Reserved cells

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

Operationg Systems

Examination task

05 September 2019

ID number ________________ Surname _________________________ Name _________________________

Professor: ○ Scanzio

It is not possible to consult texts, notes or to use calculators. The only material allowed consists in the forms distributed by the professor. Solve the exercises in the reserved spaces. Additional sheets are permitted only when strictly necessary. Report the main steps for solving exercises.
Duration: 100 **minutes.**

1. Specify the meaning of *Process Control Block* and of *Context Switching*.

   Represent and describe the state diagram of a process. Also indicate the meaning of "orphan process" and "zombie process".

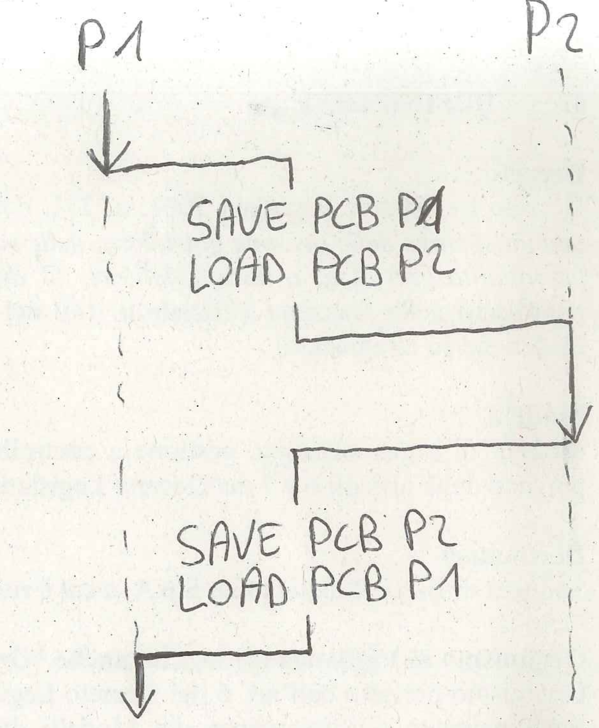
   Finally, reports two *precedence graphs* that make use of the system calls `fork` and `wait`, one must be realizable, the other not realizable.

Es 1

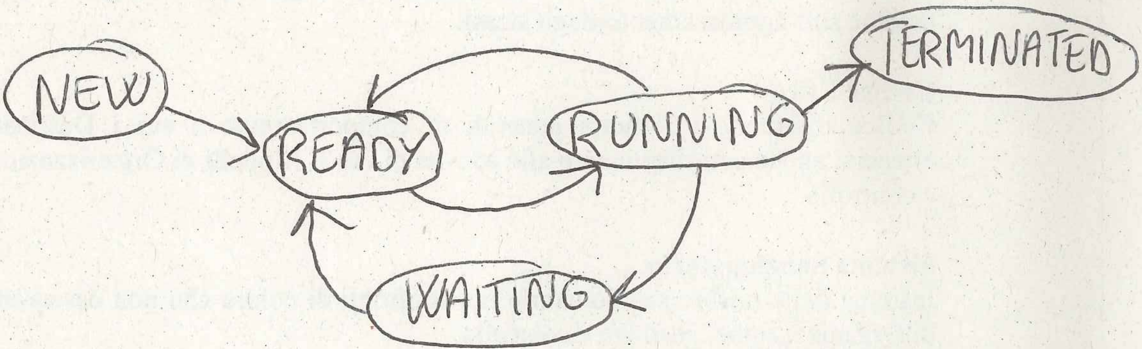
PCB

↑ PROCESS STATE
PID
PC (PROGRAM COUNTER)
REGISTERS
MEMORY LIMITS
LIST OF OPEN FILES
...

CONTEXT SWITCHING



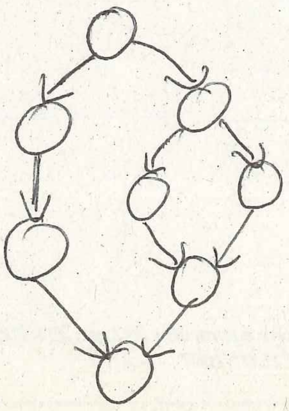
STATE DIAGRAM



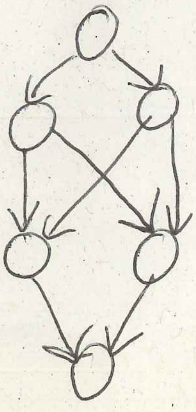
ORPHAN: PARENT TERMINATES, PROCESS INHERITHED BY INIT

ZOMBIE: ~~PARENT TERMINATES~~, PROCESS TERMINATED, PARENT HAS NOT DONE THE WAIT() YET

REALIZABLE GRAPH



NOT REALIZABLE GRAPH



2. Suppose to execute the following program

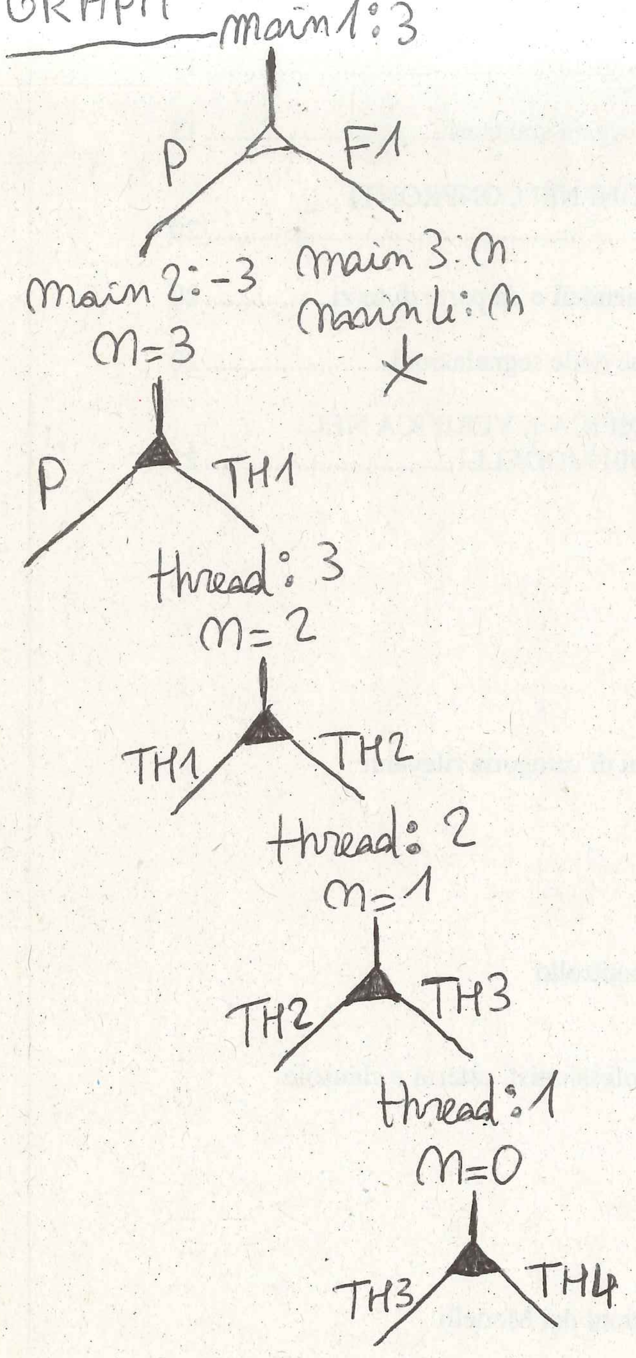
```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
void *t1 (void *p){
  pthread_t thread;
  int *pn = (int *) p;
  int n = *pn;
  if (n>0) {
    printf ("thread: %d\n", n--);
    pthread_create (&thread, NULL, t1, &n);
  }
  pthread_join (thread, 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;
}
```

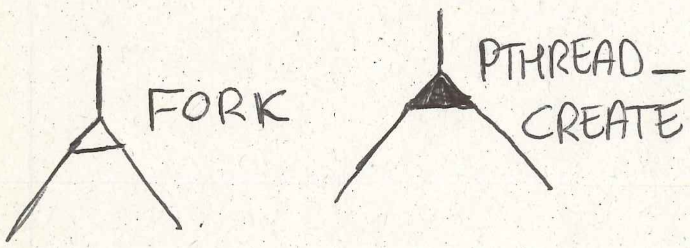
with only one integer parameter equal to 3.

Report the *control flow graph (CFG)* and the *process generation tree* after its execution. Also indicate what it produces on video and for what reason.

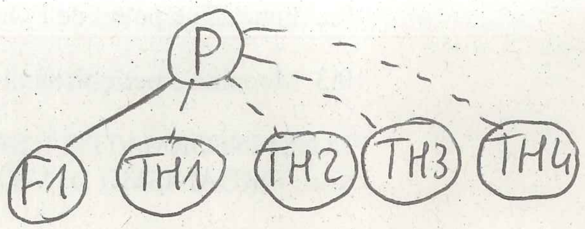
Es 2

CONTROL FLOW GRAPH





PROCESS GENERATION TREE



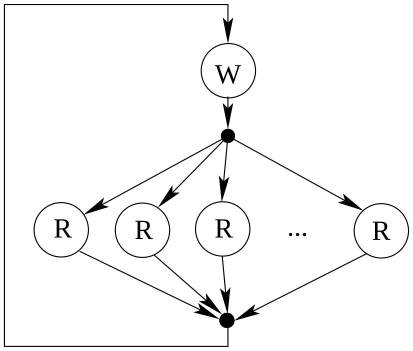
POSSIBLE OUTPUT

(Real output depends on the scheduler)

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

3. Illustrate the *Readers and Writers* problem. Report its solution using semaphore primitives in the case of precedence to Readers (illustrate the meaning of each semaphore). What do "precedenza ai Readers" mean?

   Starting from the solution of the previous problem, realize the synchronization scheme represented by the following *precedence graph*.



The number of Readers is unknown

   Cyclically, the execution of one Writer is followed by the execution of an indefinite number of Readers. At the beginning, it is necessary to run a Writer. At the end of the Writer, it is needed to execute one or more Readers. However, when the last Reader exits the critical section, before another Reader enters, it is necessary that a new Writer is executed.

Es 3

1w → mR → 1W → mR...

R

```
wait(meR);
  mR++;
signal(meR);

...

wait(meR);
  mR--;
  if(mR==0) signal(meW);
  else signal(meR);
```

W

```
wait(meW);

...

signal(meR);
```

```
mR = 0;
init(meR, 0)
init(meW, 1);
```

NOT CORRECT SOLUTION

mW → mR → mW → ..

```
wait(meR);
  mR++;
signal(meR);

...

wait(meR);
  mR--;
  if(mR==0) signal(meW);
  else signal(meR);
```

```
mR = mW = 0;
init(meR, 0);
init(meW, 1);
init(w, 1);
```

```
wait(meW);
  mW++;
signal(meW);
wait(w);

...

signal(w);
wait(meW);
  mW--;
  if(mW==0) signal(meR);
  else signal(meW);
```

POSSIBILITY TO HAVE MORE THAN ONE WRITER

4. A textual file with name `virus.dat`, stores a list of PID on subsequent lines.

   Realize a BASH script that verifies which of the processes listed in the file `virus.dat` are running in the system. For each of these processes in execution, the script has to visualize the name of the owner, and the list of the PID of all its children processes.

   Remember that the command `ps -ef` provides an output similar to the following:

```
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

#################################################################################################
# Exercise 4 - Exam
05/09/2019                                                                    #
#   Launch with: ./es4.sh
virus.dat                                                               #
#################################################################################################

# 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. Write a multi-thread program able to read and write a matrix of integer numbers as described in the following.

   The matrix, statically defined with `R` rows and `C` columns, has to be firstly read from standard input and then write on standard output. Both operations have to be performed on a specific order, from row number `0` to row number `R-1`, from column `0` to column `C-1`.

   The operations of read and write must be performed by the program using a single function thread that reads, and using a single function thread that writes, both executed `R` times. At the beginning, the program executes the `R` threads that read, which synchronize to read the matrix in the indicated order (the first thread reads the first row and the others wait, then the second thread reads the second row and the other wait, etc.). Each thread reads the whole line. After reading, the program executes the `R` threads that write, which synchronize with each other and write the matrix in the indicated order. Each thread writes the whole row. The program then ends.

```
#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: 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;
}
```

6. In reference to the *indexed allocation* in UNIX/LINUX, indicate the meaning of the following terms (possibly using appropriate graphic aids): *directory block*, *directory entry*, *data block* and *i-node*.

   Specify the meaning of "*permissions*" (or rights), "*owner*" and "*group*" of a file or a directory in the UNIX system.

   How can you modify the "*permissions*", the "*owner*" and the "*group*" of a file or a directory?

   Clarify the meaning of *soft-link* and *hard-link*, reporting the commands to create them.

   Let `s1` be a soft-link to the file `s`, `h1` a hard-link to the file `h`, and `d` a directory.

   In this circumstance, indicate what happens to the objects and to the links following the 6 following operations, which are (**not**) executed in sequence, but all performed starting from the initial state previously indicated): `cp s1 s2`, `cp h1 h2`, `rm s`, `rm h`, `mkdir d/d1`, `mkdir d/d2`.

Es. 6

DIRECTORY BLOCK: blocco contenente le directory entries

DIRECTORY ENTRY: FILENAME + #I-NODE (≡HARD-LINK)

DATA BLOCK: Block for file data

I-NODE: one i-node for every file. It includes a pointer to the data block and other informations

PERMISSIONS: $- \underbrace{\overset{RWX}{XXX}}_{OWNER}\ \underbrace{\overset{RWX}{YYY}}_{GROUP}\ \underbrace{\overset{RWX}{ZZZ}}_{OTHER}$

```
chmod xxx file          chmod u±w file
chown name file         chgrp name file
```

SOFT LINK: file the contains a pointer (in particular the name) to a file

HARD LINK: Pointer from the directory entry to the i-node

```
ln [-s] file [link]
    ↑ soft-link
```

cp s1 s2	copies file s1 in s2
cp h1 h2	copies file h1≡h in h2
rm s	s1 remains as a pending link of the file h
rm h	h1 remains as copy of the file h