**COMP 3500 Introduction to Operating Systems**

**Project 4 - Dining philosophers problem**

Points Possible: 100 due: 11:59 pm Mar 29th, 2019

**There should be no collaboration among students.** A student shouldn’t share any project code with any other student. Collaborations among students in any form will be treated as a serious violation of the University's academic integrity code.

**Objectives:**

1. Complete the source code to simulate producer/consumer problem.

2. Understand the basics of the POSIX thread library.

3. Build a binary program on a Linux machine.

4. Run the program and record the result of the simulation.

**Requirements:**

* Each student should **independently** accomplish this project assignment. You may discuss with other students to solve the coding problems.
* To embark on this project, you may choose one of the following four options.
  + Important! Option 1: For Mac and Linux users, use SSH to connect to a remote Linux server. Please read files in “tutorial” on Canvas for details.
  + Important! Option 2: For Window 10 users, Please read “enviro\_tutorial”, “Putty Tutorial” and “Win subsystem Linux” for details.
* Important! Read the I/O format specification carefully and follow. It is very important to your final grade of this project!
* You are highly recommended to use a Linux operating system.

**1. Introduction to producer and consumer model**

Dining philosopher problem is a model to schedule how concurrent processes and threads as much as possible while avoid the deadlock. It contains:

1. 5 Philosophers: represent processes or threads that consume the hardware resource
2. 5 Chopsticks: represent 5 unit of the resources

Other concepts involved in our project:

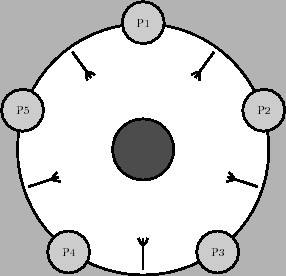
1. POSIX thread: threads mechanism that satisfies POSIX standard (most operating system)
2. Mutex: a "lock" that guarantee that only one person has access.
3. Semaphore: represent one type of resource "capacity"

The mutex only has 1 capacity, 0 or 1, but Semaphore can have more than one capacity.

In this project, we use POSIX threads. The "pthread" is a POSIX thread library written in C and provides the basic functions. To avoid conflicts, we use C only in this project.

Description: There are 5 philosophers sitting around a table and try to eat from the center of the table. 5 chopsticks also lay on the table. Let us call the 5 philosophers in clockwise P1, P2, ...P5. There also 5 chopsticks clockwise S1, S2, ..., S5 on the table. There are only one chopstick between every two philosophers. Every right hand chopstick will have the same index as the philosopher. For example, on the right hand side of P1, the chopstick is called S1. And every left hand side chopstick number is 1+number of the philosopher. For example, the left hand side of P1 is the chopstick S2. See the picture below:

S1 S2



S4

1. A philosopher spend random time to think, then he feel hungry and try to eat.
2. The middle dish can provide enough food for everyone at the same time.
3. But a philosopher only can start to eat when he picked up two chopsticks from left hand side and right hand side to form a pair of chopsticks.
4. If a philosopher take one chopsticks, he will try to fight with neighbours to get another one, and never back off to put down the one in his hand.
5. Once the philosopher is eating, you can not interrupt him.

Here comes the deadlock problem: when all the 5 philosophers start to pick up the right hand side chopsticks at the same time, they get stuck.

Here we can use the project 3 knowledge to avoid this. We use a mutex, every time whenever one philosor start to pick chopsticks, he lock the mutex, and he is the only guy can pick chopsticks. After he get two chopsticks, he unlock the mutex.

See the example.c. You can run "gcc ./example.c -o example -lpthread" and run "./example" to have similar resault:



Notice the red lines. Between the philosopher 0 pick up chopstick 0 and 1, no other philosophers can picking up any chopstick, because the mutex is locked.

This screenshot is my result, your result may differ due to randomness.

Of course, this is not a good enough, because the food is enough for everyone. We try to make as many as possible philosophers to eat at the same time!

Since the five philosopher form a cycle, to break it, we can allow only 4 philosophers at most to pick chopsticks at the same time. 4 philosophers can never form a cycle so no deadlock will happen.

**2. Follow the Format Specification (10 Points)**

In the source file "Firstname\_Lastname.cpp", you will find first four comment lines:

**// #0#BEGIN# DO NOT MODIFY THIS COMMENT LINE!**

**// Firstname**

**// Lastname**

**// #0#END# DO NOT MODIFY THIS COMMENT LINE!**

Your first task is modifying the two lines **between** the beginning line and end line. Change them into your first name and last name. Remember the strings are case-sensitive, so capitalize the first letter of the word and follow exactly the syntax of the example.

You can see lots of similar code blocks in the file. You are free and supposed to fill your answer between those special beginning and ending comment lines. You can insert and edit multiple lines between special comment lines in anyways, however(Important!), as the comment indicated, do not modify the special begin and comment lines **themselves**!

Let's do the second task. Scroll down to the bottom of the file and find those lines (press "shift + g" if you are using vi/vim):

**// #8#BEGIN# DO NOT MODIFY THIS COMMENT LINE!**

**banner\_id = 903900281;**

**// #8#END# DO NOT MODIFY THIS COMMENT LINE!**

Look at your student ID card, check your banner ID. Again, change the "banner\_id" value to your own ID. Your unique student id will be compiled into the program, and the input of the experiment also uniquely depends on your ID.

Warning: Since every student has a unique id number, the later compiled binary file is also unique. Copy binary file from other students will be easily detected!

**3. Complete Source Code (70 Points)**

Read the source code "example.c", try to understand the function of each line of code. Try to understand the basic usage of pthread library function and semaphores.

Follow the instructions in the comments Firstname\_Lastname\_4.c and insert proper code into the rest 7 blocks to implement a producer/consumer model. (Only C file acceptable in this project).

**4. Run and Record Simulation Result (10 Points)**

Compile your source code into a binary program. For example, use following command to include the pthread library:

**$ gcc Firstname\_Lastname\_4.c -o Firstname\_Lastname\_4 -lpthread**

After you compile the c source code successfully, please use the script command to record the running result of the program Firstname\_Lastname:

**$ script Firstname\_Lastname\_4.script**

**Script started, file is Firstname\_Lastname\_4.script**

**./Firstname\_Lastname\_4**

After you run the program, you will have the following results:

**Banner id: 903900281**

**Philosopher 0 is thinking**

**...**

You should have similar result and difference in the banner ID. Then exit recording and save it into typescript file "Firstname\_Lastname\_4.script" by the following command:

**$ exit**

**exit Script done, file is Firstname\_Lastname\_4.script**

Warning: Since every student has a unique id number, the result of the simulation is also unique. Copy simulation results from other students will be easily detected!

**5. Deliverables (10 Points)**

Since you have generated multiple script files, please save all the script files in one

directory. Create a folder "Firstname\_Lastname\_4" first. Please make sure the **folder name** correct. You can use the command mv to rename the folder:

**$ mv folder-name Firstname\_Lastname\_4**

Make sure all the three files are into this folder. You should have those following files inside the folder:

1. commands recording file: Firstname\_Lastname\_4.script
2. executable binary file: Firstname\_Lastname\_4
3. source code file: Firstname\_Lastname\_4.c

Achieve all the folder into a single tarred and compressed file with a tar command.

**tar -zcvf Firstname\_Lastname\_4.tar.gz Firstname\_Lastname\_4**

You need to submit one tarred file with a format: Firstname\_Lastname.tar.gz

**Grading Criteria:**

1. Follow the format specification: 10%
   1. Do not break the special comments.
   2. Input your name properly (mark #0).
   3. Input your banner id properly (mark #8).
2. Complete the source code: 70%
   1. Each blank worth 10, total 70 (mark #1 ~ #7).
   2. See detailed specification for each blank in the source code file.
3. Compiling and running result: 10%
   1. Compile the code successfully.
   2. Record the running results.
4. Deliverables: 10%
   1. Contains all the files.
   2. Naming all the files properly.

**Rebuttal Period:**

You will be given **three business days** to read and respond to the comments and grades of your homework or project assignment. The TA may use this opportunity to address any concern and question you have. The TA also may ask for additional information from you regarding your homework or project.