CPSC 3500 Computing Systems

Project 3: Synchronization on Road Construction

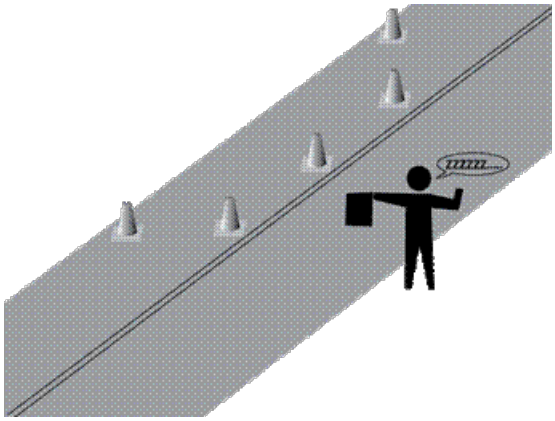
Assigned: 10:00AM, Friday, 02/08/2019

Due: 11:59PM, Sunday, 02/17/2019

Note: you can form a group size of 2 for this project. You may also take it as an individual project!

1. Descriptions

In this project, you will be modeling a common roadway occurrence, where a lane is closed and a flag person is directing traffic.



The figure above shows the scenario. We have one lane closed of a two-lane road, with traffic coming from the **North** and **South**. Because of traffic lights, the traffic on the road comes in bursts. When a car arrives, there is an **80%** chance of another car following it, but once no car comes, there is a **20 second** delay (use the provided pthread_sleep) before any new car will come.

During the times when no cars are at either end, the flag person will fall asleep. When a car arrives at either end, the flag person will wake up and allow traffic from that side to pass the construction area, until there are no more cars from that side, or until there are **10** cars or more lining up on the opposite side. If there are 10 cars or more on the opposite side, the flag person needs to allow the cars from the opposite side to pass.

Each car takes **1 second** (use the provided pthread_sleep) to go through the construction area.

Your job is to construct a simulation of these events where under **no conditions will a deadlock occur**. A deadlock could either be that the flag person does not allow traffic through from either side, or let's traffic through from both sides causing an accident.

2. Requirements

You are required to implement a C/C++ multi-threaded program that uses **pthread** library and avoids deadlock.

You are required to use **pthread synchronization structs** such as semaphores and mutex locks to do synchronization. But, *using mutex locks only for synchronization is never a right solution to this problem*. Remember: **Right synchronization should harm concurrency minimally!**

You are required to output all of the necessary events into two files named "**car.log**" and "**flagperson.log**". For "car.log", you need sequentially number (carID) each car (i.e., carID begins with 1) and indicates the direction it is heading (S or N), the arrival time on the road, the time to start passing the construction area (start-time), and the time to exit the construction area (end-time). The "car.log" looks like:

carID	direction	arrival-time	start-time	end-time
1	S	12:25:11	12:25:12	12:25:13
2	N	12:25:13	12:25:13	12:25:14

The "flagperson.log" tracks when the flag person goes to sleep and when he/she wakes up to work. The file looks like:

Time	State
12:10:11	sleep
12:25:11	woken-up

Design always goes first. In this assignment, we need a README file that must have the following components:

- Team member's names and respective contributions
- How many threads? What is the task for each thread? What is the corresponding thread function name in your code?
- How many semaphores? Describe each semaphore variable's name, initial value and purpose.
- How many mutex locks? Describe each mutex lock's name and purpose.
- Strengths
- Weaknesses.

Please download the README template file for this project.

3. Hints

This problem is similar to the producer/consumer problem with unbounded buffer. But you need to figure out producers, consumer(s), and items to produce/consume.

To get an 80% chance of something, you can generate a random number modulus 10, and see if its value is less than 8. It's like flipping an unfair coin.

When debugging your program, be reminded of using "kill" to kill your process if the process is hanging.

Do not use sleep() in your code!

4. Resources

In this project, you will use Pthread library. The example code below makes your threads sleep for a specific time by using pthread synchronization mechanism.

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

/*****************************************************************************
pthread_sleep takes an integer number of seconds to pause the current thread We
provide this function because one does not exist in the standard pthreads library. We
simply use a function that has a timeout.
 *****************************************************************************/
int pthread_sleep (int seconds)
{
       pthread_mutex_t mutex;
       pthread_cond_t conditionvar;
       struct timespec timetoexpire;

       if(pthread_mutex_init(&mutex,NULL))
       {
              return -1;
       }
       if(pthread_cond_init(&conditionvar,NULL))
       {
              return -1;
       }

       //When to expire is an absolute time, so get the current time and add
       //it to our delay time
       timetoexpire.tv_sec = (unsigned int)time(NULL) + seconds;
       timetoexpire.tv_nsec = 0;

       return pthread_cond_timedwait(&conditionvar, &mutex, &timetoexpire);
}
```

The following code is an example to use pthread_sleep(). You can copy & paste the above pthread_sleep() into your program.

```
/*****************************************************************************
 * This is an example function that becomes a thread. It takes a pointer
 * parameter so we could pass in an array or structure.
 *****************************************************************************/
void *worker(void *arg)
{
       while(1)
       {
              printf("Thread Running\n");
```

```
                fflush(stdout);
                pthread_sleep(1);
        }
       return NULL;
}
```

```
/*****************************************************************************
 * The main function is just an infinite loop that spawns off a second thread
 * that also is an infinite loop. We should see two messages from the worker
 * for every one from main.
 *****************************************************************************/
int main()
{
        pthread_t t_id;

        if ( -1 == pthread_create(&t_id, NULL, worker, NULL) )
        {
                perror("pthread_create");
                return -1;
        }

        while(1)
        {
                printf("Main Running\n");
                fflush(stdout);
                pthread_sleep(2);
        }

        return 0;
}
```

A full pthread reference can be found online at https://computing.llnl.gov/tutorials/pthreads/

The following functions may be needed in your program:

- <pthread.h > *thread control*: pthread_create(), pthread_join(), pthread_detach()
- <semaphore.h>: sem_init(), sem_wait(), sem_post(), sem_destroy()
- Pthread mutex lock: pthread_mutex_lock(), pthread_mutex_unlock(), pthread_mutex_init(), pthread_mutex_destroy(), see http://pubs.opengroup.org/onlinepubs/009695399/functions/pthread_mutex_destroy.html for details
- <time.h>: time(), localtime(), strftime()

5. Submission

For a group project, only one submission is required.

Before submission, you must make sure that your code can be compiled and run on Linux server cs1. You must run the submission command on cs1. You must make sure that your Makefile works properly!

The following files must be included in your submission:

- README
- *.h: all .h files
- *.c/cpp: all .c/cpp files
- Makefile

You should create a package p3.tar including the required files as specified above, by running the command:

```
tar -cvf p3.tar README *.h *.cpp Makefile
```

Then, use the following command to submit p3.tar:

```
/home/fac/zhuy/class/SubmitHW/submit3500 p3 p3.tar
```

If submission succeeds, you will see the message similar to the following one on your screen:

```
======================Copyright(C)Yingwu Zhu=========================
Wed Jan 17 21:53:58 PST 2018
Welcome testzhuy!
You are submitting array.cpp for assignment p3.
Transferring file.....
Congrats! You have successfully submitted your assignment! Thank you!
Email: zhuy@seattleu.edu
=====================================================================
```

You can submit your assignment multiple times before the deadline. Only the most recent copy is stored.

The assignment submission will shut down automatically once the deadline passes. You need to contact me for instructions on your late submission. Do not email me your submission!

6. Grading Criteria

Label	Notes
3a. Compilability, Complete submission, Coding Format & Style (1 pt)	- Your Makefile works properly. - All required files are included in your submission. - Clean, well-commented code.
3b. README file (2 pts)	README file follows the specified requirements, addressing the following: • Team member's names and respective contributions • How many threads? What is the task for each thread? What is the corresponding thread function name in your code? • How many semaphores? Describe each semaphore variable's name, initial value and purpose.

	• How many mutex locks? Describe each mutex lock's name and purpose. • Strengths • Weaknesses.
3c. Functionality (6 pts)	Right synchronization • No excessive semaphores used • Not using mutex locks only • No race condition • No deadlocks • An elegant synchronization solution, following the producer/consumer model Multi-threaded program Output files are produced as required and in the right format.
3d. Resource management, No messy & debugging messages (1 pt)	-No memory leaks (if applicable). -Destroy semaphores/mutex locks -No messy & debugging & testing messages
3e. Overriding policy	If the code cannot be compiled or executed (segmentation faults, for instance), it results in zero point. If the submission is incomplete (e.g., missing files), it results in zero point.
3f. Late submission	Please refer to the late submission policy on Syllabus.