Esame di Sistemi Operativi
AA 2018/19
22 Gennaio 2019

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.

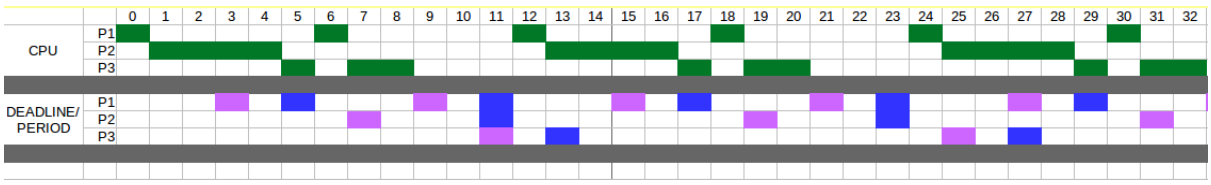


Figure 1: Traccia di esecuzione dei processi. Per ogni processo, in viola sono riportate le deadline e in blu il periodo.

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

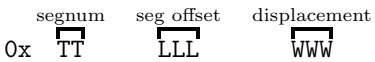
Page	Frame
0x000	0x010
0x001	0x00F
0x002	0x00E
0x003	0x00D
0x004	0x00C
0x005	0x00B
0x006	0x00A
0x007	0x009
0x008	0x008
0x009	0x007
0x00A	0x006
0x00B	0x005
0x00C	0x004
0x00D	0x003
0x00E	0x002
0x00F	0x001
0x010	0x000

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?

- `0x020000A1`
- `0x01004FAC`
- `0x01005FFF`
- `0x01FFFAAA`

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 + offset` per individuare il frame all'interno della tabella delle pagine. Ovviamente, se l'`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:

- ⊙ `0x020000A1` → `0x0080A1`
- ⊙ `0x01004FAC` → `0x00AFAC`
- ⊙ `0x01005FFF` → `0x009FFF`
- ⊙ `0x01FFFAAA` → invalido

Nome	Cognome	Matricola

Esercizio 3

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 (LLA) - e allocazione contigua - Contiguous Allocation (CA). Si assuma che nel caso di CA, eventuale spazio per estendere il file sia disponibile solo alla fine dello stesso - non all'inizio.

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

- Rimozione di un blocco all'inizio del file
- Rimozione di un blocco a meta' del file
- Rimozione di un blocco alla fine del file

Soluzione Nel caso di LLA per effettuare una operazione bisognera' scorrere la lista fino al blocco in questione. Nell'implementazione con CA i blocchi sono allocati in maniera sequenziale sul disco, quindi ogni volta bisognera' ricopiare tutti i blocchi in modo da "compattarli". Date queste premesse, i risultati sono i seguenti:

1. LLA: 1 `IO-ops`; CA: 198 `IO-ops`
2. LLA: 52 `IO-ops`; CA: 98 `IO-ops`
3. LLA: 100 `IO-ops`; CA: 0 `IO-ops`

Si noti che, nell'implementazione con CA la rimozione di un file in posizione n richiede $(2n-2)$ `IO-ops`. Cio' poiche' per spostare un blocco e' necessario prima leggerlo (1 `IO-ops`) e poi scriverlo nella posizione giusta (1 `IO-ops`).

Nome	Cognome	Matricola

Esercizio 4

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

Soluzione

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.

Nome	Cognome	Matricola

Esercizio 5

Cos'e' un Virtual File System (VFS)? Fornisci un esempio del loro utilizzo.

Soluzione Con VFS viene indicato tutto il livello di astrazione costruito su un File System concreto. Il VFS rende possibile ad un client di accedere a diversi tipi di File System in maniera uniforme e consistente. Un esempio di VFS e' la directory `/proc` in Linux. I file contenuti in detta directory sono in realta' *runtime information* del sistema - e.g. memoria di sistema, configurazione hardware - le quali vengono rese accessibili come un file tramite l'astrazione del VFS.

Nome	Cognome	Matricola

Esercizio 6

Sia dato il seguente programma:

```
#define STACK_SIZE 16384
#define ITERATIONS 5

ucontext_t main_context, f1_context, f2_context;

void f1(){
  for (int i=0; i<ITERATIONS; i++) {
    printf("f1: %d\n", i);
    swapcontext(&f1_context, &f2_context);
  }
  setcontext(&main_context);
}

void f2(){
  for (int i=0; i<ITERATIONS; i++) {
    printf("f2: %d\n", i);
    exit(0);
  }
  setcontext(&main_context);
}

char f1_stack[STACK_SIZE];
char f2_stack[STACK_SIZE];

int main() {
  getcontext(&f1_context);

  f1_context.uc_stack.ss_sp=f1_stack;
  f1_context.uc_stack.ss_size = STACK_SIZE;
  f1_context.uc_stack.ss_flags = 0;
  f1_context.uc_link=&main_context;
  makecontext(&f1_context, f1, 0, 0);


  f2_context=f1_context;
  f2_context.uc_stack.ss_sp=f2_stack;
  f2_context.uc_stack.ss_size = STACK_SIZE;
  f2_context.uc_stack.ss_flags = 0;
  f2_context.uc_link=&main_context;
  makecontext(&f2_context, f2, 0, 0);

  swapcontext(&main_context, &f1_context);
  printf("exiting\n");
}

```

Domanda Cosa stampa il programma?

Soluzione Poiche' in `f2` e' presente una `exit`, il programma si interrompera'. L'output della shell e' il seguente:

```
f1: 0
f2: 0
```

Nome	Cognome	Matricola

Esercizio 7

Cos'e' una *mailbox* nell'ambito dell'Inter-Process Communication (IPC)? Fornisci un breve esempio del suo utilizzo.

Soluzione Tra i metodi di IPC troviamo lo scambio di messaggi tra piu' processi tramite *mailbox*. I vantaggi derivanti dall'uso di una mailbox sono diversi:

- maggiore flessibilita' - e.g. e' possibile implementare una comunicazione remota tra processi eseguiti su macchine diverse
- semplicita' di implementazione
- sicurezza nella comunicazione - e.g. nessuna lettura parziale dei dati poiche' processati messaggio per messaggio

Il costo da pagare, invece, sara' in termini di performances:

- maggiore overhead poiche' la comunicazione dovra' essere gestita dal sistema operativo
- copia dei dati nella coda
- l'accesso ai dati e' in generale piu' lento perche' limitato ad un messaggio per volta (la dimensione di un messaggio e' generalmente limitata e relativamente piccola).

La comunicazione e' basata sulle operazioni di `send` e `receive`. Tali funzioni (syscall) possono essere sviluppati in base alle specifiche esigenze dell'utente (comunicazione sincrona/asincrona, diretta/indiretta, limitata/illimitata ...). I processi quindi potranno creare una coda di messaggi oppure linkarsi ad una mailbox esistente a partire da un identificatore della coda. Le altre operazioni fondamentali sono:

- ◇ check dello status della coda
- ◇ post di un messaggio
- ◇ attesa di un messaggio (bloccante o non bloccante).

Nome	Cognome	Matricola

Esercizio 8

Cos'e' la Syscall Table (ST)? Come e' possibile installare una nuova syscall in un Sistema Operativo che supporta tale meccanismo? Infine, in cosa differiscono la ST e l'Interrupt Vector (IV)?

Soluzione 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.

L' IV e un vettore di puntatori a funzioni; queste ultime saranno le Interrupt Service Routine (ISR) che gestiranno i vari interrupt. Analogamente, la ST conterra in ogni locazione il puntatore a funzione che gestisce quella determinata syscall. Alla tabella verra associata anche un vettore contenente il numero e lordine di parametri che detta syscall richiede.

Per registrare una nuova syscall, essa va registrata nel sistema ed aggiunta al vettore delle syscall del sistema operativo, specificando il numero di argomenti (ed il loro ordine) nell'altro vettore di sistema apposito.