Esame Sistemi Operativi - Operating Systems Exam
2022/09/06

Ex 1 (2.0 points)

Italiano

Relativamente alle system call **open()**, **read()**, **write()** e **close()**, si indichino quali delle seguenti affermazioni sono corrette. Si osservi che risposte errate implicano una penalità nel punteggio finale.

English

Regarding the system calls **open()**, **read()**, **write()** and **close()**, indicate which of the following statements are correct. Note that incorrect answers imply a penalty in the final score.

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

1. ✅ La system call **write()** può essere utilizzata per scrivere byte indifferentemente su file, su schermo o su pipe. The **write()** system call can be used to write bytes either on a file, console or pipe without distinction.
2. ⬜ La system call **open()** accetta esattamente due parametri. The **open()** system call takes exactly two parameters.
3. ✅ Le system call **read()** e **write()** possono essere utilizzate rispettivamente per leggere e scrivere testo (caratteri) o sequenze di byte. The **read()** and **write()** system calls can be used respectively to read or write text (characters) or byte sequences.
4. ⬜ La system call **open()** accetta esattamente tre parametri. The **open()** system call takes exactly three parameters.
5. ⬜ La system call **read()** ritorna il numero di elementi letti invece del numero di bytes letti, come nel caso della funzione **fread()**. The **read()** system call returns the number of elements read instead of the number of bytes read, as it is done by the function **fread()**.
6. ✅ Nel caso di creazione di un file, attraverso la system call **open()** si possono impostare i permessi del file creato. When creating a file, the permissions of the newly created file can be set through the **open()** system call.

Ex 2 (2.0 points)

Italiano

Relativamente alle system call **opendir()**, **dirent()** e **closedir()**, si indichino quali delle seguenti affermazioni sono corrette. Si osservi che risposte errate implicano una penalità nel punteggio finale.

English

Regarding the system calls **opendir()**, **dirent()** and **closedir()**, indicate which of the following statements are correct. Note that incorrect answers imply a penalty in the final score.

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

1. ✅ La system call **opendir()** restituisce un puntatore ad un directory stream posizionato sulla prima entry della directory il cui nome è passato in ingresso alla funzione. The **opendir()** system call returns a pointer to a directory stream that is positioned to the first entry of the directory whose name is passed as input to the function.
2. ⬜ La system call **readdir()** restituisce mediante una lista tutte le entry contenute all'interno di una directory. The **readdir()** system call returns all the entries in a directory through a list.
3. ⬜ La system call **readdir()** restituisce `NULL` **solo** nel caso di una directory vuota. The **readdir()** system call returns `NULL` **only** when a directory is empty.
4. ⬜ La system call **opendir()** restituisce un riferimento ad una struttura di tipo `struct dirent`. The **opendir()** system call returns a pointer to a structure of type `struct dirent`.
5. ✅ Usando tali system call è possibile ottenere tutti i numeri di i-node associati alle entries presenti in una directory. Using these three system calls it is possible to retrieve all the i-node numbers associated with the entries in a directory.

Ex 3 (5.0 points)

Italiano

Il comando **`ps -aux`** in Linux visualizza le seguenti informazioni:

```
USER  PID %CPU %MEM    VSZ   RSS    TTY    STAT START  TIME COMMAND
root    1  0.0  0.0  167728  11568  ?      Ss   10:38  0:01 /sbin/init splash
root  223  0.0  0.1  109036  53488  ?      S<s  10:38  0:00 /lib/systemd/systemd
quer 1299 3.9  1.0  3772520 425772 ?      Rsl  10:38  3:50 /usr/bin/gnome-shell
quer 2004 0.2  0.1  824376  60760  ?      Ssl  10:46  0:15 /usr/lib/gnome-terminal
quer 4755 0.0  0.0  11696   3456   pts/0  R+   12:15  0:00 ps -aux
```

Scrivere uno script BASH che analizzi l'output di tale comando e invii un segnale di terminazione a tutti i processi **non** di root per cui l'utilizzo della memoria (`%MEM`) supera il 25%. Si assuma che le colonne dell'output siano separate da un singolo spazio.

English

The command **`ps -aux`** in Linux displays the following information:

```
USER  PID %CPU %MEM    VSZ   RSS    TTY    STAT START  TIME COMMAND
root    1  0.0  0.0  167728  11568  ?      Ss   10:38  0:01 /sbin/init splash
root  223  0.0  0.1  109036  53488  ?      S<s  10:38  0:00 /lib/systemd/systemd
quer 1299 3.9  1.0  3772520 425772 ?      Rsl  10:38  3:50 /usr/bin/gnome-shell
quer 2004 0.2  0.1  824376  60760  ?      Ssl  10:46  0:15 /usr/lib/gnome-terminal
quer 4755 0.0  0.0  11696   3456   pts/0  R+   12:15  0:00 ps -aux
```

Write a BASH script that analyzes the output of such a command and sends a termination signal to all the **non**-root processes that have a memory usage (`%MEM`) larger than 25%. Assume that the columns are separated from each other by a single space.

Soluzione. Solution.

```
#!/usr/bin/env bash

################################################################################
# Version 1: temp file, while-read, cut
################################################################################

# Save ps output to a temporary file
ps -aux | tail -n +2 > tmp.txt

# Process the file line-by-line
while read line; do

  # Extract user, pid and memory percentage from the current line
  user=$(echo $line | tr -s " " | cut -d " " -f 1)
  pid=$(echo $line | tr -s " " | cut -d " " -f 2)
  mem=$(echo $line | tr -s " " | cut -d " " -f 4 | cut -d "." -f 1)

  # Send a termination signal if the process is non-root and has memory usage larger
than 25%
  if [ $user != "root" ] && [ $mem -ge 25 ]; then
      kill -9 $pid
  fi
done < tmp.txt

# Remove temporary file
rm tmp.txt
```

```
#!/usr/bin/env bash

################################################################################
# Version 2: pipelining, greps, xargs
################################################################################

ps -aux | tr -s " " | grep -ve "^root" | grep -E "^[^ ]+ [^ ]+ [^ ]+
(100|[3-9][0-9]|2[6-9]|25\.[1-9])" | cut -d ' ' -f 2 | xargs -n 1 echo
```

Ex 4 (3.0 points)

Italiano

Lo stato di un sistema con 5 processi e 3 tipologie di risorse è definito dalle matrici riportate alla fine della domanda. Si supponga il processo P4 effettui una richiesta per le risorse {1, 1, 1}. Si stabilisca se la richiesta può essere soddisfatta permettendo al sistema di rimanere in uno stato sicuro. In caso affermativo si risponda "YES" e si riporti la sequenza di esecuzione dei processi. Ad esempio si risponda "YES 5 4 3 2 1" nel caso in cui la sequenza sicura sia P5, P4, P3, P2, P1. In caso contrario si risponda "NO".

English

The state of a system with 5 processes and 3 types of resources is defined by the matrices reported at the end of the question. Suppose that process P4 requests the resources {1, 1, 1}. Indicate whether or not the request can be granted, allowing the system to remain in a safe state. In the affirmative case, respond with "YES" and report the sequence in which the processes will be executed. For instance, respond "YES 5 4 3 2 1" in the case the safe sequence is P5, P4, P3, P2, P1. Otherwise, respond with "NO".

Processo Process	Assegnate Allocation	Massimo Max	Disponibile Available
P1	0 0 0	4 3 5	4 3 3
P2	0 0 1	4 4 2	
P3	0 1 0	3 2 2	
P4	1 0 1	2 5 4	
P5	0 1 0	3 4 2	

Risposta. Answer.

YES 3 5 4 2 1

Ex 5 (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 Round Robin (RR), 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 processes, scheduled with a temporal quantum of 10 units. Represent using a Gantt diagram the execution of these processes using the Round Robin (RR) 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	9	1
P2	6	17	3
P3	11	28	5
P4	16	14	4
P5	21	15	2

Fasi operative. Workflow.

```
RR:
  P1: from=  0 to  =  8 end =  9 waitingTime =  0
  P2: from=  9 to  = 18 end = 19 waitingTime =  3
  P3: from= 19 to  = 28 end = 29 waitingTime =  8
  P4: from= 29 to  = 38 end = 39 waitingTime = 13
  P2: from= 39 to  = 45 end = 46 waitingTime = 20
  P5: from= 46 to  = 55 end = 56 waitingTime = 25
  P3: from= 56 to  = 65 end = 66 waitingTime = 27
  P4: from= 66 to  = 69 end = 70 waitingTime = 27
  P5: from= 70 to  = 74 end = 75 waitingTime = 14
  P3: from= 75 to  = 82 end = 83 waitingTime =  9
  EndTime:  9 46 83 70 75
  AverageWaitingTime: [(  0 ) + (  3 + 20 ) + (  8 + 27 +  9 ) + ( 13 + 27 ) + ( 25
+ 14 ) ] / 5 = 29.20
  AverageTournaroudTime: (  9 + 40 + 72 + 54 + 54 ) / 5 = 45.80
  GanttChart:
11111111122222222223333333333444444444422222225555555555333333333344445555533333333
```

Risposta. Answer.

```
9 46 83 70 75 29.2
```

Ex 6 (5.5 points)

Italiano

Un programma esegue N thread di tipo A e N thread di tipo B. N è un intero positivo passato sulla riga di comando (ad esempio, 10) e tutti thread sono identificati dalla loro categoria (A o B) e da un indice di creazione (1, 2, …, N). Si vuole coordinare il lavoro dei thread in modo tale che si esegua sempre un thread dii tipo A prima di un thread di tipo B. Di seguito due esempi di esecuzioni corrette con N=10:

```
A3 B1 A1 B8 A2 B7 A8 B6 A9 B3 A7 B9 A6 B2 A5 B4 A4 B5 A10 B10
A10 B5 A6 B1 A7 B3 A1 B4 A3 B2 A2 B9 A8 B8 A5 B6 A4 B10 A9 B7
```

Si noti che l'ordine con cui i thread di una data categoria devono essere eseguiti non è fissato a priori ma dipende dallo scheduling dei thread. Scrivere il programma in codice C. Realizzare il programma completo, incluso il codice di creazione dei thread.

English

A program runs N threads of type A and N threads of type B. N is a positive integer passed on the command line (e.g., 10), and all threads are identified by their category (i.e., A or B) and a creation index (i.e., 1, 2, ..., N). We want to coordinate the effort of all threads to always run a thread of category A before one thread of category B. In the following, there are two examples of correct execution with N=10:

```
A3 B1 A1 B8 A2 B7 A8 B6 A9 B3 A7 B9 A6 B2 A5 B4 A4 B5 A10 B10
A10 B5 A6 B1 A7 B3 A1 B4 A3 B2 A2 B9 A8 B8 A5 B6 A4 B10 A9 B7
```

Note that the order in which the threads of each category are executed is not fixed, but it depends on the threads' scheduling. Write the code using the C language. Realize the complete program, including the creation of the threads.

Soluzione. Solution.

```
#include <stdio.h>
#include <unistd.h>
#include <ctype.h>
#include <pthread.h>
```

```
#include <stdlib.h>
#include <semaphore.h>

sem_t s_a, s_b;


/* Threads of type A */
static void *TA(void *arg){
  int id;
  id = *((int*)arg);

  while(1){
    sem_wait(&s_a);
    printf("A%d ", id); fflush(stdout);
    sem_post(&s_b);
  }

  pthread_exit(NULL);
}

/* Threads of type B */
static void *TB(void *arg){
  int id;
  id = *((int*)arg);

  while(1){
    sem_wait(&s_b);
    printf("B%d ", id); fflush(stdout);
    sem_post(&s_a);
  }

  pthread_exit(NULL);
}

int main(int argc, char *argv[]){
  int N ;
  int i = 0;
  pthread_t *threads_A;
  pthread_t *threads_B;
  int *id;

  N = atoi(argv[1]);

  /* Allocation */
  sem_init(&s_a, 0, 1);
  sem_init(&s_b, 0, 0);
  threads_A = (pthread_t*)malloc(N*sizeof(pthread_t));
  threads_B = (pthread_t*)malloc(N*sizeof(pthread_t));
  id = (int*)malloc(N*sizeof(int));

  /* Generation */
  for (i=0; i<N;i++){
    id[i] = i;
    pthread_create(&threads_A[i], NULL, TA, (void *)&(id[i]));
    pthread_create(&threads_B[i], NULL, TB, (void *)&(id[i]));
  }
```

```
  /* Join */
  for (i=0; i<N;i++){
    pthread_join(threads_A[i], NULL);
    pthread_join(threads_B[i], NULL);
  }

  /* Deallocation */
  free(threads_A); free(threads_B); free(id);
  return 0;
}
```

Ex 7 (3.0 points)

Italiano

Si supponga che il seguente programma venga eseguito con il valore 5 sulla linea di comando. Si riporti esattamente l'output generato. Si prega di riportare la risposta su un'unica riga, indicando i vari messaggi e valori di output separati da un unico spazio. Non inserire nessun altro carattere nella risposta.

English

Suppose the following program is executed, passing the value 5 in the command line. Report the exact output generated. Please report the response in one line, indicating all the messages and the values in output, which must be separated by a single space. Do not insert any other character in the response.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <semaphore.h>
#include <pthread.h>
#include <stdlib.h>
#include <stdio.h>

static void *T1 ();
static void *T2 ();

void *T1 (void *p){
  pthread_t thread;
  int *pn = (int *) p;
  int n = *pn;
  if (n>0) {
      n-=2;
      printf ("(%d)", n);
      pthread_create (&thread, NULL, T2, &n);
  }
  pthread_join (thread, NULL);
  pthread_exit (NULL);
}
void *T2 (void *p){
  pthread_t thread;
  int *pn = (int *) p;
  int n = *pn;
  if (n>0) {
      n+=1;
      printf ("[%d]", n);
      pthread_create (&thread, NULL, T1, &n);
  }
  pthread_join (thread, NULL);
```

```
  pthread_exit (NULL);
}
int main (int argc, char *argv[]) {
  pthread_t thread;
  int n = atoi (argv[1]);
  setbuf (stdout, 0);
  pthread_create (&thread, NULL, T1, &n);
  pthread_join (thread, NULL);
  return 1;
}
```

Risposta. Answer.

(3) [4] (2) [3] (1) [2] (0)

Ex 8 (5.0 points)

Italiano

Due processi usano una pipe per scambiare dati tra loro. Nello specifico:

- Il processo P1 legge da standard input una sequenza di stringhe e scrive tali stringhe sulla pipe. P1 termina quando viene incontrata la stringa “STOP”.
- Il processo P2 legge le stringhe dalla pipe e le scrive su standard output trasformando ogni lettera maiuscola in minuscola e ogni lettera minuscola in maiuscola.

Si supponga che le stringhe abbiano un numero indefinito di caratteri ma minore di 100.
Si scriva in linguaggio C l’intero programma.

English

Two processes use a pipe to exchange data. More specifically:

- Process P1 reads a sequence of strings from standard input, and it writes these strings onto the pipe. P1 terminates when the string "STOP" is encountered.
- Process P2 reads the strings from the pipe, and it writes them on standard output, transforming each uppercase letter into a lowercase letter and each lowercase letter into an uppercase letter.

Suppose the strings have an undefined number of characters but less than 100.
Write in C code the entire program.

Soluzione. Solution.

```
int main(int argc, char** argv){
  int p[2];
  if(pipe(p)==-1){
    printf("error\n");
    exit(-1);
   }
   if(fork()){
    P1(p);
    wait((void*)NULL);
   } else {
    P2(p);
   }
  return 0;
}

void P1(int* p){
 char str[N];
 int i;
 close(p[0])
 do {
   fscanf(stdin, "%s", str);
   write(p[1], str, strlen(str)+1);
 } while(strcmp(str, "STOP")!=0);
```

```
 close(p[1]);
 return;
}

void P2(int* p){
  char c, last[5];
  int i;
  close(p[1]);
  do {
    i=0;
    do {
      read(p[0],&c,1);
      if(isalpha(c)){
        if((c>'a')&&(c<'z'))
          c=toupper(c);
        else
          c=tolower(c);
      }
      printf("%c",c);
    } while(c!='\n');
    printf("\n");
  while (strcmp(last,"stop")==0);
  close(p[0]);
  return;
}



#define MAX_STR 100

int main() {
  int fd[2];
  pipe(fd);
  if (fork() == 0) {
    close(fd[0]);
    P1(fd[1]);
    return 0;
  }
  if (fork() == 0) {
    close(fd[1]);
    P2(fd[0]);
    return 0;
  }
  wait(0);
  wait(0);
  return 0;
}

void P1(int fd) {
  char buffer[MAX_STR];
  int length = 0;
  do {
    fscanf(stdin, "%s", buffer);
    length = strlen(buffer);
    write(fd, &length, sizeof(int));
    write(fd, buffer, length + 1);
```

```
  } while (strcmp(buffer, "STOP") == 0);
  return;
}

void P2(int fd) {
  int i, length;
  char buffer[MAX_STR];
  do {
    read(fd, &length, sizeof(int));
    read(fd, buffer, length + 1);
    for (i = 0; i < length; i++) {
      if (buffer[i] <= 'z' && buffer[i] >= 'a')
        buffer[i] += 'A' - 'a';
      else if (buffer[i] <= 'Z' && buffer[i] >= 'A')
        buffer[i] += 'a' - 'A';
    }
    printf("%s\n", buffer);
  } while (strcmp(buffer, "stop") == 0);
  return;
}
```

Ex 9 (2.0 points)

Italiano

Quali dei seguenti algoritmi di schedulazione è dotato di prelazione

English

Which of the following scheduling algorithms has preemption.

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

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

Ex 10 (3.5 points)

Italiano

Si faccia riferimento alle soluzioni hardware al problema della sezione critica mediante l'istruzione di **test-and-set**. Se ne illustrino le principali caratteristiche, riportandone il codice e il relativo protocollo di utilizzo, e i principali vantaggi e svantaggi da altri tipi di soluzione.

English

Consider the hardware solutions to the critical section problem that use the **test-and-set** instruction. Describe their main characteristics, report their code and usage protocol, and detail their advantages and disadvantages with respect to other types of solutions to the same problem.

Soluzione. Solution.

Test-and-set utilizza un lock globale che permette di bloccare una parte di codice che deve essere eseguita in mutua esclusione tra processi.
Vantaggi: no deadlock, progresso, ME, simmetria, facilmente estendibile a N processi.
Svantaggi: difficoltà nell'implementazione hardware, spin-lock o busy waiting (consuma cicli di cpu), starvation.

Test-and-set uses a global lock that allows you to lock a piece of code that needs to be executed in mutual exclusion between processes.
Advantages: no deadlock, progress, ME, symmetry, and ease of extension to N processes.

Disadvantages: difficulty in hardware implementation, spin-lock or busy waiting (consumes CPU cycles), starvation.

```
// lock globale
int lock = 0;

// Test and Set
int testAndSet (int *lock){
  int tmp;
  tmp=*lock;
  *lock=1;
  return tmp;
}

// Protocol
while (1) {
  while(testAndSet(&lock));
  SC
  lock=0;
  SnonC
}
```

Ex 11 (2.0 points)

Italiano

Quale tra le seguenti informazioni **non** è memorizzata all'interno del Process Control Block (PCB)?

English

Which of the following pieces of information is **not** stored inside the Process Control Block (PCB)?

Scegli UNA SOLA alternativa. Choose JUST ONE option.

1. ⬜ Stato del processo. Process state.
2. ⬜ Lista dei file aperti. List of open files.
3. ⬜ Registri della CPU. CPU registers.
4. ⬜ Gestori dei segnali. Signal handlers.
5. ⬜ Dati amministrativi ad esempio sull'uso della CPU. Administrative data, for instance related to CPU usage.
6. ⬜ Program Counter. Program Counter.
7. ✅ Tutte le informazioni nelle risposte precedenti sono salvate all'interno del PCB. All the pieces of information in the previous answers are stored within the PCB.