Lab1 : Threads in POSIX systems

1 Generalities

1.1 Introduction

Threads are a fundamental concept in computer programming that allows for parallel execution of different parts of a program. In real-time programming, threads are especially important because they allow a program to perform multiple tasks simultaneously.

A thread is essentially a separate sequence of instructions that can be executed independently of the main program. In real-time programming, a thread can be used to handle tasks that require immediate attention or that need to be executed quickly. For example, in a system that controls a manufacturing process, a thread might be used to monitor temperature or pressure readings and make adjustments in real-time to ensure that the process stays within safe limits.

One of the challenges of real-time programming is ensuring that different threads execute in a predictable and deterministic way while respecting their temporal constraints. That's why threads may be assigned different priorities so that critical tasks can be given higher priority than less important ones. Real-time scheduling allows the system to prioritize threads and allocate CPU time to them based on their importance and time sensitivity. The scheduler ensures that defined high-priority threads are executed first.

During this lab, we implement some uniprocessor scheduling policies for POSIX threads. Specifically, we explore the SCHED_FIFO and SCHED_RR policies.

Requirements : knowledge of C language + machine with GCC compiler

1.2 POSIX threads

In C language, POSIX threads are manipulated using `Pthreads` library. Here are some common functions that can be used to manipulate threads :

- `pthrad_create()`: creates a new thread and assigns a unique thread ID to it.
- `pthread_join()`: waits for a thread (passed in argument) to terminate and returns the exit status of the thread
- `pthread_exit()`: terminates the calling thread and returns a value to the parent thread

Please check more details about the syntax of functions that manipulate threads in Pthread library or by typing `$man pthreads` in the terminal if you're a Linux user

2 Getting started exercise

This first exercise is about managing the creation and termination of threads.

Download the file `thread_ex.c` from the course directory and run it using the provided compilation command (for Linux users). In this code, there is a main process implemented in the (`main()`) function. In this process, 3 threads are created. Their function is implemented in `thread_function()`.

Q1- In the provided code, identify the variable that informs about each thread's identifier and describe what a thread does when it is executed (without running the program). Do you think that `thread_function()` is related to only one thread or to the all of the 3 threads ?

Q2- Compile and run the program, what do you notice ?

The 3 created threads belong to the same process that runs the `main` function.

Q3- To verify this, call `getpid()` in the thread function and display its return. What do you notice ?
NB : you can refer to documentation of `getpid()` or type `$man getpid` in the terminal if you're a Linux user

Q4- Create a C file in which you implement your main process (eg. `main.c`). Include in your file `<pthread.h>`, `<stdio.h>`, `<stdlib.h>` and `<time.h>` libraries

Q5- Implement a thread function that runs for a number of seconds passed in arguments. See example in the listing below

```
void *task_func(void *arg)
{
        int nb_sec = (int) arg;
        printf("This thread is executing for %d seconds\n", nb_sec);
        sleep(nb_sec);
        printf("This thread is terminating\n");
        pthread_exit(NULL);
}
```

Q6- In your main function, declare a variable of type `pthread_t` for storing a thread's identifier and another integer variable for function returns. Then create your thread, pass its function and the number of seconds in arguments and wait for its termination before exiting the main process

3 Basic scheduling in POSIX

This is a basic exercise about scheduling a number of threads. Download and extract files from `fifo_rr_sched` folder, analyze the code in each file and explain its functioning and the role of the following :

— `thread_function()`

— `threads_table`

Q1- Run the code using the `Makefile` or using the GCC compiler in your environment. What do you notice about the execution periodicity and scheduling of threads ?

In fact, scheduling policy used in this example is the default scheduling policy of the pthreads library, which is implementation-defined and can vary depending on the operating system and system configuration. The default scheduling policy typically uses a time-sharing algorithm to schedule threads, where each thread is given a slice of CPU time to execute before being preempted and another thread is scheduled to run. However, the exact behavior of the default scheduling policy may differ between systems.

It is possible to implement its own scheduling policy. To do so, you need to define a set of scheduling functions and data structures that implement the scheduling policy to be used. `Pthread` library along with `Sched` library offer several functions that help specifying a customized scheduling policy

The `Sched` library defines the `sched_param` structure which stores information about thread's scheduling parameters in two fields :

— `priority` from 1 to 99

— `policy` : `SCHED_FIFO`, `SCHED_RR` and `SCHED_OTHER` (for implementation specific policy)

This structure is typically used in conjunction with `sched_setscheduler()` and `sched_getscheduler()` functions may be called in the program of the affected thread

Please refer to the Sched library documentation for more details and analyze what the aforementioned functions do.

Q2- Use the previous example and specify in the threads function that threads are to use `SCHED_FIFO`. Compile and execute your example, what do you notice ?

Q3- Use the same example and specify in the threads function that threads are to use `SCHED_RR`. Compile and execute your example, what do you notice ?