Caselle riservate	
Ex. 1	
Ex. 2	
Ex. 3	
Ex. 4	
Ex. 5	
Ex. 6	
Tot.	

Sistemi Operativi

Compito d'esame

05 Settembre 2019

Matricola _____ Cognome _____ Nome _____

Docente: ◯ Quer ◯ Sterpone

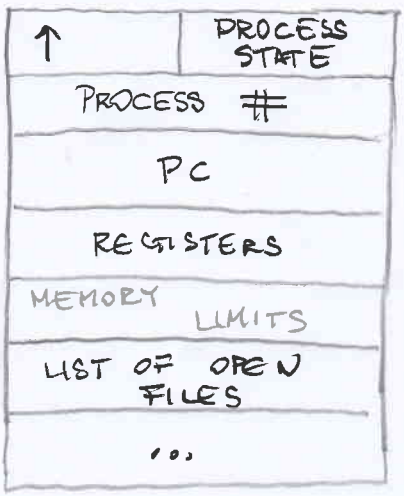
Non si possono consultare testi, appunti o calcolatrici a parte i formulari distribuiti dal docente. Risolvere gli esercizi negli spazi riservati. Fogli aggiuntivi sono permessi solo quando strettamente necessari. Riportare i passaggi principali.
Durata della prova: 100 minuti.

1. Si specifichi che cosa si intende per Process Control Block e per Context Switching.

   Si rappresenti e si descriva il diagramma degli stati di un processo. Si indichi inoltre che cosa si intende per "processo orfano" e per "processo zombie".

   Si riporti infine un grafo di precedenza realizzabile con le sole system call `fork` e `wait` e uno irrealizzabile.

PCB



CONTEXT SWITCHING

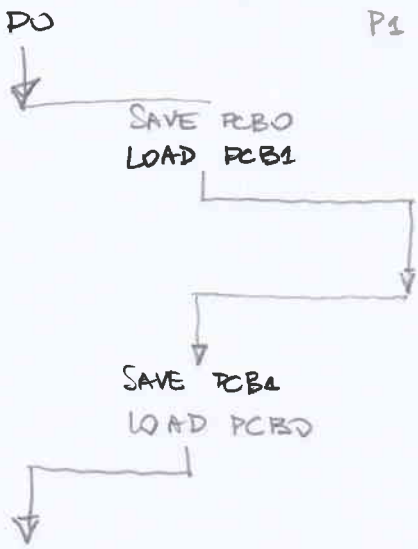
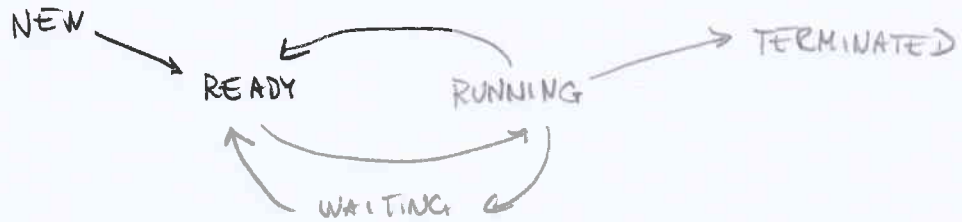


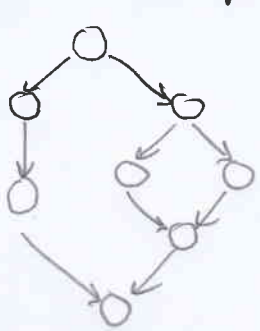
DIAGRAMMA A STATI



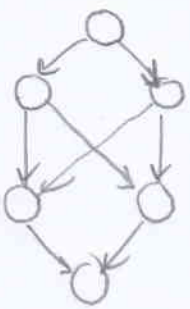
ORFANO: genitore muore; viene ereditato da init

ZOMBIE: P muore; genitore non fa la wait

GRAFO REALIZZABILE



GRAFO NON REALIZZABILE

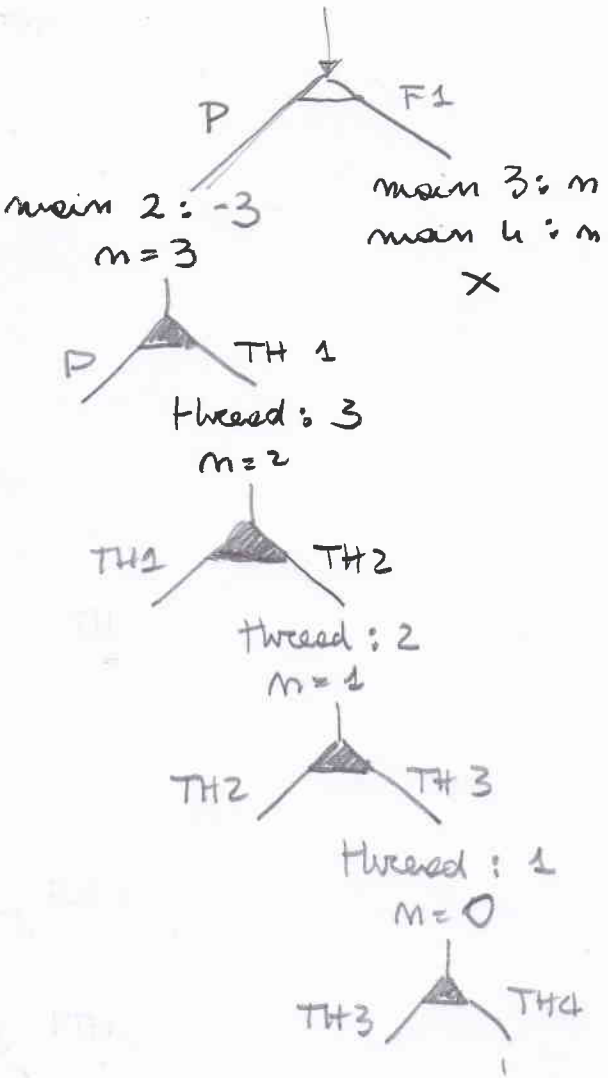


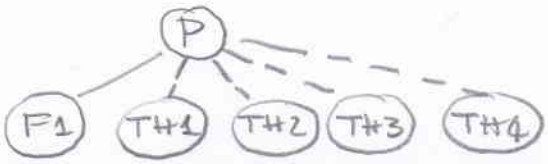
2. Si supponga di eseguire il programma successivo

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
void *t1 (void *p){
  pthread_t thread;
  int *pn = (int *) p;
  int n = *pn;
  if (n>0) {
    printf ("thread: %d\n", 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);
  printf ("main 1: %d\n", n);
  if (fork()) {
    printf ("main 2: %d\n", -n);
    pthread_create (&thread, NULL, t1, &n);
    pthread_join (thread, NULL);
  } else {
    system ("echo main 3: n\n");
    execlp ("echo", "bash", "main 4:", "n", NULL);
  }
  return 1;
}
```

con un unico parametro intero di valore 3.

Si riporti il grafo di controllo del flusso e l'albero di generazione dei processi a seguito della sua esecuzione. Si indichi inoltre che cosa esso produce su video e per quale motivo.



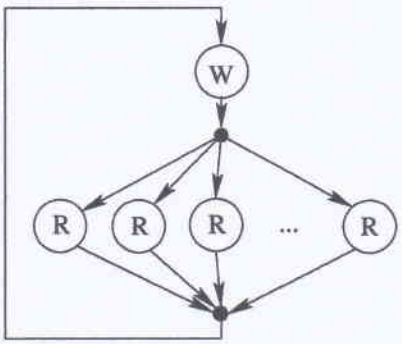


OUTPUT (possibile)

main 1: 3
main 2: -3
main 3: n
main 4: n
thread: 3
thread: 2
thread: 1

3. Si illustri il problema dei *Readers e Writers*. Se ne riporti la soluzione mediante primitive semaforiche nel caso di precedenza ai Readers illustrando il significato di ciascun semaforo. Che cosa si intende per "precedenza ai Readers"?

   Partendo dal problema precedente, si realizzi lo schema di sincronizzazione rappresentato dal seguente grafo di precedenza.



Il numero di Reader è ignoto

Ciclicamente, l'esecuzione di un unico Writer viene seguita dall'esecuzione di un numero indefinito di Reader. All'inizio occorre eseguire il Writer. Al termine del Writer, occorre eseguire uno o più Reader. Però, quando l'ultimo Reader esce dalla regione critica, prima che vi entri un ulteriore Reader, è necessario che un nuovo Writer venga eseguito.

1W → nR → 1W → nR ...

(R)

```
wait (meR)
    nR++
signal (meR)
~
wait(meR)
    nR--
    if(nR==0) signal (meW)
    else  signal (meR)
```

(W)

```
wait(meW)
~
signal (meR)
```

```
nR=0
init(meR, ∅)
init (meW, 1)
```

OR

(R)

```
WFSO (meR, ∞)
nR++
if(nR==1) WFSO(Ron ,∞)
RS(meR, 1, NULL)
~
WFSO (meR, ∞)
nR --
if(nR==0) RS( Won, 1, NULL)
RS(meR, 1, NULL)
```

(W)

```
WFSO(Won, ∞)
~
RS(Ron, 1, NULL)
```

~ o ~ o ~ o

```
nR=nW=0
init (meR, 0)
init (meW, 1)
init (W, 1)
```

~ o ~ o ~ o

nW → nR → nW → ..

```
wait (meR)
    nR++
signal (meR)
~
wait(meR)
   nR--
   if(nR==0) signal(meW)
   else signal(meR)
```

```
wait(meW)
   nW++
signal(meW)
wait (W)
~
signal(W)
wait(meW)
   nW--
   if(nW==0) signal(meR)
   else signal(meW)
```

4. Un file di testo, di nome `virus.dat`, memorizza un elenco di PID su righe successive.

   Si scriva uno script BASH in grado di verificare quali processi elencati nel file `virus.dat` sono in esecuzione nel sistema. Per ciascuno di tali processi in esecuzione, lo script visualizzi il nominativo del proprietario e l'elenco dei PID di tutti i suoi processi figlio.

   Si ricorda che il comando `ps -ef` fornisce un output simile al seguente:

```
UID          PID  PPID  C STIME TTY          TIME CMD
root           1     0  0 giu14 ?        00:00:08 /sbin/init splash
syslog       630     1  0 giu14 ?        00:00:00 /usr/sbin/rsyslogd -n
avahi        665   643  0 giu14 ?        00:00:00 avahi-daemon: chroot helper
quer         938     1  0 giu14 ?        00:00:01 /lib/systemd/systemd --user
quer         946   938  0 giu14 ?        00:00:00 (sd-pam)
...
```

```
#!/bin/bash

# Check arguments
if [ $# -ne 1 ]; then
  echo "Usage: es4.sh <filename>"
  exit 1
fi

# Read input file
while read pid; do

  # Check if process is running
  res=$(ps -ef | tr -s " " | cut -d " " -f 1,2 | grep -e " $pid$")
  if [ $? -eq 0 ]; then

    # Get process name
    name=$(echo $res | cut -d " " -f 1)

    # Find PIDs of children
    children=$(ps -ef | tr -s " " | cut -d " " -f 2,3 | grep -e " $pid$" | cut -d "
 -f 1 | tr '\n' ' ')

    # Print output
    echo "$pid [$name]: $children"
  fi

done < $1
```

5. **Per i candidati iscritti al corso nell'anno accademico 2018–2019.**
   Si scriva un programma multi-thread in grado di leggere e scrivere una matrice di interi come segue.

   La matrice, staticamente definita di `R` righe e `C` colonne, deve essere prima letta da standard input e poi scritta su standard output. Entrambe le operazioni devono essere effettuate procedendo in ordine, dalla riga `0` alla riga `R-1` e, per ciascuna riga, dalla colonna `0` alla colonna `C-1`.

   Le operazioni di lettura e scrittura devono essere effettuate dal programma utilizzando un'unica funzione thread di lettura e da un'unica funzione thread di scrittura, entrambe eseguite `R` volte. All'inizio il programma esegue gli `R` thread di lettura, i quali sincronizzandosi tra di loro leggono la matrice nell'ordine indicato (il primo thread legge la prima riga mentre gli altri aspettano, quindi il secondo thread legge la seconda riga mentre gli altri aspettano, etc.). Ogni thread effettua la lettura di un'intera riga. Terminata la lettura, il programma esegue gli `R` thread di scrittura, i quali sincronizzandosi tra di loro scrivono la matrice nell'ordine indicato. Ogni thread effettua la scrittura di un'intera riga. Il programma quindi termina.

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

#define R 4
#define C 4

/*
 * SOLUTION A: separate semaphores.
 */

// Global variables
int mat[R][C];
pthread_t tids[R];
sem_t semaphores[R-1];

// Function to read a single row of the matrix from stdin
void* read_row(void *arg) {

  // Get row index
  int i = (intptr_t) arg;

  // Wait for the threads reading preceeding rows
  if(i > 0) {
    sem_wait(&(semaphores[i-1]));
  }

  // Print prompt message
  fprintf(stdout, "Inserting values for row %d\n", i);

  // Read row from stdin
  for(int j=0; j<C; ++j) {
    fscanf(stdin, "%d", &(mat[i][j]));
  }

  // Signal reading of the current row completed to the next thread
  if(i<R-1) {
    sem_post(&(semaphores[i]));
  }

  return NULL;
}

// Function to write a single row of the matrix to stdout
void* write_row(void *arg) {

  // Get row index
  int i = (intptr_t) arg;

  // Wait for the threads writing preceeding rows
  if(i > 0) {
    sem_wait(&(semaphores[i-1]));
  }

  // Write row to stdout
  for(int j=0; j<C; ++j) {
    fprintf(stdout, "%d ", mat[i][j]);
  }

  // Print new line
  fprintf(stdout, "\n");

  // Signal writing of the current row completed to the next thread
```

```
  if(i<R-1) {
    sem_post(&(semaphores[i]));
  }

  return NULL;
}


int main(int argc, char **argv) {

  // Prepare semaphores
  for(int i=0; i<R-1; ++i) {
    sem_init(&(semaphores[i]), 0, 0);
  }

  // Launch reading threads
  for(int i=0; i<R; ++i) {
    pthread_create(&(tids[i]), NULL, read_row, (void *) (intptr_t) i);
  }

  // Wait termination of reading threads
  for(int i=0; i<R; ++i) {
    pthread_join(tids[i], NULL);
  }

  fprintf(stderr, "\n");

  // Launch writing threads
  for(int i=0; i<R; ++i) {
    pthread_create(&(tids[i]), NULL, write_row, (void *) (intptr_t) i);
  }

  // Wait termination of writing threads
  for(int i=0; i<R; ++i) {
    pthread_join(tids[i], NULL);
  }
  return 0;
}
```

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

#define R 4
#define C 4

/*
 * SOLUTION B: turn variable, turn-specific threads.
 */

// Global variables
int mat[R][C];
pthread_t tids[R];
sem_t mutex;
int turn = 0;

// Function to read a single row of the matrix from stdin
void* read_row(void *arg) {

  // Get row index
  int i = (intptr_t) arg;

  // Wait until thread turn
  while(1) {
    sem_wait(&mutex);
    if(turn == i) {
      break;
    }
    sem_post(&mutex);
  }

  // Print prompt message
  fprintf(stdout, "Inserting values for row %d\n", i);

  // Read row from stdin
  for(int j=0; j<C; ++j) {
    fscanf(stdin, "%d", &(mat[i][j]));
  }

  // Pass the turn
  turn = (turn + 1) % R;
  sem_post(&mutex);

  return NULL;
}

// Function to write a single row of the matrix to stdout
void* write_row(void *arg) {

  // Get row index
  int i = (intptr_t) arg;

  // Wait until thread turn
  while(1) {
    sem_wait(&mutex);
    if(turn == i) {
      break;
    }
    sem_post(&mutex);
  }

  // Write row to stdout
```

```
  for(int j=0; j<C; ++j) {
    fprintf(stdout, "%d ", mat[i][j]);
  }

  // Print new line
  fprintf(stdout, "\n");

  // Pass the turn
  turn = (turn + 1) % R;
  sem_post(&mutex);

  return NULL;
}


int main(int argc, char **argv) {

  // Prepare mutex
  sem_init(&mutex, 0, 1);

  // Launch reading threads
  for(int i=0; i<R; ++i) {
    pthread_create(&(tids[i]), NULL, read_row, (void *) (intptr_t) i);
  }

  // Wait termination of reading threads
  for(int i=0; i<R; ++i) {
    pthread_join(tids[i], NULL);
  }

  fprintf(stderr, "\n");

  // Launch writing threads
  for(int i=0; i<R; ++i) {
    pthread_create(&(tids[i]), NULL, write_row, (void *) (intptr_t) i);
  }

  // Wait termination of writing threads
  for(int i=0; i<R; ++i) {
    pthread_join(tids[i], NULL);
  }
  return 0;
}
```

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

#define R 4
#define C 4

/*
 * SOLUTION C: turn variable, turn agnostic threads.
 */

// Global variables
int mat[R][C];
pthread_t tids[R];
sem_t mutex;
int turn = 0;

// Function to read a single row of the matrix from stdin
void* read_row(void *arg) {

  // Wait for the turn to be free
  sem_wait(&mutex);

  // Print prompt message
  fprintf(stdout, "Inserting values for row %d\n", turn);

  // Read row from stdin
  for(int j=0; j<C; ++j) {
    fscanf(stdin, "%d", &(mat[turn][j]));
  }

  // Pass the turn
  turn = (turn + 1) % R;
  sem_post(&mutex);

  return NULL;
}

// Function to write a single row of the matrix to stdout
void* write_row(void *arg) {

  // Wait for the turn to be free
  sem_wait(&mutex);

  // Write row to stdout
  for(int j=0; j<C; ++j) {
    fprintf(stdout, "%d ", mat[turn][j]);
  }

  // Print new line
  fprintf(stdout, "\n");

  // Pass the turn
  turn = (turn + 1) % R;
  sem_post(&mutex);

  return NULL;
}


int main(int argc, char **argv) {

  // Prepare mutex
```

```
  sem_init(&mutex, 0, 1);

  // Launch reading threads
  for(int i=0; i<R; ++i) {
    pthread_create(&(tids[i]), NULL, read_row, NULL);
  }

  // Wait termination of reading threads
  for(int i=0; i<R; ++i) {
    pthread_join(tids[i], NULL);
  }

  fprintf(stderr, "\n");

  // Launch writing threads
  for(int i=0; i<R; ++i) {
    pthread_create(&(tids[i]), NULL, write_row, NULL);
  }

  // Wait termination of writing threads
  for(int i=0; i<R; ++i) {
    pthread_join(tids[i], NULL);
  }
  return 0;
}
```

Per i candidati iscritti al corso prima dell'anno accademico 2018–2019.
Un file di testo riporta i risultati di tutti gli studenti di un certo corso di laurea in tutti gli appelli della sessione. Il file riporta su ciascuna riga il risultato di uno studente a un singolo appello, con formato:

```
matricola cognome nome esame voto
```

Si osservi che tutti i voti sono indicati con valori interi di valore massimo 30 e un esame si considera superato con una votazione superiore o uguale a 18.

Si scriva uno script AWK in grado di:

- Ricevere, sulla riga di comando, il nome di un file avente il formato precedentemente descritto.
- Valutare, per ciascuno studente, la media di tutti gli esami superati (ovvero sufficienti).
- Valutare, per ciascun esame, la media dei voti di tutti gli studenti che lo hanno sostenuto (sufficienti o meno).

Per esempio, se il file di ingresso è il seguente:

```
123 c1 n1 OperatingSystem 14
123 c1 n1 Mathematics 20
234 c2 n2 Mathematics 20
345 c3 n3 Mathematics 25
123 c1 n1 English 18
234 c2 n2 English 12
345 c3 n3 English 30
123 c1 n1 CProgramming 30
234 c2 n2 CProgramming 28
345 c3 n3 CProgramming 10
```

Lo script deve visualizzare quanto segue:

```
Media studenti:
  123 c1 n1 22.7
  234 c2 n2 24.0
  345 c3 n3 27.5
Media esami:
  Mathematics 21.6
  CProgramming 22.6
  OperatingSystem 14.0
  English 20.0
```

```
{
  studenti[$1]=$2" "$3
  if($5 >= 18) {
    voti_studenti_tot[$1] += $5
    voti_studenti_num[$1] += 1
  }
  voti_esami_tot[$4] += $5
  voti_esami_num[$4] += 1
}

END {
  printf("Media studenti:\n")
  for(studente in voti_studenti_tot) {
    if(voti_studenti_num[studente] > 0) {
      media = voti_studenti_tot[studente]/voti_studenti_num[studente]
    } else {
      media = 0
    }
    printf("\t%s %s %.1f\n", studente, studenti[studente], media)
  }
  printf("Media esami:\n")
  for(esame in voti_esami_tot) {
    media = voti_esami_tot[esame]/voti_esami_num[esame]
    printf("\t%s %.1f\n", esame, media)
  }
}
```

6. Nell'allocazione indicizzata in ambiente UNIX/LINUX si indichi che cosa si intende con i seguenti termini (possibilmente utilizzando ausili grafici opportuni): directory block, directory entry, data block e i-node.

In un file system UNIX, specificare che cosa si intende per permessi (o diritti), "owner", "group" e "other". Come si possono modificare i permessi, l'owner e il group di un file o di un direttorio?

Si chiarisca il significato di soft-link e hard-link, riportando i comandi per crearli. Sia `s1` un soft-link al file `s`, `h1` un hard-link al file `h`. In tale circostanza di indichi che cosa succede agli oggetti e ai relativi link a seguito delle 4 operazioni successive (**non** eseguite in sequenza, ma eseguite tutte a partire dalla situazione iniziale indicata): `cp s1 s2`, `cp h1 h2`, `rm s`, `rm h`.

directory block: blocco per directory entry

directory entry: filename + #i-node (≡ hard-link)

data blocco : blocco per dati file

i-node : 1 ∀ file ; include ↑ blocchi dati e altre informazioni

NO NO NO

permessi : - XXX YYY ZZZ RWX

OWNER GROUP OTHER

chmod xxx file

chmod u±w file

chown nome file

chgrp nome file

NO NO NO

soft link: file che contiene puntatore a file originario

hard link: puntatore dalla directory entry all'i-node

ln [-s] file [link]

↳ soft-link

NO NO NO

cp s1 s2	copia il file s in s2
cp h1 h2	copia il file h1 ≡ h in h2
rm s	s1 rimane come link pendente del file h
rm h	h1 rimane come copia del file h