Real Time Programming
TD+TP = synchronizing threads with semaphores

A - General instructions

Exercises realization:

• Programs must be in C language and run on linux.
• Favor a clear and readable style
• Name functions and variables with appropriate names
• Comment on your code
• Look after your indentation
• **Each exercise must be the subject of a separate C program (.c file).**
• **The answers to the questions should be placed as comments at the beginning of the file.**
(containing the hand) of the corresponding exercise.

Delivery of exercises:

• Each file must be named with the exercise number (ex: "exo1" or "exercise1").
• **Send your work by email to** roula.osta@gmail.com or to easyclass.com!
• **Indicate [PTR] at the beginning of the subject (with the brackets).**
• **Indicate your name and surname! (Otherwise I do not know who to give the note!)**

B - Reminders

Use the "man" command or look in the manual on the internet to learn more about a function.

```
// headers to import
#include <pthread.h> // for threads
#include <semaphore.h> // for semaphores
#include <fcntl.h> // for flags O_CREAT, O_EXCL, ...
// anonymous semaphore, not persistent ("deprecated" under MacOS)
int sem_init (sem_t * sem, int pshared, unsigned int value);
int sem_destroy (sem_t * sem);
// named and persistent semaphore created in the / dev / shm directory under linux
sem_t * sem_open (char const * name, int oflag, ...);
// oflag = list of flag separated by logical ORs
// O_CREAT = create the semaphore if it does not already exist
// O_CREAT | O_EXCL = error if the semaphore already exists
// 2 optional arguments:
//
1 - the rights = S_IRUSR | S_IWUSR
//
(right of reading and writing for the user)
//
2 - the initial value of the semaphore
int sem_close (sem_t * sem); // report that the semaphore is no longer useful
int sem_unlink (const char * name); // destroy the semaphore
// for debugging = display the value of the semaphore
int sem_getvalue (sem_t * sem, int * sval);
// to lock / unlock a semaphore (anonymous or not)
int sem_wait (sem_t * sem); // primitive P ()
```

int sem_post (sem_t * sem); // primitive V ()
// other functions to request the semaphore
int sem_trywait (sem_t * sem);
int sem_timedwait (sem_t * sem, const struct timespec * abs_timeout);
// Compile your code with the -Wall and -pthread options:
$ gcc -pthread –o exe exe.c
Compilation should not return error or warning.

C - Exercise Statements

Exercise 1 - unsynchronized threads

Write a program with 1 main thread, the heavy process, displaying A on standard output and 2 threads show R and G respectively on the standard output. Each thread performs 100 iterations. The Expected result is a set of 300 letters grouping 100 A, 100 R and 100 G without notion of order.
Note 1: You can reuse the given code in progress as a starting point.
Note 2: For a concise display, use printf () without wrapping and use fflush (stdout) to force the buffer contents to appear (without waiting for a newline or the buffer is full).

Exercise 2 - synchronization with anonymous semaphores

Make a copy of the previous file and use an anonymous semaphore to synchronize the threads and display letters in an ordered sequence (ARG) *

Exercise 3 - synchronization with named semaphores

Make a copy of the previous file and now use a named semaphore to get the same result.
Question: What happens if you omit sem_unlink () at the end of the program? Describe your observations and explain them.
To answer, comment out the lines calling sem_unlink (), re-compile, start your program and check the contents of the directory / dev / shm / in both cases (presence or absence of sem_unlink ()).

Exercise 4 - persistent semaphore

Make a copy of the code written for question 3 that **does not use sem_unlink ()** and modify your program.
Ensure that semaphores are created only at the first run of the program and then reused (we test if they already exist or we create them and initialize them to 0). Add a call to sem_post ()
to unblock a semaphore and start writing the first Y.
Question: What happens when you restart the program several times? Describe and explain the behavior of your program.

Exercise 5 - more general program (to give as graded assignment)

Create a program that receives an N number as an argument and starts N threads each displaying one of the numbers from 1 to N synchronously (with named or anonymous semaphores of choice) to produce a ordered sequence (12 ... N) *. Here, it is necessary to use one and the same function executed by all threads and pass the number to be displayed as an argument when starting the thread.

Exercise 6 - Synchronization with semaphores
From a copy of the code written for question 2 or 3, add a thread displaying H on the standard output.
Use anonymous or named semaphores (depending on your preference) to synchronize threads and display the letters in an ordered sequence (ARRGHHH) *, the threads displaying respectively 100 A, 200 R, 100 G and 300 H.

Exercise 7 - (optional) semaphores and heavy processes
Repeat exercise 5 with fork and heavy processes. Attention, only named semaphores can be shared simply between heavy processes. Recall on the use of fork:

```
#include <unistd.h>
pid = fork ();
// the heavy process is duplicated
if (pid> 0) {
// the father process will execute this code
int status;
waitpid (pid, & status, 0);
// the parent is waiting for the end of his child
} else if (pid == 0) {
// the child process will execute this code
} else {
// this code will be executed if fork returns an error
}
```