TP 10 : Threads and semaphores

1 Basics

How to compile:

```
gcc -pthread program.c -o program
```

How to declare, create and join threads:

```
pthread_t t1;
```

pthread_create takes as arguments a pointer to the thread ID, attributes to set the properties of thread, a function pointer to the function that thread will run in parallel on start (this function should accept a void * and return void * too) and arguments to be passed to the function. For instance :

```
pthread_create(&t1, NULL, &function, NULL);
```

Don't forget to join the thread afterward:

```
pthread_join(t1, NULL);
```

The second argument is a pointer to store the return value from the thread. pthread_create and pthread_join both return 0 if the thread has been created/joined successfully.

How to declare, initialize, destroy and use semaphores:

```
sem_t s1;
sem_open(&s1, 0, n);
sem_destroy(&s1);
```

sem_open takes as second argument 0 because the semaphore is shared between threads and not processes. The argument n means that sem_wait can be called n times until the semaphore is locked. For intance if $n = 0$ it means a sem_post must be used first. sem_open and sem_destroy both return 0 if the semaphore has been opened/destroyed successfully.

```
sem_wait(&s1);
\\ critical code section
sem_post(&s1);;
```

sem_post frees the semaphore (i.e., increases n), sem_wait waits for the semaphore to be free ($n > 0$) and decreases n.

See `sem.c` as an example.

How to declare, initialize, destroy and use mutexes:

```
pthread_mutex_t m1;
pthread_mutex_init(&m1, NULL);
pthread_mutex_destroy(&m1);
pthread_mutex_lock(&m1);
pthread_mutex_unlock(&m1);
```

2 Shared data conflict

1. What is the main difference between threads and processes? Can we use them interchangeably?

2. Download the program `mailbox.c` from the website. You have seen this example in class. What happens when the value of MAX is 10? 100? 1000? 1000000? Why does it not work as planned? Can you fix it?

   Note: Observe that inside the `for` *loop, we have only one line of code, i.e.* `mails++;` *Why still do we have a race condition?*

2. Now, write a program in C with two threads. The first thread displays even integers up to 100, the second the odd. Use semaphores to ensure the correct order of integers.

3 Producer-Consumer

A typical problem in concurrent programming is that of producer and consumer: one thread produces data that the other consumes. We consider the `procon.c` program. Here the producer gets characters from a file and sends them to a shared space with the thread 'consumer' who displays them on the screen. However, in the current version, no synchronization exists in between. Modify the program so that it displays the contents of a file correctly.

4 Binary Semaphores vs. Mutexes

Download the program `mutexvsem.c`. Run it. Do you see any errors? Now, change the mutexes and have it use semaphores instead. Does it work?

Can you justify your observation?

5 Counting semaphores

Implement a logging queue using semaphores. You can use the template given in `login.c`. Your objective is to ensure that the resource (in our case the section enclosed in 'finite resource') is used by at most 12 users at a time. Create 16 threads and use a semaphore to manage the resource.