RTOS, Spring 2015 – Lab #4: Process synchronization

Paolo Torroni, paolo.torroni@unibo.it
Davide Chiaravalli, davide.chiaravalli@studio.unibo.it

Objective: to learn the basics of programming concurrent threads

1. Background

Make sure you have read and understood Chapter 6 from the OS textbook ("Synchronization").

Pthreads provides various tools for synchronizing processes. These include

- **Mutex locks**
- **Condition variables**
- **Semaphores** (via the `sem.h` library)

The steps for using a mutex lock are:

1. Definition of lock (`pthread_mutex_t lock;`)
2. Initialization of lock (`pthread_mutex_init(&lock,NULL);`)
3. Use of lock to synchronize thread execution
   (`pthread_mutex_lock(&lock);`
   `pthread_mutex_unlock(&lock);`)
4. Dismissal of lock (`pthread_mutex_destroy(&lock);`)

The definition of a variable (say, `lock`) of type `pthread_mutex_t` creates a mutex lock, but before `lock` can be used, it must be initialized using function `pthread_mutex_init(&lock, attr)`. This function initialises `lock` with attributes specified by `attr`. If `attr` is `NULL`, default attributes are used (it's OK to leave it `NULL` if you don't have to deal with **priority inversion**).

Notice that `lock` is **initially unlocked**.

Remember to always pass the mutex to the above functions **by address** (use `&lock`, not simply `lock`).

As soon as you don't need the mutex any longer, you should destroy it to free the resources it uses. Be sure to destroy it when it is **unlocked**.

2. Synchronization using mutex locks

A) Consider the sample thread program below, where a parent thread creates two children threads, and each thread prints out a "Hello World" message. Build. Execute repeatedly. Consider the output: is it always in the same order?
B) Use mutex locks to ensure that the always shows in the order:

First Hello World
Second Hello World
Third Hello World
Last Hello World

```c
/**
 * Sample thread program
 */

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

void *f_a(void*arg) {
	puts("!!!First Hello World!!!");
	return NULL;
}

void *f_b(void*arg) {
	puts("!!!Second Hello World!!!");
	return NULL;
}

int main(void) {
	pthread_t thread_a, thread_b;
	pthread_create(&thread_a, NULL, f_a, NULL);
	pthread_create(&thread_b, NULL, f_b, NULL);

	puts("!!!Third Hello World!!!");

	pthread_join(thread_a,NULL);
	pthread_join(thread_b,NULL);

	puts("!!!Last Hello World!!!");
	return EXIT_SUCCESS;
}
```

Note: you can download the file lab4-mutex-hw.c from http://lia.deis.unibo.it/Courses/RTOS/

3. Rendez-vous

A) Consider the program below. Use mutex locks to ensure that

Third Hello World

and

Fourth Hello World

always come after **both**

First Hello World

and

Second Hello World

have been printed out (the relative order between Third and Fourth and between First and Second is irrelevant).

```c
/**
 * Sample thread program
 */

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

void *f_a(void*arg) {
	puts("!!!First Hello World!!!");
	puts("!!!Third Hello World!!!");
	return NULL;
}
```

```
void *f_b(void*arg) {
	puts("!!!Second Hello World!!!");
	puts("!!!Fourth Hello World!!!");
	return NULL;
}

int main(void) {
	pthread_t thread_a, thread_b;
	pthread_create(&thread_a, NULL, f_a, NULL);
	pthread_create(&thread_b, NULL, f_b, NULL);

	pthread_join(thread_a,NULL);
	pthread_join(thread_b,NULL);

	return EXIT_SUCCESS;
}
```

Note: you can download the file lab4-mutex-rendezvous.c from http://lia.deis.unibo.it/Courses/RTOS/

4. Readers-Writers

A) Implement a program with 4 threads (2 readers and 2 writers). You can start from the sample code in lab4-readers-writers.c from the lab page http://lia.deis.unibo.it/Courses/RTOS/ but then **you must fix it**, to make sure that synchronization is correct.
B) Execute the program. Observe if there is risk of **starvation**.

6. Self assessment

- ☐ How can I secure exclusive access to a critical section?
- ☐ As soon as a mutex lock is initialized, is it open or closed?
- ☐ How can I implement a rendez-vous?
- ☐ How does a rendez-vous differ from a "before" type synchronization?
- ☐ Is there a possible situation of starvation in the solution we have seen of the Readers-Writers problem?

Notice that, to correctly use Pthreads (with Eclipse), you should:
1) #include <pthread.h>
2) add the -pthread option both in the GCC C Compiler and in the GCC C Linker:
->Project->Properties->C/C++ Build->Settings->Tool Settings
->GCC C Compiler->Miscellaneous-> Other flags: add **-pthread**
->Project->Properties->C/C++ Build->Settings->Tool Settings
->GCC C Linker->Miscellaneous-> Linker flags: add **-pthread**