Esame Sistemi Operativi - Operating Systems Exam
2023/02/17

Ex 1 (2.0 points)

Italiano

Si supponga che il disco rigido di un piccolo sistema embedded sia costituito da 25 blocchi di 1 MByte, che tali blocchi siano numerati da 0 a 24, che il sistema operativo mantenga traccia dei blocchi liberi (occupati) indicandoli in un vettore con il valore 0 (1), e che la situazione attuale del disco sia rappresentata dal seguente vettore:

`0 1 1 0 0 0 1 0 0 1 1 1 1 1 0 0 1 0 1 0 0 0 1 0 0`

Si indichino quali delle seguenti affermazioni sono vere. Si osservi che risposte errate implicano una penalità nel punteggio finale.

English

Suppose that the hard disk of a small embedded system is composed of 25 blocks of 1 MByte each, which are numbered from 0 to 24. Suppose that the operating system keeps track of the free (occupied) blocks indicating them in a vector with the value 0 (1), and that the current situation of the disk is represented by the following vector:

`0 1 1 0 0 0 1 0 0 1 1 1 1 1 0 0 1 0 1 0 0 0 1 0 0`

Indicate which of the following statements are correct. Note that wrong answers imply a penalty in the final score.

Scegli una o più alternative: Choose one or more options:

1. ⬜ Un file di dimensione 2.5 MByte NON può essere allocato utilizzato una strategia di allocazione contigua. A file with dimension 2.5 MByte CANNOT be allocated using a contiguous allocation strategy.
2. ✅ Con la strategia di allocazione contigua ottenuta mediante l'algoritmo BEST-FIT possono essere allocati nell'ordine i file F1 di 1.2 MByte, F2 di 2.9 MByte e F3 di 2.8 MByte. With the contiguous allocation strategy based on the BEST-FIT algorithm, the following files can be allocated in the following order: F1 of 1.2 MByte, F2 of 2.9 MByte, and F3 of 2.8 MByte.
3. ⬜ Con la strategia di allocazione contigua ottenuta mediante l'algoritmo FIRST-FIT possono essere allocati nell'ordine i file F1 di 1.2 MByte, F2 di 2.9 MByte e F3 di 2.8 MByte. With the contiguous allocation strategy based on the FIRST-FIT algorithm, the following files can be allocated in the following order: F1 of 1.2 MByte, F2 of 2.9 MByte, and F3 of 2.8 MByte.
4. ⬜ Un file di dimensione 13.8 MByte può essere allocato utilizzando una strategia di allocazione indicizzata. A file with dimension 13.8 MByte can be allocated using an indexed allocation strategy.
5. ✅ La strategia di allocazione contigua soffre sia di frammentazione esterna che di frammentazione interna. The contiguous allocation strategy suffers for both external and internal fragmentation.

Ex 2 (4.5 points)

Italiano

Un numero imprecisato di thread può invocare una delle seguenti due funzioni:

1. `int br_rec(void)` che blocca il thread che la invoca finché non viene eseguita la funzione `br_send`.
2. `void br_send(int x)` che sblocca **tutti** i thread bloccati sino quel momento dalla funzione `br_rec()` e in più fa in modo che la funzione `br_rec()` restituisca (con la sua return) il valore `x` a tutti i thread che vengono risvegliati.

Esempio di funzionamento:

- Il thread A chiama `br_rec()` e viene posto in attesa.
- Il thread B chiama `br_rec()` e viene posto in attesa.
- Il thread D chiama `br_rec()` e viene posto in attesa.
- Il thread C chiama `br_send(5)` che sblocca i thread A, B e D, e fa in modo che le rispettive funzioni `br_rec()` restituiscono il valore 5 ai tre thread.

Si realizzino le funzioni `br_send()` e `br_rec()` utilizzando i semafori POSIX e si definiscano le relative variabili condivise. Per semplicità, si ipotizzino tutti i thread siano di tipo aciclico (cioè che le funzioni siano usate una sola volta dai thread compresi nel set) e che la funzione `br_send()` sia chiamata una sola volta e dopo che siano state chiamate tutte le funzioni di tipo `br_rec()`.

English

An unknown number of threads can invoke one of the following two functions:

1. `int br_rec(void)` that blocks the calling thread until the function `br_send` is executed.
2. `void br_send(int x)` that unlocks **all** threads blocked until that moment by the function `br_rec()`, and allows the function `br_rec()` to return (with its return statement) the value `x` to all unlocked threads.

Example of operation:

- Thread A calls `br_rec()` and is blocked.
- Thread B calls `br_rec()` and is blocked.
- Thread D calls `br_rec()` and is blocked.
- Thread C calls `br_send(5)` which unlocks threads A, B, and D, and causes their functions `br_rec()` to return a value of 5 to the three threads.

Implement the functions `br_send()` and `br_rec()` using POSIX semaphores and define all shared variables. For the sake of simplicity, assume that all threads are acyclic (i.e., the functions are used only once by the threads in the set) and that the function `br_send()` is called only once and after all calls to the function `br_rec()`.

Risposta: Answer:

```
int v;
sem_t m;
sem_t sync;

int main() {
  sem_init(&m, 0, 1);
  sem_init(&sync, 0, 0);

  return 0;
}

int br_rec() {
  sem_wait(&m);
  n++;
  sem_post(&m);
  sem_wait(&sync);
  return v;
}

void br_send(int x) {
  int i;
  sem_wait(&m);
  v = x;
  for(i=0;i<n;i++)
    sem_post(&sync);
  n = 0; /* Not strictly needed because the code is used only one time */
  sem_post(&m);
}
```

Ex 3 (3.0 points)

Italiano

Lo stato di un sistema con 4 processi e 4 differenti risorse è il seguente:

- Il processo P1 detiene le risorse {R2, R3} ed è in attesa della risorsa {R1}
- Il processo P2 detiene la risorsa {R1} ed è in attesa della risorsa {R2}
- Il processo P3 detiene la risorsa {R4} ed è in attesa della risorsa {R3}
- Il processo P4 detiene la risorsa {R2} ed è in attesa della risorsa {R4}

Dire quali sono le possibilità di eliminare il deadlock rimuovendo un solo arco?

English

The state of a system with 4 processes and 4 different resources is the following:

- Process P1 holds resources {R2, R3} and waits resource {R1}
- Process P2 holds resource {R1} and waits resource {R2}
- Process P3 holds resource {R4} and waits resource {R3}
- Process P4 holds resource {R2} and waits resource {R4}

Report which are the possible ways to eliminate deadlock by removing only one edge?

Scegli una o più alternative: Choose one or more options:

1. ⬜ P2→R1
2. ⬜ R2→P1
3. ✅ P1→R1
4. ⬜ R4→P3
5. ✅ R1→P2
6. ⬜ R2→P4
7. ✅ P2→R2

Ex 4 (3.0 points)

Italiano

Si indichi l'output o gli output possibili che possono essere ottenuti eseguendo concorrentemente i processi PA, PB e PC riportati di seguito.

English

Indicate the possible output or outputs that we can obtain by concurrently executing the following processes PA, PB, and PC.

```
init (S1, 1); init(S2, 0); init(S3, 0);
```

PA

```
wait(S1);
printf("5");
signal(S3);
signal(S2);
wait(S1);
printf("6");
```

PB

```
wait(S3);
wait(S3);
printf("3");
signal(S3);
signal(S1);
```

PC

```
printf("4");
wait(S2);
printf("7");
signal(S3);
wait(S3);
printf("2");
```

Scegli una o più alternative: Choose one or more options:

1. ✅ 547326
2. ⬜ 546273
3. ✅ 547362
4. ✅ 547236 (Dovrebbe essere non corretta)
5. ⬜ 453276
6. ✅ 457362
7. ⬜ 456732
8. ⬜ 457263
9. ✅ 457326
10. ✅ 457236 (Dovrebbe essere non corretta)

Ex 5 (2.0 points)

Italiano

Data la seguente riga di comando bash:

```
egrep -e "^B" file.txt | egrep -e  "[02468]$" | egrep -v dollar| cut -d ":" -f 1-3
```

Riportare quali righe sono stampate in output dal precedente comando, quando esso è eseguito sul file input.txt riportato alla fine della domanda.

English

Given the following bash command:

```
egrep -e "^B" file.txt | egrep -e  "[02468]$" | egrep -v dollar| cut -d ":" -f 1-3
```

Report which lines are printed in output from the previous command when it is executed on the file input.txt, which is reported at the end of the question.

Contenuto del file input.txt Content of the file input.txt

```
B:Bread:5.40:euro:23
S:Pasta:7.40:euro:12
B:Rice:3.30:dollar:34
B:Spinach:4.50:euro:38
S:Strawberry:3.40:euro:40
B:Kiwi:2.50:euro:10
B:Cheese:12.5:euro:12
```

Scegli una o più alternative: Choose one or more options:

1. ⬜ `B:3.30`
2. ✅ `B:Spinach:4.50`
3. ⬜ `B:Rice:3.30`
4. ⬜ `B:2.50`
5. ⬜ `B:Cheese:12.5`
6. ✅ `B:Kiwi:2.50`
7. ✅ `B:Cheese:12.5`
8. ⬜ `B:4.50`
9. ⬜ `S:7.40`

Ex 6 (2.0 points)

Italiano

Quali dei seguenti algoritmi di schedulazione NON è dotato di prelazione.

English

Which of the following scheduling algorithms has NOT preemption.

Scegli una o più alternative. Si osservi che risposte errate implicano una penalità nel punteggio finale.
Choose one or more options. Note that incorrect answers imply a penalty in the final score.

1. ✅ Priority Scheduling (PS)
2. ✅ Shortest Job First (SJF)
3. ⬜ Shortest Remaining Time First (SRTF)
4. ✅ First Come First Served (FCFS)
5. ⬜ Round Robin (RR)

Ex 7 (2.0 points)

Italiano

Si indichino quali delle seguenti affermazioni relative ai processi e ai thread sono corrette. Si osservi che risposte errate implicano una penalità nel punteggio finale.

English

Indicate which of the following statements referred to processes and threads are correct. Note that incorrect answers imply a penalty in the final score.

Scegli una o più alternative: Choose one or more options:

1. ✅ Dopo una `fork()`, l'address space del processo figlio e un duplicato di quello del processo padre. After a fork(), the address space of the child process is a duplicate of the one of the parent.
2. ✅ Le operazioni di scrittura di una variabile globale da parte di un thread sono visibili da tutti gli altri thread. Write operations on global variables performed by a thread can be seen from all other threads.
3. ⬜ Si supponga la variabile x sia definita come globale prima della creazione di due processi P1 e P2. Se P1 e P2 modificano contemporaneamente x, si può avere una race condition? Suppose that x is defined as a global variable before creating processes P1 and P2. If P1 and P2 modify x at the same time, may a race condition happen?
4. ⬜ Se due thread modificano contemporaneamente la stessa variabile si può avere deadlock. If two threads modify the same variable at the same time a deadlock may occur.
5. ⬜ TUTTI gli schemi di sincronizzazione che si possono realizzare mediante le funzioni `sem_wait()` e `sem_post()` per i thread, possono essere realizzati utilizzando le funzioni `fork()` e `wait()` per i processi. ALL the synchronization schemes that can be implemented using the `sem_wait()` and `sem_post()` functions for threads can be implemented using `fork()` and `wait()` for processes.
6. ✅ Dopo una `fork()` padre e figlio condividono la stessa working directory. After a `fork()` the father and the child processes share the same working directory.
7. ✅ Si supponga la variabile x sia definita come globale e venga effettuata una `fork()`. Dopo tale `fork()` padre e figlio condividono lo stesso valore di x? Suppose that x is defined as a global variable and we execute a `fork()`. After the `fork()`, do the father and the child processes share the same value of x?
8. ⬜ Con il termine copy-on-write si intende che la scrittura effettiva su un file avviene solo quando il sistema operativo chiama la system call `write()`. With the term copy-on-write we mean that the actual write operations on a file take place only when the operating system calls the `write()` system call.

Ex 8 (2.5 points)

Italiano

Se esegui il seguente programma, quanti caratteri 'X' sono visualizzati su standard output? Indicare un unico intero nella risposta

English

If you run the following program, how many characters 'X' are displayed on standard output? Report a single integer value in your response.

```
int main () {
	int i;
	int pid;
	for (i=1; i<3; i++) {
		pid = fork ();
		i++;
		if (pid!=0) {
			fork();
			printf("X");
		}
	}
	printf("X");
	return 1;
```

```
}
```

Risposta: Answer:
5

Ex 9 (2.0 points)

Italiano

Si faccia riferimento alla system call `kill()` e al relativo comando di shell `kill`. Si indichino quali delle seguenti affermazioni sono vere. Si osservi che risposte errate implicano una penalità nel punteggio finale.

English

Refer to the system call `kill()` and the related shell command `kill`. Indicate which of the following statements are correct. Note that wrong answers imply a penalty in the final score.

Scegli una o più alternative: Choose one or more options:

1. ✅ La ricezione di un segnale può portare alla terminazione del processo che lo ha ricevuto. Receiving a signal may lead to the termination of the receiving process.
2. ⬜ Il comando `kill -9` indirizzato ad un processo può essere ignorato dal processo ricevente. The command `kill -9` addressed to a process can be ignored by the receiving process.
3. ⬜ Il comando di shell `kill` può essere utilizzato per uccidere un processo, la system call `kill()` no. The shell command `kill` can be used to terminate a process, the `kill()` system call cannot.
4. ⬜ La system call `signal()` può inviare un segnale generico, invece la system call `kill()` viene usata per inviare uno specifico segnale per uccidere il processo ricevente. The `signal()` system call can be used to send a generic signal, whereas the `kill()` system call is used to send a specific signal that terminates the receiving process.
5. ✅ Un signal handler è una porzione di codice che viene eseguita alla ricezione di uno specifico segnale. A signal handler is a fragment of code that is executed whenever a specific signal is received.
6. ⬜ Un processo può spedire un segnale a se stesso mediante la system call `signal()`. A process can send a signal to itself using the `signal()` system call.

Ex 10 (3.0 points)

Italiano

Si consideri il seguente insieme di processi schedulati con un quantum temporale di 10 unità di tempo. Rappresentare mediante diagramma di Gantt l'esecuzione di tali processi utilizzando l'algoritmo Shortest Remaining Time First (SRTF), al fine di calcolare il tempo di terminazione di ciascun processo e il tempo di attesa medio. Si prega di riportare la risposta su un'unica riga, indicando i tempi di terminazione di P1, P2, P3, P4 e P5 seguiti dal tempo di attesa medio. Separare i numeri con un unico spazio. Riportare il tempo di attesa medio con 1 sola cifra decimale. Non inserire nessun altro carattere nella risposta. Esempio di risposta corretta: 20 23 11 45 67 30.5

English

Consider the following set of processes, which are scheduled with a temporal quantum of 10 units. Represent using a Gantt diagram the execution of these processes using the Shortest Remaining Time First (SRTF) scheduling algorithm, in order to compute the termination time of each process, and the average waiting time. Please, write your answer on a single line, indicating the termination times of P1, P2, P3, P4 and P5 followed by the average waiting time. Separate the numbers with a single space. Report the average waiting time with a single decimal digit. Do not enter any other character in the response. Example of correct answer: 20 23 11 45 67 30.5

Processo Process	TempoArrivo ArrivalTime	BurstTime	Priorità Priority
P1	0	25	1
P2	5	19	5

P3	9	11	4
P4	14	13	2
P5	23	5	3

Risposta: Answer:
73 53 20 38 28 17.6

Ex 11 (5.0 points)

Italiano

Si implementi uno script BASH che accetti due parametri da linea di comando:

- Il primo parametro può essere la stringa `start` oppure la stringa `stop`.
- Il secondo parametro corrisponde al percorso di un file che contiene, uno per riga, una serie di percorsi di programmi. Si assuma che i programmi indicati nel file siano presenti nel sistema. Il seguente è un esempio corretto di tale file:
  ```
  /usr/local/bin/pgrm1
  /bin/pgrm2
  /usr/bin/pgrm2
  ```

Dopo aver controllato il corretto passaggio di parametri, lo script deve eseguire le seguenti operazioni:

- Quando lo script riceve la stringa `start` come primo parametro, deve eseguire in background tutti i programmi indicati dal file passato come secondo parametro.
- Quando lo script riceve la stringa `stop` come primo parametro, deve terminare l'esecuzione di tutti i programmi indicati dal file passato come secondo parametro e deve stampare il PID di tutti i processi terminati in tale modo.

Si assuma che non esistano istanze dei programmi indicati in esecuzione al di fuori di quelle eseguite mediante lo script stesso con il parametro `start`. Si ricorda che il comando `ps -aux` permette di individuare i PID dei processi in esecuzione, visualizzando un output simile a quello riportato di seguito.

English

Implement a BASH script that takes two command line parameters:

- The first parameter is either the string `start` or `stop`.
- The second parameter is the path of a file that contains a sequence of paths of programs, one for each line. Assume that each of these programs exists in the filesystem. The following is a correct example of such a file:
  ```
  /usr/local/bin/pgrm1
  /bin/pgrm2
  /usr/bin/pgrm2
  ```

The script should check the correctness of the parameters and then perform the following operations:

- When the script receives the string `start` as its first parameter, it should execute each program reported in the file (second parameter) in background.
- When the script receives the string `stop` as its first parameter, it should terminate the execution of each program reported in the file (second parameter) and then print the PID of each process terminated this way.

Assume that for each of the programs reported in the file, there are no further running instances besides those that have been executed by invoking the script itself with the `start` parameter. Recall that command `ps -aux` can be used to check the PIDs of the running processes and that it displays an output similar to the following one.

```
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root        1  0.0  0.0 168372 13628 ?        Ss   gen21   0:06 /sbin/init splash
root        2  0.0  0.0      0     0 ?        S    gen21   0:00 [kthreadd]
root        3  0.0  0.0      0     0 ?        I<   gen21   0:00 [rcu_gp]
root        4  0.0  0.0      0     0 ?        I<   gen21   0:00 [rcu_par_gp]
root        5  0.0  0.0      0     0 ?        I<   gen21   0:00 [slub_flushwq]
root        6  0.0  0.0      0     0 ?        I<   gen21   0:00 [netns]
...
```

Risposta: Answer:

```
#!/bin/bash

################################################################################
# Version 1: while-read loops to read the input file line-by-line, grep
# command to find the correct pid
################################################################################

# Check if the correct parameters has been passed
if [ $# -lt 2 ]; then
  echo "Usage: ./batch-exec.sh <command> <list>"
  exit 1
fi

# Check if the second parameter is a valid filename
if [ ! -f $2 ]; then
      echo "The second parameter should be the path to a valid file"
      exit 1
fi

# Start/stop list of programs
if [ $1 == "start" ]; then
      while read cmd; do
      "$cmd" &
      done < $2
elif [ $1 == "stop" ]; then
      while read cmd; do
      pid=$(ps -aux | grep -e "$cmd" | grep -v grep | tr -s " " | cut -d " " -f 2)
      kill -9 $pid
      echo "Terminated process $pid running $cmd"
      done < $2
else
      echo "The first parameter should be either the command 'start' or 'stop'"
      exit 1
fi

#!/bin/bash

################################################################################
# Version 2: for loops with modified IFS to read the input file line-by-line,
# comparisons to find the correct pid
################################################################################

# Check if the correct parameters has been passed
if [ $# -lt 2 ]; then
  echo "Usage: ./batch-exec.sh <command> <list>"
  exit 1
fi

# Check if the second parameter is a valid filename
if [ ! -f $2 ]; then
      echo "The second parameter should be the path to a valid file"
      exit 1
fi

# Start/stop list of programs
```

```
IFS="
"
if [ $1 == "start" ]; then
      for cmd in $(cat $2); do
      "$cmd" &
      done
elif [ $1 == "stop" ]; then
      for cmd in $(cat $2); do
      ps -aux | tr -s " " | tail -n +2 | cut -d " " -f 2,11 > tmpfile
      for line in $(cat tmpfile); do
            pid=$(echo $line | cut -d " " -f 1)
            command=$(echo $line | cut -d " " -f 2)
            if [ "$command" == "$cmd" ]; then
            kill -9 $pid
            echo "Terminated process $pid running $cmd"
            fi
      done
      rm tmpfile
      done
else
      echo "The first parameter should be either the command 'start' or 'stop'"
      exit 1
fi
```

Ex 12 (5.0 points)

Italiano

Si implementi un programma in linguaggio C in grado di eseguire tre processi P1, P2 e P3 in modo che siano connessi tra di loro da tre pipe, tra P1-P2, P2-P3 e P3-P1. I tre processi lavorano in maniera circolare, ovvero ciascun processo Pi (ovvero P1, P2, oppure P3) legge da standard input un numero intero e lo trasmette al processo successivo (ovvero P2, P3, oppure P1, rispettivamente) che dorme quel numero di secondi e ripete la stessa operazione (ovvero legge un numero intero e lo trasmette al processo seguente nella disposizione circolare dei processi). Il primo processo a effettuare l'operazione di lettura da standard input è P1. Per semplicità si supponga che i processi non abbiano termine, ovvero non si gestisca alcuna condizione di terminazione. Si riporti l'implementazione del main e dei processi P1, P2 e P3.

English

Implement a program in C language that runs three processes P1, P2, and P3 that are connected to each other by three pipes, P1-P2, P2-P3, and P3-P1. The three processes work in a circular manner, i.e., each process Pi (i.e., P1, P2, or P3) reads an integer from standard input and transmits it to the next process (i.e., P2, P3, or P1, respectively) which sleeps that number of seconds and repeats the same operation (i.e., reads an integer and transmits it to the next process in the circular arrangement of processes). The first process to perform the read operation from standard input is P1. For the sake of simplicity, assume that processes do not terminate, i.e., no termination conditions must be handled. Please report the implementation of the main and processes P1, P2, and P3.

Risposta: Answer:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

static void p1(int, int);
static void p2(int, int);
static void p3(int, int);
```

```
int main () {
  int p12[2], p23[2], p31[2];

  setbuf (stdout, 0);

  // Missing checks for pipes
  pipe (p12);
  pipe (p23);
  pipe (p31);

  if (fork()!=0) {
    close (p12[0]); close (p23[0]); close (p23[1]); close (p31[1]);
    p1 (p31[0], p12[1]);
  } else {
    if (fork()!=0) {
      close (p12[1]); close (p23[0]); close (p31[0]); close (p31[1]);
      p2 (p12[0], p23[1]);
    } else {
      close (p12[0]); close (p12[1]); close (p23[1]); close (p31[0]);
      p3 (p23[0], p31[1]);
    }
  }

   return (0);
}

static void p1 (int pr, int pw) {
  int n;
  n = 0;
  while (1) {
    fprintf (stdout, "P1 waiting: %d secs\n", n);
    sleep (n);
    fprintf (stdout, "P1 reading: ");
    scanf ("%d", &n);
    write (pw, &n, sizeof (int));
    read (pr, &n, sizeof (int));
  }
  return;
}

static void p2 (int pr, int pw) {
  int n;
  while (1) {
    read (pr, &n, sizeof (int));
    fprintf (stdout, "P2 waiting: %d secs\n", n);
    sleep (n);
    fprintf (stdout, "P2 reading: ");
    scanf ("%d", &n);
    write (pw, &n, sizeof (int));
  }
  return;
}

static void p3 (int pr, int pw) {
  int n;
  n = 0;
```

```
  while (1) {
    read (pr, &n, sizeof (int));
    fprintf (stdout, "P3 waiting: %d secs\n", n);
    sleep (n);
    fprintf (stdout, "P3 reading: ");
    scanf ("%d", &n);
    write (pw, &n, sizeof (int));
  }
  return;
}
```