Esame Sistemi Operativi - Operating Systems Exam
2024/06/20

Ex 01 (2.0 points)

Italiano

Si supponga che la memoria principale di un sistema embedded sia costituita da 16 blocchi di 4 KByte ciascuno, che tali blocchi siano numerati da 0 a 15, che il sistema operativo tenga traccia dei blocchi liberi (occupati) indicandoli in un vettore con il valore 0 (1), e che la situazione attuale della memoria sia rappresentata dal seguente vettore:
0 1 1 0 0 1 0 0 1 0 1 1 0 0 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 main memory of an embedded system is composed of 16 blocks of 4 KBytes each, which are numbered from 0 to 15. 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 memory is represented by the following vector:
0 1 1 0 0 1 0 0 1 0 1 1 0 0 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 che richiede 18 KByte di memoria PUO' essere allocato utilizzando una strategia di allocazione contigua. A file requiring 18 KBytes of memory CAN be allocated using a contiguous allocation strategy.
2. ✅ Un file che richiede 8 KByte di memoria PUO' essere allocato utilizzando una strategia di allocazione contigua. A file requiring 8 KBytes of memory CAN be allocated using a contiguous allocation strategy.
3. ✅ Un file che richiede 20 KByte di memoria PUO' essere allocato utilizzando una strategia di allocazione concatenata. A file requiring 20 KBytes of memory CAN be allocated using a linked allocation strategy.
4. ✅ Con la strategia di allocazione contigua ottenuta mediante l'algoritmo BEST-FIT può essere allocato un file di 7 KByte. With the contiguous allocation strategy based on the BEST-FIT algorithm, a file of 7 KBytes can be allocated.
5. ⚪ Con la strategia di allocazione contigua ottenuta mediante l'algoritmo WORST-FIT può essere allocato un file F1 di 6 KByte e successivamente un file F2 di 12 KByte. With the contiguous allocation strategy based on the WORST-FIT algorithm, it is possible to allocate a file F1 of 6 KBytes and a file F2 of 12 KBytes.
6. ⚪ L'allocazione di un file di 11 KByte usando la strategia di allocazione contigua con l'algoritmo BEST-FIT porta ad una frammentazione interna pari a 5 KByte. The allocation of an 11 KBytes file with the contiguous allocation strategy and the BEST-FIT algorithm leads to 5 KBytes of internal fragmentation.
7. ✅ L'allocazione di un file di 9 KByte usando la strategia di allocazione contigua con l'algoritmo BEST-FIT porta ad una frammentazione interna pari a 3 KByte. The allocation of an 9 KBytes file with the contiguous allocation strategy and the BEST-FIT algorithm leads to 3 KBytes of internal fragmentation.

Ex 02 (3.5 points)

Italian

Si supponga che il seguente programma venga eseguito mediante l'istruzione `./pgrm 3`.

Si indichino quanti sono i PID visualizzati. Si osservi che risposte errate implicano una penalità nel punteggio finale.

English

Suppose that the following program is run using the command `./pgrm 3`.
Indicate how many PIDs the program displays. Note that wrong answers imply a penalty in the final score.

```
#define N 100

int main (int argc, char **argv){
  int n;
  char str[N];
  setbuf (stdout, 0);
  printf ("%d ", getpid());
  n = atoi(argv[1]);
  if (n<=0)
    return 1;
  if (fork()==0) {
    sprintf (str, "%d", n-1);
    execlp (argv[0], argv[0], str, NULL);
  }
  sprintf (str, "%s %d", argv[0], n-1);
  system (str);
}
```

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

1. ⬜ 7
2. ⬜ 9
3. ⬜ 11
4. ⬜ 13
5. ✅ 15
6. ⬜ 16
7. ⬜ 17

Ex 03 (3.5 points)

Italiano

Si supponga di eseguire il seguente programma. Si indichino quali sono gli output prodotti ammissibili. Si osservi che risposte errate implicano una penalità nel punteggio finale.

English

Suppose to run the following program. Indicate the possible admissible outputs. Note that wrong answers imply a penalty in the final score.

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

sem_t *s;
int i, n;

static void *T (void *);

int main (int argc, char **argv) {
  char A='A', B='B', C='C';
```

```
  pthread_t th;
  s = (sem_t *) malloc (sizeof(sem_t));
  i = 0;
  n = 1;
  sem_init (s, 0, 1);
  setbuf(stdout, 0);
  pthread_create (&th, NULL, T, &A);
  pthread_create (&th, NULL, T, &B);
  pthread_create (&th, NULL, T, &C);
  pthread_exit(0);
}

static void *T (void *p) {
  char *tmp;
  char c;
  tmp = (char *) p;
  c = *tmp;
  pthread_detach (pthread_self ());
  while (1) {
    sem_wait (s);
    printf ("%c", c);
    i++;
    if (i>=n) {
      printf ("-");
      i = 0;
      n++;
      if (n>=4) {
      n = 1;
      }
    }
    sem_post (s);
  }
  return 0;
}
```

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

1. ✅ B-CA-BCA-B-CA-CBA-C-BA-CBA-C-AB-CBA-...
2. ✅ B-AC-BAC-B-AC-ABC-A-BC-ABC-C-BA-BAC-C-AB-CAB-B-AC-ABC-B-...
3. ✅ A-BA-CBA-B-AC-BAC-C-BA-CAB-...
4. ⬜ AA-AAA-BB-BBB-CC-CCC-...
5. ✅ B-BB-BBB-B-BB-BBB-...
6. ⬜ CC-CCC-CCCC-AA-AAA-AAAA-...
7. ✅ C-CC-CCC-A-AA-AAA-..
8. ⬜ CCC-CC-C-BBB-BB-B-AAA-AA-A-...
9. ⬜ ABC-ABC-ABC-...

Ex 04 (2.0 points)

Italiano

Quali affermazioni descrivono accuratamente le differenze tra processi zombie e orfani in un sistema UNIX? Si osservi che risposte errate implicano una penalità nel punteggio finale.

English

Which statements accurately describe the differences between zombie and orphan processes in a UNIX system? Note that wrong answers imply a penalty in the final score.

1. ✅ Un processo zombie mantiene il proprio ID processo (PID) nella tabella del processo, ma ha completato l'esecuzione. Al contrario, un processo orfano viene creato quando il suo processo padre

esce prima del figlio, rendendo il figlio adottato dal processo init (o processo equivalente). A zombie process retains its process ID (PID) in the process table but has completed execution. In contrast, an orphan process is created when its parent process exits before the child does, making the child adopted by the init process (or an equivalent process).

2. ⬜ I processi zombie non consumano risorse di sistema ad eccezione dell'ID processo nella tabella del processo, mentre i processi orfani consumano risorse di CPU e memoria. Zombie processes consume no system resources except their process ID in the process table, while orphan processes consume CPU and memory resources.
3. ⬜ Il processo init può rimuovere un processo zombie dalla tabella dei processi, mentre l'amministratore di sistema deve terminare manualmente un processo orfano. The init process can remove a zombie process from the process table, while the system administrator must manually terminate an orphan process.
4. ⬜ Sia i processi zombie che quelli orfani continuano a essere eseguiti e possono influire sulle prestazioni del sistema consumando risorse fino a quando non vengono terminati in modo esplicito. Both zombie and orphan processes continue running and can affect system performance by consuming resources until explicitly terminated.
5. ⬜ I processi orfani vengono riassegnati a nuovi PID una volta adottati dal processo init, mentre i processi zombie mantengono quelli originali fino a quando non vengono raccolti. Orphan processes are reassigned to new PIDs once adopted by the init process, while zombie processes retain their original ones until reaped**.**
6. ✅ I processi zombie non possono essere terminati utilizzando il comando kill in quanto sono già morti, mentre i processi orfani possono essere terminati utilizzando il comando kill. Zombie processes cannot be terminated using the kill command as they are already dead, whereas orphan processes can be terminated using the kill command.
7. ⬜ I processi orfani causano perdite di memoria trattenendo la memoria inutilizzata, a differenza dei processi zombie, che utilizzano solo una voce nella tabella dei processi. Orphan processes cause memory leaks by holding onto unused memory, unlike zombie processes, which only consume an entry in the process table.

Ex 05 (5.0 points)

Italiano

Scrivere uno script BASH che riceva un percorso sulla riga di comando ed esegua le attività seguenti:

1. Cerca ricorsivamente nella directory corrente e in tutte le sottodirectory i file con estensione ".txt".
2. Se un file ".txt" è inferiore a 1 MByte, il suo contenuto viene concatenato a un file denominato small_files_N.txt, dove N è un valore intero a partire da 1 (per la prima concatenazione).
3. Se un file ".txt" è più grande di 1 GByte, deve essere suddiviso in file più piccoli di circa 100 MB ciascuno. I file divisi devono mantenere il nome del file originale con un suffisso numerico e l'estensione ".txt". Ad esempio, se il file originale è "largefile.txt", i file divisi devono essere denominati "largefile_1.txt", largefile_2.txt, ecc.

I file concatenati o divisi devono essere creati nella stessa directory del file originale. Si ricordi che i comandi `head` e `tail` posseggono una opzione `"-c"` per visualizzare uno specifico nuomero di bytes.

English

Write a BASH script that receives a path on the command line and performs the following tasks:

1. It recursively searches the current directory and all subdirectories for files with the ".txt" extension.
2. If a ".txt" file is smaller than 1 MByte, its content is concatenated to a file named small_files_N.txt, where N is an integer value starting from 1 (for the first concatenation).
3. If a ".txt" file is larger than 1 GByte, it should be split into smaller files of approximately 100 MB each. The split files should retain the original filename with a numeric suffix and the ".txt" extension. For example, if the original file is "largefile.txt", the split files should be named "largefile_1.txt", largefile_2.txt, etc.

The concatenated or split files should be created in the same directory as the original file. Note that the `head` and `tail` commands have a `"-c"` option to display a specific number of bytes.

Solution. Solution.

```
#!/bin/bash
```

```
# Check if a path was provided
if [ -z "$1" ]; then
  echo "Usage: $0 <directory path>"
  exit 1
fi

# Get the directory path from the command line argument
directory="$1"

# Check if the provided path is a directory
if [ ! -d "${directory}" ]; then
  echo "The provided path is not a directory."
  exit 1
fi

# List all files in the directory and its subdirectories
# NOTE: The + and - prefixes signify greater than and less than, as usual;
#       i.e., an exact size of n units does not match.
#       Bear in mind that the size is rounded up  to  the  next unit.
#       Therefore -size -1M is not equivalent to -size -1048576c.
#          The former only matches empty files, the latter matches files from 0 to
1,048,575 bytes.
small_files=$(find "${directory}" -type f -name "*.txt" -size -1048576c)
large_files=$(find "${directory}" -type f -name "*.txt" -size +1G)

# Initialize the index for small files
small_files_index=1

# Loop over the small files
for file in ${small_files}; do
    # Concatenate contents to small_files_N.txt
    cat "${file}" >> "./small_files_${small_files_index}.txt"

    # Increment the index
    ((small_files_index++))
done

# Loop over the large files
for file in ${large_files}; do
    # Initialize the index for large files
    large_files_index=1

    # Get the large file size
    file_size=$(cat "${file}" | wc -c)

    # Get the large file basename
    large_base_name=$(basename "${file}" ".txt")

    # Split the large file into chunks of 100MB renaming it basename_N.txt
    chunk_size=$((100 * 1024 * 1024))
    head_size=$((${chunk_size} * ${large_files_index}))
    while [ $head_size -le $file_size ]; do
             cat  "${file}"  |  head  -c  "${head_size}"  |  tail  -c  "${chunk_size}"  >
"./${large_base_name}_${large_files_index}.txt"
        # Increment the index
```

```
        ((large_files_index++))
        head_size=$((${chunk_size} * ${large_files_index}))
    done

    # Check if there is a remainder
    remainder=$((${file_size} % ${chunk_size}))

    # Concatenate the remainder of the file
    if [ $remainder -ne 0 ]; then
                            cat    "${file}"    |    tail    -c    "${remainder}"    >
"./${large_base_name}_${large_files_index}.txt"
    fi
done
```

Ex 06 (2.0 points)

Italiano

Sia dato il seguente comando bash:

```
egrep "^2.*[13579]\>" file.txt | egrep "B(BB)*" | tr -s "B" | cut -d " " -f 1
```

Riportare quali righe vengono riportate in output quando il comando viene eseguito sul file file.txt riportato alla fine della domanda.

Si osservi che risposte errate implicano una penalità nel punteggio finale.

English

Given the following bash command:

```
egrep "^2.*[13579]\>" file.txt | egrep "B(BB)*" | tr -s "B" | cut -d " " -f 1
```

Report which lines are displayed when the previous command is executed on the file file.txt reported at the end of the question.

Note that incorrect answers imply a penalty in the final score.

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

```
20215 BBB
20134 B
21199 BBBBB
22157 B
25371 BBBB
20231 BBB
31245 BBB
32123 BBBB
```

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

1. ⬜ 20215 BBB
2. ✅ 20215
3. ✅ 21199
4. ⬜ 21199 BBBBB
5. ✅ 25371
6. ⬜ 32123 BBBB
7. ⬜ 32123
8. ✅ 20231
9. ⬜ 25371 BBBB
10. ⬜ 22157 B
11. ✅ 22157

Ex 07 (5.0 points)

Italiano

Si scriva un programma in linguaggio C che genera due processi P1 e P2 e rimane in attesa della loro terminazione. I processi P1 e P2 devono trasmettersi reciprocamente delle stringhe su una pipe e devono sincronizzare tali operazioni di trasmissione mediante l'utilizzo dei segnali SIGUSR1 e SIGUSR2.
Più nel dettaglio il processo P1 deve:

1. Attendere da P2 un segnale SIGUSR1.
2. Una volta ricevuto tale segnale, leggere da standard input una stringa e trasmetterla a P2 sulla pipe.
3. Inviare a P2 un segnale SIGUSR2 per avvertirlo che la stringa è stata scritta nella pipe.
4. Ritornare nello stato di attesa descritto al punto 1.

Il processo P2 deve:

A. Inviare a P1 un segnale SIGUSR1.
B. Attendere da P1 un segnale SIGUSR2.
C. Una volta ricevuto tale segnale.
    a. Leggere dalla pipe la stringa inviata da P1.
    b. Trasformare tutti i suoi caratteri minuscoli in maiuscoli.
    c. Trasformare tutti i suoi caratteri maiuscoli in minuscoli.
    d. Visualizzare la stringa su standard output.
D. Ritornare nello stato descritto al punto A.

Si supponga che le stringhe gestite dal programma abbiano lunghezza **indefinita** ma limitata a **1000** caratteri. Si supponga inoltre esse includano esclusivamente caratteri alfabetici maiuscoli oppure minuscoli. Infine, si osservi che l'utilizzo della pipe **deve** permettere la gestione di stringhe di **lunghezza variabile**.

English

Write a program in the C language that generates two processes, P1 and P2, and waits for their termination. Processes P1 and P2 must transmit strings to each other over a pipe and must synchronize transmission operations using SIGUSR1 and SIGUSR2 signals.
More specifically, the P1 process must:

1. Wait to receive from P2 a signal SIGUSR1.
2. Once you have received this signal, read a string from the standard input and transmit it to P2 on the pipe.
3. Send P2 a SIGUSR2 signal to inform it that the string has been written to the pipe.
4. Return to the waiting state described in step 1.

The P2 process must:

A. Send a SIGUSR1 signal to P1.
B. Wait to receive from P1 a SIGUSR2 signal.
    a. Once such a signal is received.
    b. Read the string sent by P1 from the pipe.
    c. Turn all of its lowercase characters into uppercase.
    d. Turn all of its uppercase characters into lowercase.
    e. Display the string on Standard Output.
C. Return to the state described in step A.

Assume that the strings handled by the program are **indefinite** but limited to **1000** characters. Also, suppose they include only uppercase or lowercase alphabetic characters. Finally, note that pipes **allow** the handling of **variable-length** strings.

Risposta: Answer:

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

#define N 1000+1

static void sigmgr (int);
static void p1(int, int []);
```

```
static void p2(int, int []);

static void sigmgr (int sig) {
#if 0
  if (sig==SIGUSR1)
    fprintf (stdout, "Received SIGUSR1.\n");
  else
    if (sig==SIGUSR2)
      fprintf (stdout, "Received SIGUSR2.\n");
    else
      fprintf (stdout, "Received WRONG signal.\n");
#endif
  return;
}

int main () {
  pid_t pp1, pp2;
  int p[2];

  setbuf (stdout, 0);

  signal (SIGUSR1, sigmgr);
  signal (SIGUSR2, sigmgr);

  if (pipe (p)!= 0) {
    fprintf (stderr, "Error on pipe.\n");
    return (0);
  }

  pp1 = fork();
  if (pp1==0) {
    pp1 = getpid();
    pp2 = fork();
    if (pp2==0) {
      // P2
      p2 (pp1, p);
    } else {
      // p1
      p1 (pp2, p);
      waitpid (pp2, NULL, 0);
    }
  } else {
    // Original P
    waitpid (pp1, NULL, 0);
  }

  return (0);
}

static void p1 (int pp2, int p[]) {
  char s[N];
```

```
  int l;

  close (p[0]);
  while (1) {
    pause();
    fprintf (stdout, "P1 - send string: ");
    scanf ("%s", s);
    l = strlen (s);
    write (p[1], &l, sizeof (int));
    write (p[1], s, l+1);
    kill (pp2, SIGUSR2);
  }

  return;
}

static void p2 (int pp1, int p[]) {
  char s[N];
  int l;

  close (p[1]);
  while (1) {
    kill (pp1, SIGUSR1);
    pause();
    read (p[0], &l, sizeof (int));
    read (p[0], s, l+1);
    for (int i=0; i<l; i++) {
      if (islower (s[i]))
     s[i] = toupper (s[i]);
      else
        if (isupper (s[i]))
       s[i] = tolower (s[i]);
    }
    fprintf (stdout, "P2 - received string: %s\n", s);
  }

  return;
}
```

Ex 08 (4.0 points)

Italiano

Si descriva il problema dei Readers e Writers nel caso di precedenza ai Readers. Si indichi il significato delle varie primitive di sincronizzazione utilizzate.

English

Describe the problem of the Readers and Writers in the case of precedence to Readers. Indicate the meaning of the various synchronization primitives used.

Risposta: Answer:

The problem of Readers and Writers is a problem of synchronization in case of access to a common resource (critical section). In particular, multiple readers can access the critical section at the same time if no writer is accessing it, while a writer can access it only if there is no reader using it and no other writer. For the case of precedence to readers specifically, "precedence to readers" means prioritizing reader processes over writer

processes when accessing a shared resource.
The following is the implementation:

```
// INITIALIZATION
nR=0

init(meR, 1)
init(meW, 1)
init(w, 1)

//READER

wait(meR);

nR++;

if (nR == 1){

    wait(w);

}

signal(meR);

... reading...

wait(meR);

nR--;

if (nR == 0){

signal(w);

}

signal(meR);


//WRITER
wait(meW);

wait(w);

... writing...
signal(w);

signal(meW);
```

The semaphore "meR" is used for mutual exclusion between readers to access the variable nR, read its value, and modify it without concurrency between multiple readers.

The semaphore "w" is used for moving the possibility of accessing the critical section from readers to writers when there are no other readers accessing it in that moment. It is used then by the writers to give back the access to readers after every writing.

"nR" represents the number of readers present at the same time in the critical section. It is used to acquire access permission or to release it to writers.

The semaphore "meW" is used for enforcing precedence to readers. It is better to use it, but is not strictly necessary.

Ex 09 (2.0 points)

Italiano

In riferimento all'uso dei segnali in un ambiente UNIX/Linux. Dire quali delle seguenti risposte sono vere. Si osservi che risposte errate implicano una penalità nel punteggio finale.

English

Refer to the use of signals in a UNIX/Linux environment. 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. ✅ Un signal handler può essere eseguito in risposta alla ricezione di un segnale. A signal handler can be executed in response to the reception of a signal.
2. ⬜ I segnali in un sistema UNIX/Linux sono utilizzati solo per la terminazione dei processi. Signals in a UNIX/Linux system are used only for process termination.
3. ⬜ Usando la system call signal è possibile ignorare ogni tipo di segnale. By using the system call signal it is possible to ignore any type of signal.
4. ✅ All'interno di un signal handler dovrebbero essere utilizzate solo funzioni rientranti. Inside a signal handler, only reentrant functions should be used.
5. ⬜ La system call raise() invia un segnale a un altro processo. The system call raise() sends a signal to another process.
6. ✅ La funzione sleep() può essere interrotta dalla ricezione di un segnale. The sleep() function can be interrupted by the reception of a signal.
7. ⬜ La ricezione di un segnale da parte di un processo causa sempre la terminazione del processo. The reception of a signal by a process always causes the termination of the process.
8. ⬜ L'uso della coppia di funzioni kill() e pause() o delle primitive semaforiche sem_post() e sem_wait() sono equivalenti per la sincronizzazione dei processi. The use of the pair of functions kill() and pause() or of the semaphore primitives sem_post() and sem_wait() are equivalent for process synchronization.
9. ✅ L'esecuzione di un signal handler dopo la ricezione di un segnale da parte di un processo può portare a race conditions. The execution of a signal handler after the reception of a signal by a process can lead to race conditions.
10. ✅ Entrambi il comando di shell kill e la system call kill() possono terminare un processo. Both the shell command kill and the system call kill() can terminate a process.

Ex 10 (2.0 points)

Italiano

Un programma multi-thread è costituito da diversi thread. Il thread A prima di terminare effettua una exit(), il thread B una pthread_exit() e il thread C una return.
Si indichino quali delle seguenti affermazioni sono vere. Si osservi che risposte errate implicano una penalità nel punteggio finale

English

A multi-threaded program consists of several threads. Thread A executes an exit() before terminating, thread B a pthread_exit(), and thread C a return.
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. ⬜ Per non far terminare gli altri thread la exit() deve essere effettuata solo dalla funzione iniziale del thread (ad esempio main()). In order not to terminate the other threads, the exit() must be performed only by the initial function of the thread (e.g., main()).
2. ⬜ Tutti gli altri thread terminano sicuramente insieme al thread B. All other threads end surely with thread B.

3. ⬜ Tutti gli altri thread terminano sicuramente insieme al thread C. All other threads end surely with thread C
4. ✅ Per poter terminare il thread C deve effettuare la return dalla sua funzione iniziale del thread (ad esempio main()). In order to terminate, thread C must perform the return from its initial function of the thread (e.g., main()).
5. ✅ Tutti gli altri thread terminano sicuramente insieme al thread A. All other threads end with thread A.

Ex 11 (2.5 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). Si calcolino i tempi di terminazione di ciascun processo e il tempo di attesa medio. Si riporti 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 numeri e stringhe 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. Using a Gantt diagram, represent the execution of these processes using the Round-Robin (RR) scheduling algorithms. Compute the termination time of each process and the average waiting time. Report your answer on a single line, indicating P1, P2, P3, P4, and P5 termination times, 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	23	4
P2	0	16	5
P3	7	13	2
P4	9	19	1
P5	11	25	3

Soluzione. Solution.

TotalWaitingTime:
P1=58
P2=50
P3=49
P4=50
P5=60
AverageWaitingTime: 53.40
GanttChart:
1111111111222222222233333333334444444444111111111155555555552222223334444444441115555555555555555

Risposta. Answer.

58 50 49 50 60 53.40

Ex 12 (2.5 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 P5 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 P5 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” if the safe sequence is P5, P4, P3, P2, P1. Otherwise, respond with “NO”.

Processo **Process**	**Necessità** **Need**	**Massimo** **Max**	**Disponibile** **Available**
P1	1 0 1	2 5 4	3 2 3
P2	0 2 0	3 4 2	
P3	0 0 0	4 3 5	
P4	0 0 1	4 4 2	
P5	0 1 0	3 2 2	

Soluzione. Solution.

La soluzione è NO perché il processo P5 non può eseguire la richiesta {1, 1, 1}, perché la necessità del processo P5 è {0, 1, 0}, cioè non c'è la possibilità di richiedere un'altra risorsa di tipo R0 e R2, e l'algoritmo del banchiere termina con un errore.

The solution is NO because process P5 cannot perform the request {1, 1, 1}, because the need of process P5 is {0, 1, 0}, i.e., there is no possibility to request one more resource of type R0 and R2, and the banker algorithm finishes with an error.

Risposta. Answer.

NO