6G7Z1004 Advanced Computer Networks and Operating Systems
(Lab 3: Concurrency: threads & synchronization)

A part of this lab was adapted from a lab originally designed by Emma Norling, MMU, UK
Other sources: Kai Hwang et al, "Distributed and Cloud Computing: From Parallel Processing to the Internet of Things"

The aim of this lab session is to discover the different thread libraries in use today and learn how to create and manage threads in POSIX standard and JVM (Java Virtual Machine).

An overview on Thread Libraries

A thread library provides the programmer with an API for creating and managing threads. There are two primary ways of implementing a thread library. The first approach is to provide a library entirely in user space with no kernel support. All code and data structures for the library exist in user space. This means that invoking a function in the library results in a local function call in user space and not a system call.

The second approach is to implement a kernel-level library supported directly by the operating system. In this case, code and data structures for the library exist in kernel space. Invoking a function in the API for the library typically results in a system call to the kernel.

Three main thread libraries are in use today:

1. **POSIX (Portable Operating System Interface) pthreads**: for more details about this library type **man pthreads** or visit this link
2. **Windows** (check this link for more details)
3. **Java**: more details can be found here

Pthreads, the threads extension of the POSIX standard, may be provided as either a user-level or a kernel-level library. The Windows thread library is a kernel-level library available on Windows systems. The Java thread API allows threads to be created and managed directly in Java programs. However, because in most instances the JVM (Java Virtual Machine) is running on top of a host operating system, the Java thread API is generally implemented using a thread library available on the host system. This means that on Windows systems, Java threads are typically implemented using the Windows API; UNIX and Linux systems often use Pthreads.

Task A: Pthreads

Pthreads refers to the POSIX standard (IEEE 1003.1c) defining an API for thread creation and synchronization. This is a specification for thread behavior, not an implementation. Operating-system designers may implement the specification in any way they wish. Numerous systems implement the Pthreads specification; most are UNIX-type systems, including Linux, Mac OS X, and Solaris. Although Windows does not support Pthreads natively, some third-party implementations for Windows are available.

The C program shown below demonstrates the basic Pthreads API for constructing a multithreaded program that calculates the summation of a non-negative integer in a separate thread. To run this program do the following:
Use **gedit** to create a file named threads1-posix.c

gedit threads1-posix.c &

(Remember, the "&" sign allows the Terminal to accept commands while gedit is still open, which is convenient.)

and add the following code

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

int sum; /* this data is shared by the thread(s) */
void *runner(void *param); /* threads call this function */
int main(int argc, char *argv[])
{
pthread_t tid; /* the thread identifier */
pthread_attr_t attr; /* set of thread attributes */
if (argc != 2)
{
fprintf(stderr,"usage: a.out <integer value>\n");
return -1;
}
if (atoi(argv[1]) < 0)
{
fprintf(stderr,"%d must be >= 0\n",atoi(argv[1]));
return -1;
}

/* get the default attributes */
pthread_attr_init(&attr);

/* create the thread */
pthread_create(&tid,&attr,runner,argv[1]);

/* wait for the thread to exit */
pthread_join(tid,NULL);
printf("sum = %d\n",sum);
}
```

```
/* The thread will begin control in this function */
void *runner(void *param)
{
int i, upper = atoi(param);
sum = 0;

for (i = 1; i <= upper; i++)
sum += i;

pthread_exit(0);
}
```

Compile the file threads1-posix.c using the command

gcc –pthread -o threads1 threads1-posix.c

Now an executable file named threads1 should appear in the current directory. Type **ls** to see the new file. You can now run it by typing

./threads1 3

Download the file threads2-posix.c from the lab material folder on Moodle and examine its source code; try to determine what the program does. Compile the program by typing
gcc -pthread -o threads2 threads2-posix.c

Run it by typing

./threads2 4

What happens if you make the number 4 a large integer (e.g. 50, 100, 500)?
Does the program still work correctly?
How does the program test for thread termination? What other pthread functions could be used to test for termination, and or to synchronise the operation of the threads.

Task B: Threads in Java

In this section, we will learn how to create and manage Threads in Java. Before you examine, compile and run the files that you download from the lab material folder on Moodle, you should examine Java API information on the Thread class, and the Oracle tutorial on Java concurrency.
Download and compile the Data.java file to bytecode by typing
javac Data.java

Check the new generated files, what is their extension (.out or .class)? Are they similar to the files generated in Task A when you use gcc?

Run the generated file by typing:
java data
To learn more about javac and java type **man javac** or **man java**

Examine the source code. How does it create threads? What is the body or method of code that the thread executes? How do the threads cooperate to solve a problem, do they share memory or exchange messages?

Experiment with a different number of total threads – what happens? The Data.java program has a deliberate but very common concurrent programming error. Can you explain the nature of the error? HINT: if you are stuck then read the program source again! Can you think of a programming constructs in Java that could avoid this deliberate error?

Download, compile and run the Driver.java program. What does it calculate? Examine the source code. How does it create threads?
Compare Data.java with threads2-posix.c and Driver.java with threads1-posix.c, do they perform the same task?

Exercise:

Implement the echo() function discussed in the Concurrency lecture (see slides 31 and 32) in C or java such that you create two threads that execute the same echo() function. To prevent the race conditions, you need to use one of the mechanisms discussed in the Concurrency lecture to protect the Critical Section of your program.