Introduction to Threads

Objectives

Acquaint yourselves with multi-threaded programs and parallel execution

Aim

The aim of this session is to familiarize yourselves with multithreaded programs. The tasks of this lab session will be run on Linux.

How to Compile and run a C program in Linux.

In this lab session we will be using gedit text editor to write our programs and the Linux Terminal to compile and run our programs. Please note that there are more efficient programming environments to develop software such as Eclipse (it is free), but for small programs, this is not necessary.

You can compile a .c file using the following command:

```
gcc source.c -o exec -other_options
```

gcc is a well-known and used compiler. The executable name is specified just after the '-o' ('o' stands for output). *'-other_options'* refers to other compilation options, e.g., '-lm' will include the 'math.h' library.

Then, you can run the executable by using the following command:

```
./exec
```

POSIX Pthreads

Pthreads refer to the POSIX standard defining an API for thread creation and synchronization. Thread operations include thread creation, termination, synchronization, scheduling, data management and process interaction. All threads within a process share the same address space and have the same process ID. Each thread has its own thread ID.

Pthreads are created using '*phtread_create()*' which is shown below. If successful, this function returns zero. Otherwise, an error number is returned to indicate the error.

*int **pthread_create** (pthread_t * thread, const pthread_attr_t * attr, void * (*start_routine)(void *), void *arg);*

- thread – this is the thread id (unsigned long int)
- attr - Set to NULL (not discussed in this module)
- void * (*start_routine) - pointer to the function to be threaded. Function has a single argument: pointer to void (in other words pointer to anything).

- *arg - pointer to argument of function. To pass multiple arguments, send a pointer to a structure.

Given that the 'phtread_create()' input operands are using pointers, the input parameters will always be given to you, so don't worry.

Task1. Study the 'pthreads_introduction.c' program. **Read carefully all the code comments to understand how this program works**.

In this example, four threads are created and they all run their own copy of function '*print_message_function()*'. This function takes as input a pointer to void (pointer to everything). It returns a pointer to void too. A void pointer is a pointer that has no associated data type with it. A void pointer can hold address of any type and can be typecasted to any type.

By calling Pthread_join( ) command, the program will wait there until the thread finishes.

Run this program many times. You will realize that the order of the running threads is not the same. Make sure you understand that each thread has its own thread ID, but a common process ID. The output of the program should be

```
Thread 2  has ID 139796146226944 --- The Process ID is 7551

Thread 1  has ID 139796154619648 --- The Process ID is 7551

Thread 3  has ID 139796137834240 --- The Process ID is 7551

Thread 4  has ID 139795995223808 --- The Process ID is 7551


Thread 1 with ID= 139796154619648 completed, returned: 0

Thread 2 with ID= 139796146226944 completed, returned: 0

Thread 3 with ID= 139796137834240 completed, returned: 0

Thread 4 with ID= 139795995223808 completed, returned: 0
```

Task2. Try to run the previous program without using the join() function. What do you observe?

Use Pthreads to parallelize a simple program

In this example, we will use pthreads in order to parallelize two simple programs and run them on multiple CPU cores. In sqrt.c and MVM_pthreads.c are the parallel versions of the sqrt() and MVM() programs, respectively. **Read carefully all the code comments to understand how they work**. These two programs and their solutions are further explained in the slides. An OpenMP implementation is also provided on github.

Task3. Run just the MVM_pthreads.c program and measure the execution time for different number of threads and different input size (N). Make sure you understand the purpose of starting_row and ending_row variables. Make the graph of speedup achieved vs number of threads, for N=[100, 500, 1000, 2000, 4000, 8000, 16.000] and NUM_THREADS=[1, 2, 4, 8]. You will observe that for small input sizes, the program does not scale well. This is because the overhead of creating and synchronizing the threads is comparable to the thread’s execution time. Compare your results with the ones shown in the slides.

Further Reading

1. POSIX thread (pthread) libraries, available at https://www.cs.cmu.edu/afs/cs/academic/class/15492-f07/www/pthreads.html
2. POSIX Threads Programming, available at https://computing.llnl.gov/tutorials/pthreads/