OPERATING SYSTEMS
UCCD 2103
JAN 2020

ASSIGNMENT REPORT
Part A

No.	Student ID	Name	Programme (CN/CS/CT/IA)
1			
2			
3			

Assignment

Guideline:

This assignment is a **group assessment**. Each group (maximum 3 students per group, not limited to same programme nor tutorial classes) has to practice the following and submit the hardcopy reports. **Maximum marks for this assignment (Part A) is 90.** It requires you to install the Linux OS (30 marks). Running a virtualized Linux (Virtualbox or VMWare Player) is also acceptable. You can use either **32-bit** or **64-bit** Linux, depending on your host OS if you use virtualization. For example, if your computer runs on Windows 7 32-bit, then you have to use Linux 32-bit on your virtual machine. To limit uncertainties and possible issues you might run into, you are recommended (but not limited) to use Fedora or CentOS for this assignment. Make sure you allocate at least 50GB for your Linux partition. **The username of your Linux system must be set to your first name.** For example, the username of my system is *cheesiang*. Your report shall contain all group member's names and IDs, and your group leader's email address. Also, write down the specification of the system that you use to perform this assignment at the last page of your report. Your report should provide the **screenshots** of the terminal to support **all** your answers. If the screenshot size is too big and resulting to a very small font size in your printout, you may resize your desktop resolution to 800x600 before taking the screenshot. If the screenshot is still too large (or the font is too small), you may crop it; **as long as you can proof the originality of your work, by showing the username in your terminal**. For example, for command lines-based question, you should not crop out your username as it is used to identify your group's work. Any group reports that have the same screenshots or usernames will be counted as plagiarism and be given 0 marks. The report and source codes shall be printed in black and white colour, double sided, and without comb binding to minimize cost. Please compress the softcopy of your report and source codes and upload it to the WBLE. Use the following naming format: `yourname_tutorialgroup`. Only the group leader needs to upload them. Please cite all the references you use. The due date is 9th of March, 2020, 12PM. The submission process in WBLE later on.

For Part A:

Your softcopy WBLE submission should contain the following files:

- Your report
- `chef-solution.c`
- `businessman-solution.c`

Your hardcopy submission should contain:

- Your report with:
  - Appendix A: `chef-solution.c`
  - Appendix B: `businessman-solution.c`

Multi-threading, race condition, and Semaphore tutorials

1. *Multi-threading and race condition*

   a) This question is about multi-threading and race condition. It requires the following packages to be installed as **root** before you can attempt it. Thus, at your terminal, switch to root before executing the following steps:

      i. `gcc` C language compiler will be used. Command: `yum install gcc`

      ii. `pthread` library is required to be installed. `pthread` is a POSIX Linux KLT threading library. Command: `yum install glibc-devel` to install.

   b) Switch back to your own user login. Download `multi-thread.c` source code from the WBLE.

      i. At your terminal, browse to the `Downloads` directory by running:
         `cd ~/Downloads`

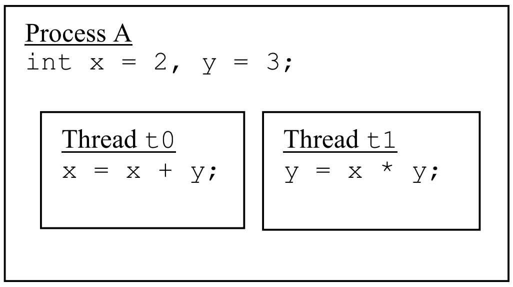
      ii. To compile it into an executable file called `multi-thread`, enter:
         `gcc multi-thread.c -o multi-thread -lpthread -lrt`

      iii. To run it, enter: `./multi-thread`

   The program will create two concurrent threads that greet "Hello world!" together with their respective thread number. Both threads perform the same thing: (1) Identify their own thread number, and (2) Print their own thread number on the screen. Study the source codes to get familiar with the syntax of multi-threading using `pthread`.

2. *Threads Synchronization using Semaphore:*

   a) Download `race.c` source code from the WBLE. The process will create two threads and each thread will perform the calculation as shown in Figure 1 and print the final values of both variables after both threads have finished.



```
Process A
int x = 2, y = 3;

Thread t0          Thread t1
x = x + y;         y = x * y;
```

Figure 1

   Compile it into an executable file called `race` and run it 100 iterations by using the shell script provided, called `runner.sh`.
   First, enable `runner.sh` to be executable: `chmod 755 runner.sh`
   Then, run the script, enter: `./runner.sh`
   Study the source codes and observe the output in the terminal: For 100 runs, you will notice some of the output is different from the rest. This is because race condition happened; i.e. if Thread 0 runs first, the final output will be x = 5, y = 15. If Thread 1 runs first, the final output will be x = 8, y = 6.

b) We can use Semaphore to control which thread to enter the critical section first. For example, we want thread 1 to enter the critical section first. To learn how to use semaphore, you may download the `race-sem.c` sample source code as a tutorial. Compile it using: `gcc race-sem.c -o race-sem -lpthread -lrt` and run it using: `./race-sem`

c) Study how semaphore is implemented (declaration, initialization, sem_wait, sem_post, and sem_destroy). In that source code, we force thread `t0` to wait for thread `t1` so that thread `t1` will enter the Critical Section first every time we execute, and thus eliminate race condition all together.

Threads synchronization and Semaphore questions

1. Figure 2 shows a kitchen with two chefs, Chef 0 and 1; and three frying pans, Pan A, B, and C. A new order is received; chef 0 is required to prepare 10 sandwiches while chef 1 is required to prepare 10 hotdog burgers. Pan B is shared by the chefs and only one chef can use it at a time. Chef 0 needs pan B to fry ham and chef 1 needs pan B to fry egg. The problem can be solved using semaphore.

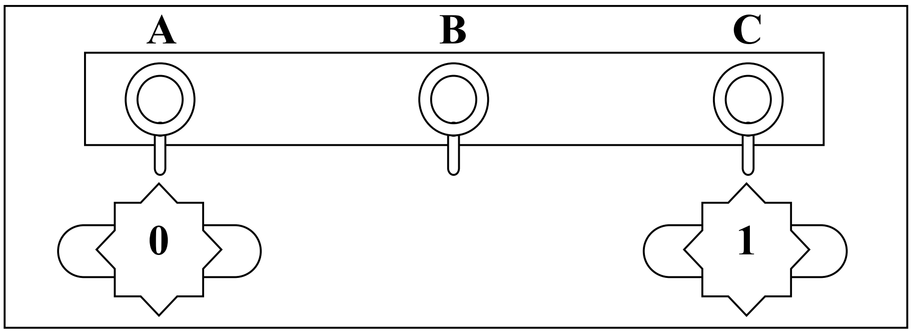


Figure 2

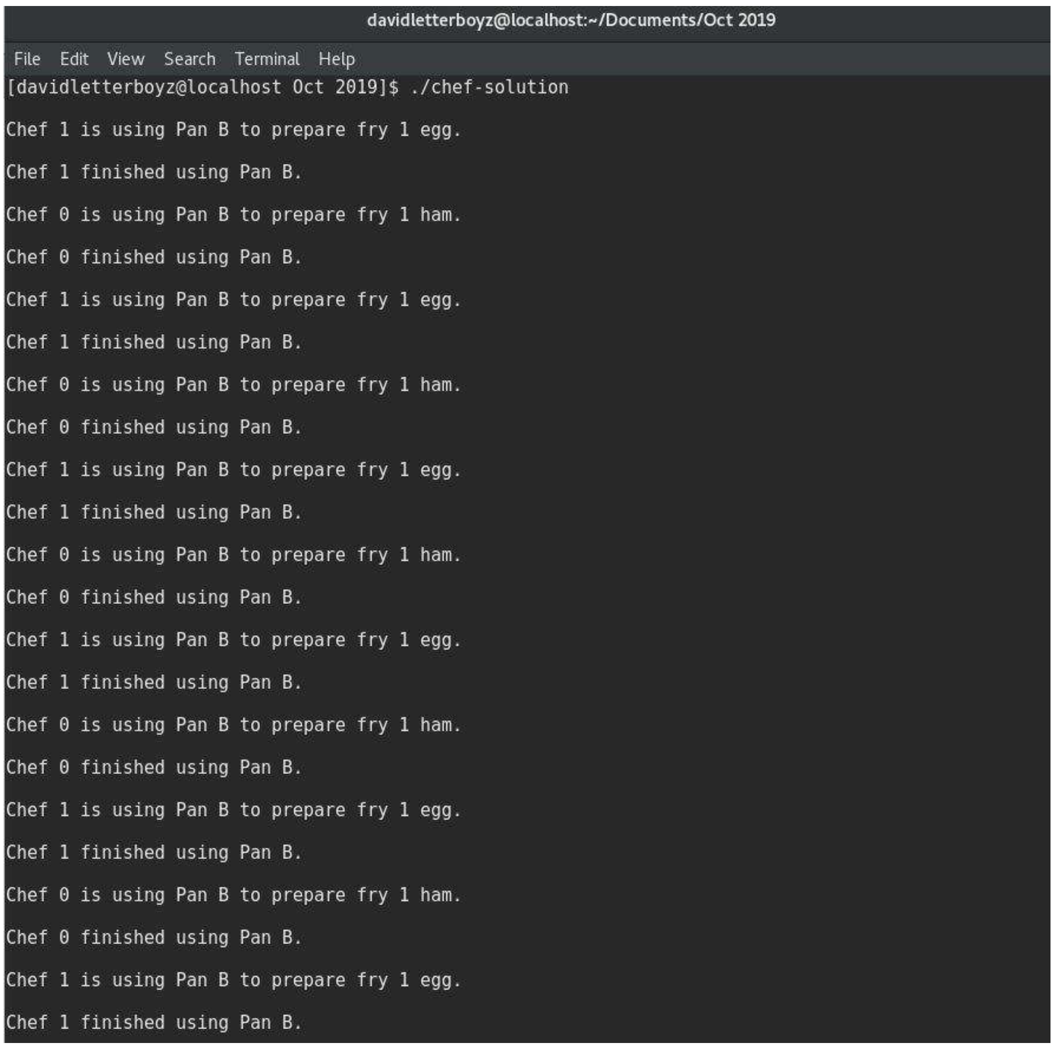
The problem can be simulated by using two threads. Assume that:

- thread `t0` simulates Chef 0. Chef 0 uses 2 seconds to fry a ham and wait 1 second before repeating the task again.
- thread `t1` simulates Chef 1. Chef 1 uses 2 seconds to fry an egg and wait 1 second before repeating the task again.

Download the file `chef.c`, compile it (`gcc chef.c -o chef -lpthread -lrt`) and execute it (`./chef`).

a) Write down your observation about the output of the program by describing what is wrong with the execution sequence. (10 marks)

b) Fix the issue by implementing semaphore into the `chef.c` source code. Rename your solution file into `chef-solution.c` and test the correctness by running your `chef-solution` program multiple times. (15 marks)
Hint: A correct sample output is shown in the Figure 3.


davidletterboyz@localhost:~/Documents/Oct 2019
File Edit View Search Terminal Help

```
[davidletterboyz@localhost Oct 2019]$ ./chef-solution
Chef 1 is using Pan B to prepare fry 1 egg.
Chef 1 finished using Pan B.
Chef 0 is using Pan B to prepare fry 1 ham.
Chef 0 finished using Pan B.
Chef 1 is using Pan B to prepare fry 1 egg.
Chef 1 finished using Pan B.
Chef 0 is using Pan B to prepare fry 1 ham.
Chef 0 finished using Pan B.
Chef 1 is using Pan B to prepare fry 1 egg.
Chef 1 finished using Pan B.
Chef 0 is using Pan B to prepare fry 1 ham.
Chef 0 finished using Pan B.
Chef 1 is using Pan B to prepare fry 1 egg.
Chef 1 finished using Pan B.
Chef 0 is using Pan B to prepare fry 1 ham.
Chef 0 finished using Pan B.
Chef 1 is using Pan B to prepare fry 1 egg.
Chef 1 finished using Pan B.
Chef 0 is using Pan B to prepare fry 1 ham.
Chef 0 finished using Pan B.
Chef 1 is using Pan B to prepare fry 1 egg.
Chef 1 finished using Pan B.
```

Figure 3

2. Imagine if you have two businessmen who have the same logic: each businessman will offer his business card and then waits for the other party to make his offer. To simulate this, each businessman may use semSignal (`sem_post` in code) to indicate "This is my card" and followed by semWait to signify "I'm waiting for your card". This is done by two semaphores: A and B, which are both initialized to 0 because everyone must wait until the other party offers his card.

   Additionally, global variables `buf_A` and `buf_B` are used. Variable `buf_A` is for thread A to store his contact number (1234) for thread B to retrieve. `thread_B` will do the opposite. In the `businessman.c` code, each iteration of `thread_A()` does the following:

   - Store his phone number "1234" in its local variable `Var_A`.
   - Notifies `thread_B`, with a signal call `sem_post`(&B), that "Here is my card and I am ready to exchange my card with you". Then, blocks itself with a call `sem_wait`(&A) to simulate he's waiting for the other party to offer his card.
   - Once `thread_B` is ready to exchange his business card by calling `sem_post`(&A), `thread_A` wakes up, moves its card into the global

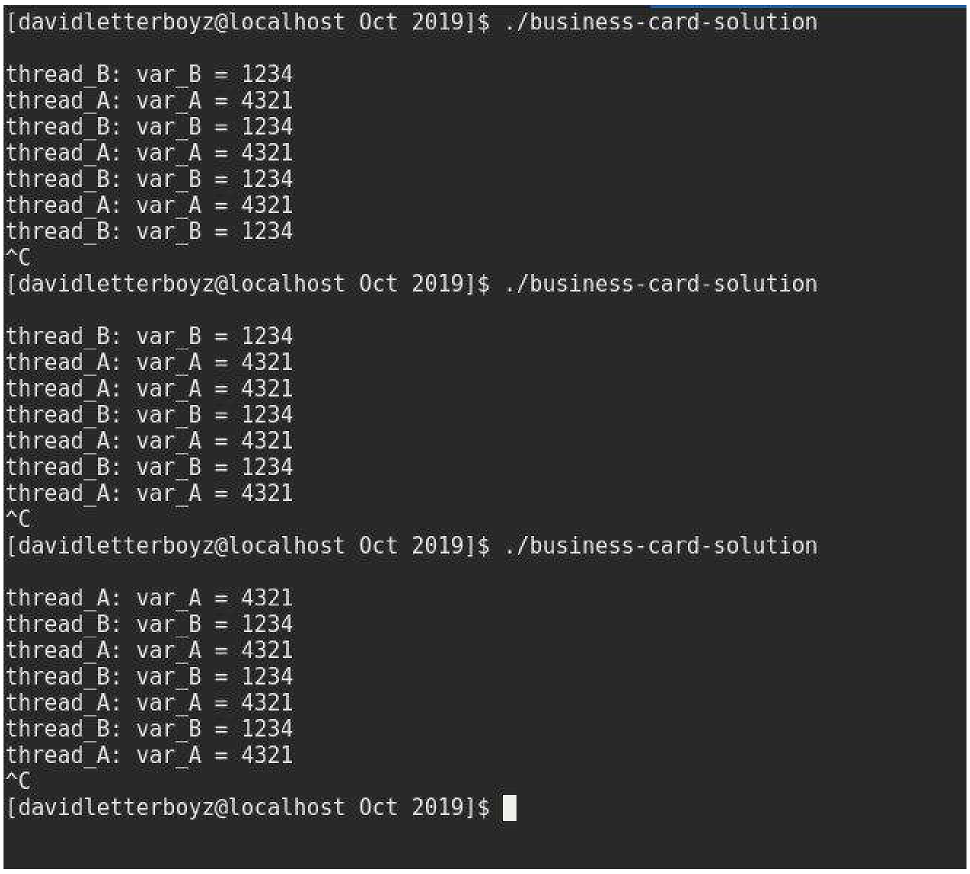
variable `buf_A` for that `thread_B` to retrieve, and, finally, retrieves the other party's card from variable `buf_B` to be stored in `buf_A`.

`businessman.c` tries to simulate the scenario. Compile it and run the code. Answer the following question:

a) Write down your observation about the output of the program by describing the output and what is wrong with the execution sequence. You may use diagram(s)/source codes lines to support your answer. (15 marks)

b) Fix the issue by implementing mutual exclusion into the `businessman.c` source code. Test the correctness by running your solution multiple times. (20 marks)

Hint: A correct sample output is shown in the Figure 4 where the `thread_B`'s `var_B` has value 1234 and `thread_A`'s `var_A` has 4321 no matter how many times you repeat the program.


```
[davidletterboyz@localhost Oct 2019]$ ./business-card-solution

thread_B: var_B = 1234
thread_A: var_A = 4321
thread_B: var_B = 1234
thread_A: var_A = 4321
thread_B: var_B = 1234
thread_A: var_A = 4321
thread_B: var_B = 1234
^C
[davidletterboyz@localhost Oct 2019]$ ./business-card-solution

thread_B: var_B = 1234
thread_A: var_A = 4321
thread_A: var_A = 4321
thread_B: var_B = 1234
thread_A: var_A = 4321
thread_B: var_B = 1234
thread_A: var_A = 4321
^C
[davidletterboyz@localhost Oct 2019]$ ./business-card-solution

thread_A: var_A = 4321
thread_B: var_B = 1234
thread_A: var_A = 4321
thread_B: var_B = 1234
thread_A: var_A = 4321
thread_B: var_B = 1234
thread_A: var_A = 4321
^C
[davidletterboyz@localhost Oct 2019]$ █
```

References:

[1] https://www.geeksforgeeks.org/use-posix-semaphores-c/