono tradotti in indirizzi fisici i seguenti indirizzi logici?

- `0x05001FA1`
- `0x01007001`
- `0x00000000`
- `0x01005AAC`
- `0x03F02000`

Soluzione In base alle specifiche del sistema, ogni indirizzo virtuale si potra' scomporre come segue:



I primi due digit piu' significativi individuano il `segment number`. Useremo quindi `base + seg-offset` per individuare il frame all'interno della tabella delle pagine. Ovviamente, se il `seg-offset` eccede il `limit` di quel segmento, si avra' un errore. L'indirizzo finale sara' dato da `frame + displacement`.

Sulla base di quanto detto, avremo i seguenti indirizzi fisici:

- ⊙ `0x05001FA1` → `0x001FA1`
- ⊙ `0x01007001` → invalido
- ⊙ `0x00000000` → `0x010000`
- ⊙ `0x01005AAC` → `0x009AAC`
- ⊙ `0x03F02000` → invalido

Nome	Cognome	Matricola

Esercizio 3

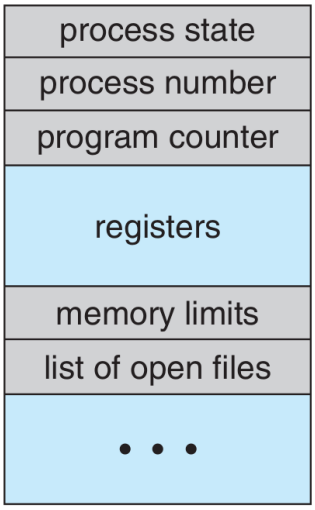
Cosa contiene il Process Control Block (PCB) di un processo? Illustrare inoltre il meccanismo di *Context Switch* (avvalendosi anche di schemi approssimativi).

Soluzione Il PCB di un processo contiene tutte le informazioni relative al processo a cui e' associato. Esempi di informazioni contenute nel PCB sono:

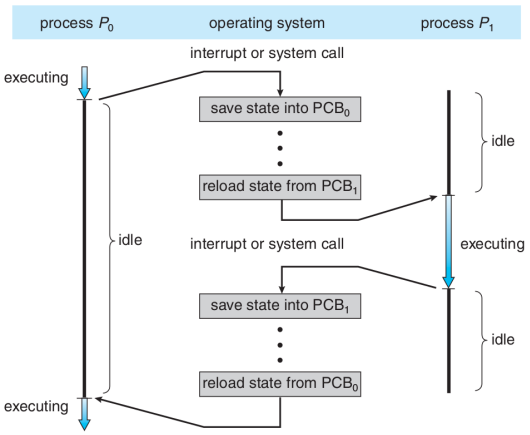
- stato del processo (running, waiting, zombie ...)
- Program Counter (PC), ovvero il registro contente la prossima istruzione da eseguire
- registri della CPU
- informazioni sulla memoria allocata al processo
- informazioni sull'I/O relativo al processo.

Una illustrazione di tale struttura dati e' riportata in Figura 2a.

Supponendo di avere due processi `P0` e `P1` e che il primo sia in running. Per mettere in esecuzione `P1`, l'Operating System (OS) deve salvare lo stato corrente di `P0` in modo da poter rirpistinare l'esecuzione dello stesso in un secondo momento. Quindi, il *context switch* può essere riassunto visivamente nella Figura 2b. E' bene notare che il *context switch* e' fonte di overhead a causa delle varie operazioni di preambolo e postambolo necessarie allo switch - e.g. salvare lo stato, blocco e riattivazione della pipeline di calcolo, svuotamento e ripopolamento della cache.



(a) Una rappresentazione grafica del PCB. Esso contiene tutte le informazioni relative alla gestione di un processo da parte dell'OS.



(b) Esempio di *context switch*.

Figure 2: Processi: PCB e *context switch*.

Nome	Cognome	Matricola

Esercizio 4

A cosa serve la syscall `ioctl`? Come si puo configurare la comunicazione con *devices a caratteri e seriali* in Linux?

Soluzione La syscall `ioctl` permette di interagire con il driver di un device generico - e.g. una webcam. Tramite essa sara' possibile ricavare e settare i parametri di tale device - e.g. ricavare la risoluzione della webcam o settarne la tipologia di acquisizione dati.

Per configurare devices seriali a caratteri - e.g. terminali - e' possibile usare le API racchiuse nella interfaccia `termios`. Tramite di essa, avremo accesso a tutte le informazioni relative al device - e.g. baudrate, echo, ...

Nome	Cognome	Matricola

Esercizio 5

Sia dato il seguente programma:

```
1
2   #define STACK_SIZE 16384
3   #define ITERATIONS 4
4   ucontext_t main_context, f1_context, f2_context;
5
6   char f1_stack[STACK_SIZE];
7   char f2_stack[STACK_SIZE];
8   unsigned int cnt = 0;
9
10  void f1(void) {
11    printf("f1: start\n");
12
13    swapcontext(&f1_context, &f2_context);
14
15    if (cnt < ITERATIONS) {
16      printf("f1: %d\n", cnt++);
17      setcontext(&f1_context);
18    }
19
20    printf("f1: end\n");
21    setcontext(&main_context);
22  }
23
24  void f2(void) {
25    printf("f2: start\n");
26
27    getcontext(&f2_context);
28
29    if (cnt < ITERATIONS/2) {
30      printf("f2: %d\n", cnt++);
31      setcontext(&f2_context);
32    }
33
34    printf("f2: end\n");
35    swapcontext(&f2_context, &f1_context);
36  }
37
38  int main() {
39    printf("- program start -\n");
40
41    getcontext(&f1_context);
42
43    f1_context.uc_stack.ss_sp=f1_stack;
44    f1_context.uc_stack.ss_size = STACK_SIZE;
45    f1_context.uc_stack.ss_flags = 0;
46    f1_context.uc_link=&main_context;
47    makecontext(&f1_context, f1, 0, 0);
48
49    f2_context=f1_context;
50    f2_context.uc_stack.ss_sp=f2_stack;
51    f2_context.uc_stack.ss_size = STACK_SIZE;
52    f2_context.uc_stack.ss_flags = 0;
53    f2_context.uc_link=&main_context;
54    makecontext(&f2_context, f2, 0, 0);
55
56    swapcontext(&main_context, &f1_context);
57
58    printf("- program end -\n");
59  }
60
```

Indicare quale dei seguenti puo essere un possibile output del programma e motivare la risposta:

- Output A:

```
1
2       - program start -
3       f1: start
4       f2: start
5       f2: 0
6       f2: 1
7       f2: end
8       f1: 2
9       f1: 3
10      f1: end
11      - program end -
12
```

- Output B:

```
1
2       - program start -
3       f1: start
4       f1: 0
5       f1: 1
6       f1: 2
7       f1: 3
8       f1: end
9       f2: start
10      f2: 0
11      f2: 1
12      f2: end
13      - program end -
14
```

- Output C:

```
1
2       - program start -
3       f1: start
4       f1: 0
5       f1: 1
6       f1: end
7       f2: start
8       f2: 2
9       f2: 3
10      f2: end
11      - program end -
12
```

- Output D:

```
1
2       - program start -
3       f1: start
4       f1: 0
5       f1: end
6       f2: start
7       f2: 1
8       f2: end
9       - program end -
10
```

- Output E: Nessuna delle precedenti (specificare quindi l'output di seguito).

Soluzione L'Output A rispecchia cio' che il programma stampa realmente. Nello specifico, una volta inizializzati i contesti (nella funzione `main`), viene eseguito `f1_context`. La `swapcontext` alla linea 13, portera' in esecuzione `f2`, aggiornando anche `f1_context`. Qui, la `getcontext` in linea 27, aggiornera'

il contesto. A causa di cio', la `setcontext` in riga 31 fara' ripartire la funzione dalla linea 28. Una volta finite le iterazioni (`ITERATIONS/2`), la `swapcontext` in linea 35 fara' ripartire `f1_context` - dalla linea 14. Di nuovo la `setcontext` in linea 17 ci riportera' alla linea 14. Una volta finite le iterazioni, `f1` riportera' in esecuzione il main. Concludendo, il programma esegue due loop senza l'utilizzo di istruzioni `for` o `while`.

Nome	Cognome	Matricola

Esercizio 6

Spiegare brevemente cose il Direct Memory Access (DMA) e quando viene usato.

Soluzione Alcuni device necessitano di trasferire grandi quantita' di dati a frequenze elevate. Effettuare tali trasferimenti tramite il protocollo genericamente usato per altri tipi di periferiche richiederebbe l'intervento della CPU per trasferire un byte alla volta i dati - tramite un processo chiamato Programmed I/O (PIO). Cio' risulterebbe in un overhead ingestibile per l'intera macchina, consumando inutilmente la CPU. Per consentire il corretto funzionamento di tali device evitando gli svantaggi del PIO, tali periferiche possono avvalersi di controllori dedicati che effettuano DMA, cioe' andando a scrivere direttamente sul bus di memoria. La CPU sara' incaricata soltanto di "validare" tale trasferimento e poi sara' di nuovo libera di eseguire altri task.

Questo tipo di periferiche sono molto comuni ai giorni nostri e sono usate nella maggior parte dei dispositivi elettronici - pc, smartphones, servers, ... Esempi di periferica che si avvalgono di controller DMA sono videocamere, dischi, schede video, schede audio, ecc.

Nome	Cognome	Matricola

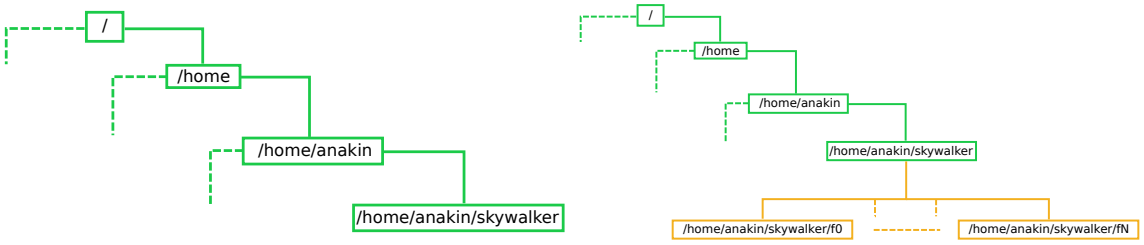
Esercizio 7

Sia dato una macchina con un generico OS installato. Si supponga che il disco primario `/dev/sda0` sia gestito da un File System (FS) `ext4` e che il suo *mount-point* sia `/`. Si supponga di collegare fisicamente alla macchina due dischi secondari - `/dev/sda1` e `/dev/sda2` - rispettivamente formattati in `FAT32` e `NTFS`.

Si schematizzi l'albero delle directory prima e dopo l'esecuzione del comando di cui sotto, spiegando brevemente come viene gestita dall'OS tale richiesta:

```
$ sudo mount /dev/sda2 /home/anakin/skywalker
```

Soluzione Tramite il comando `mount` l'OS viene informato che un nuovo FS e' pronto per essere usato. L'operazione, quindi, provvedera' ad associarlo con un dato `mount-point`, ovvero la posizione all'interno della gerarchia del FS del sistema dove il nuovo FS verra' caricato. Prima di effettuare questa operazione di *attach*, ovviamente bisognera' controllare la tipologia e l'integrita' del FS. Una volta fatto cio', il nuovo FS sara' a disposizione del sistema (e dell'utente). In Figura 3 viene schematizzato tale processo. L'omogeneita' dell'interfaccia offerta all'utente e' garantita dal layer di Virtual File System (VFS) dell'OS. Tramite il VFS, dischi con FS diversi possono coesistere nello stesso sistema ed essere esplorati senza complicazioni da parte dell'utente.



(a) Overview prima dell'operazione di `mount`.

(b) Overview dopo l'operazione di `mount`.

Figure 3: Schematizzazione dei FS del sistema prima e dopo l'operazione di `mount`. La directory `/` rappresenta il `mount-point` del disco principale `/dev/sda0`, mentre il disco secondario `/dev/sda2` avra' come `root` la directory `/home/anakin/skywalker`. Gli elementi di quest'ultimo sono visualizzati in giallo nella Figura 3b.

Si noti che il disco `/dev/sda1` non e' visibile all'utente. Cio' poiche' il disco e' fisicamente attaccato alla macchina ma *non* e' stata effettuata alcuna operazione di `mount` per esso.

Nome	Cognome	Matricola

Esercizio 8

Che relazione ce 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)).