OBL2-OS

October 25, 2018

This is a mandatory assignment. Use resources from the course to answer the following questions. Take care to follow the numbering structure of the assignment in your submission. Some questions may require a little bit of web searching. Some questions require you to have access to a Linux machine, for example running natively or virtually on your own PC, or by connecting to gremlin.stud.iie.ntnu.no over SSH (Secure Shell). Working in groups is permitted, but submissions must be individual.

**Processes and threads**

1. Explain the difference between a process and a thread.

The difference between a process and a thread is that a process we need to create an entire address space while with threads we use the address space of an already existing process.

1. Describe a scenario where it is desirable to:

• Write a program that uses threads for parallel processing

It is desirable to do this when you have a program built up by many small tasks, because you can create a thread for each task for an efficient execution by utilizing concurrency.

• Write a program that uses processes for parallel processing

It is desirable ff you want to write a program where you want to it to several

things at ones, but you don’t want the process to have access to all the information like a thread would have. For example, when you have tasks that cover entire applications.

1. Explain why each thread requires a thread control block (TCB).

Since the threads exist in different states, they need their own control block to keep track of those states.

1. What is the difference between cooperative (voluntary) threading and pre-emptive

(involuntary) threading? Briefly describe the steps necessary for a context switch for each case.

Cooperative threading means you have to ask the thread to switch to a different thread or state, while pre-emptive threading is when the OS decides what threads to switch to for maximum efficiency. For the cooperative threading you have to exit the thread then initialize another thread, while the OS takes care of that when you have pre-emptive threading.

**2. C program with POSIX threads**

\*nix operating systems use POSIX threads, which are provided by the pthread library. Consider the following adapted code from the textbook (the code has been modified slightly to use pthread, while the book assumes its own thread implementation).

#include <stdio.h>

#include <pthread.h>

#define NTHREADS 10

pthread\_t threads[NTHREADS];

void \*go (void \*n) {

printf("Hello from thread %ld\n", (long)n);

pthread\_exit(100 + n);

// REACHED?

}

int main() {

long i;

for (i = 0; i < NTHREADS; i++) pthread\_create(&threads[i], NULL, go, (void\*)i);

for (i = 0; i < NTHREADS; i++) {

long exitValue;

pthread\_join(threads[i], (void\*)&exitValue);

printf("Thread %ld returned with %ld\n", i, exitValue);

}

printf("Main thread done.\n");

return 0;

}

We can compile the code and tell the compiler to link the pthread library:

$ gcc -o threadHello threadHello.c -lpthread

At the command prompt, run the program using ./threadHello. The program gives output similar to the following:

Hello from thread 0

Hello from thread 3

Hello from thread 5

Hello from thread 1

Hello from thread 4

Thread 0 returned with 100

Thread 1 returned with 101

Hello from thread 9

Hello from thread 8

Hello from thread 2

Hello from thread 7

Hello from thread 6

Thread 2 returned with 102

Thread 3 returned with 103

Thread 4 returned with 104

Thread 5 returned with 105

Thread 6 returned with 106

Thread 7 returned with 107

Thread 8 returned with 108

Thread 9 returned with 109

Main thread done.

Study the code and the output. Run the code several times. Answer the following questions.

1. Which part of the code (e.g., the task) is executed when a thread runs? Identify the

function and describe briefly what it does.

The task go is executed when a thread runs, this will print “Hello from thread” then add a exitValue to the thread:

void \*go (void \*n) {

printf("Hello from thread %ld\n", (long)n);

pthread\_exit(100 + n);

1. Why does the order of the “Hello from thread X” messages change each time you run the program?

The order changes because the threads in the first for-loop are unpredictably scheduled, which means the threads are running at unpredictable speeds, so we can´t see how they´re scheduled, unlike the second for-loop where there is a sequence of instructions which makes it deterministic, therefor this will print in order.

1. What is the minimum and maximum number of threads that could exist when thread 8 prints “Hello”?

The maximum number of threads called when thread 8 prints “Hallo” will be the 10 threads you create in the first for-loop including number 8, and the main thread to 8, so that’s a total of 11 threads. The minimum number of threads existing would be 2, the thread for 8 and the main thread for 8, since this could happen before any other thread is created.

1. Explain the use of pthread join function call.

The pthread join waits for the thread created in main to exit before returning back to the main thread again. This is what makes “Thread n returned with 100 + n” return in order.

1. What would happen if the function go is changed to behave like this:

void \*go (void \*n) {

printf("Hello from thread %ld\n", (long)n);

if(n == 5)

sleep(2); // Pause thread 5 execution for 2 seconds

pthread\_exit(100 + n);

// REACHED?

}

This will cause thread 5 to be printed later than the rest, which will cause the main thread to wait for thread 5 to be executed before closing the rest of the threads.

1. When pthread join returns for thread X, in what state is thread X?

Then thread X will be in an exited state.