Computer Science 330
Operating Systems
Siena College
Fall 2022

# Lab 3: Concurrency

**Due: 11:59 PM, Thursday, October 13, 2022**

In this lab, we will begin POSIX threads (pthreads) programming and look at related concurrency issues.

You may work alone or in groups of size 2 of 3 on this lab. However, in order to make sure you learn the material and are well-prepared for the exams, those who work in a group should either collaborate closely while completing the problems or work through the problems individually then discuss them within your group to agree on a solution. In particular, the "you do these and I'll do these" approach is sure to leave you unprepared for upcoming tasks and the exams.

Learning goals:

1. To consider an alternate software solution to the critical section problem for 2 proecesses.
2. To learn the basics of pthreads programming.

---

## Getting Set Up

In Canvas, you will find link to follow to set up your GitHub repository, which will be named `concurrency-lab-yourgitname`, for this lab. Only one member of the group should follow the link to set up the repository on GitHub, then others should request a link to be granted write access.

You may choose to answer the lab questions in the `README.md` file in the top-level directory of your repository, or upload a document with your responses to your repository, or add a link to a shared document containing your responses to the `README.md` file.

---

## Dekker's Algorithm

We looked at Peterson's Algorithm in class as a software solution to the critical section problem for two processes. Another solution is due to Dekker:

- Shared data

```
int turn=0;
boolean flag[2];
flag[0]=false;
flag[1]=false;
```

- Process $P_i$

```
while (1) {
  flag[i]=true;
  while (flag[j]) {
    if (turn != i) {
      flag[i]=false;
      while (turn != i);
      flag[i]=true;
    }
  }

    /* critical section */

  turn=j;
  flag[i]=false;

    /* non-critical section */

}
```

**Question 1:** State the conditions that must be guaranteed for an approach such as this to be considered a solution to the critical section problem (3 points)

**Question 2:** Explain, as precisely as you can, how each of the these conditions is satisfied by Dekker's Algorithm. (5 points)

---

## pthreads Basics

We saw how to use `fork()` to create processes in Unix. We can also create threads with system calls. Thread support in modern operating systems includes POSIX threads, Java threads, and Windows threads. We will look at and make heaviest use of just one type: POSIX threads, usually known as "pthreads".

The basic idea is that we can create and destroy threads of execution in a program, on the fly, during its execution. These threads can then be executed in parallel by the operating system scheduler. If we have multiple processors, we should be able to achieve a speedup over the single-threaded equivalent.

We start with a look at a pthreads "Hello, world" program, which is in the `pthreadhello` directory of your repository for this lab.

The most basic functionality involves the creation and destruction of threads:

- `pthread_create(3THR)` – This creates a new thread. It takes 4 arguments. The first is a pointer to a variable of type `pthread_t`. Upon return, this contains a thread identifier that may be used later in a call to `pthread_join()`. The second is a pointer to a

  pthread_attr_t structure that specifies thread creation attributes. In the pthreadhello program, we pass in NULL, which will request the system default attributes. The third argument is a pointer to a function that will be called when the thread is started. This function must take a single parameter of type void * and return void *. The fourth parameter is the pointer that will be passed as the argument to the thread function.
- pthread_exit(3THR) – This causes the calling thread to exit. This is called implicitly if the thread function called during the thread creation returns. Its argument is a return status value, which can be retrieved by pthread_join().
- pthread_join(3THR) – This causes the calling thread to block (wait) until the thread with the identifier passed as the first argument to pthread_join() has exited. The second argument is a pointer to a location where the return status passed to pthread_exit() can be stored. In the pthreadhello program, we pass in NULL, and hence ignore the value.

Prototypes for pthread functions are in pthread.h and programs need to link with libpthread.a (use -lpthread at link time).

Any global variables in your program are accessible to all threads. Local variables are directly accessible only to the thread in which they were created, though the memory can be shared by passing a pointer as part of the last argument to pthread_create().

**Practice Program:** Create a new version of the pthreadhello.c program in a file pthread-hello-more.c in the pthreadmore directory of your repository that takes a command-line parameter that specifies the number of threads to create. Pass pointers to entries of an array of numbers 1 through n as the thread parameter to identify the thread numbers. (10 points)

A more interesting example is in proctree_threads.

This example builds a "tree" of threads to a depth given on the command line. It includes calls to pthread_self(), which returns the thread identifier of the calling thread.

Try it out and study the code to make sure you understand how it works.

**Question 3:** What is the output when you run the program for a tree of depth 3 on noreaster? (1 points)

**Question 4:** How many calls are made to the split_proc function for a tree of depth *n*? (3 points)

**Question 5:** What is the largest tree you can create on noreaster before you start to get thread creation errors? (1 point)

**Question 6:** What is the largest tree you can create on your Linux VM before you start to get thread creation errors? (1 point)

---

## Submission

Commit and push!

---

## Grading

This assignment will be graded out of 25 points.

| Feature | Value | Score |
|---|---|---|
| Question 1 | 3 | |
| Question 2 | 5 | |
| `pthreadhello-more.c` | 10 | |
| Question 3 | 2 | |
| Question 4 | 3 | |
| Question 5 | 1 | |
| Question 6 | 1 | |
| Total | 25 | |