COMP604 Operating Systems
Assignment

Semester 2, 2016

DUE ON: Week 12 Monday, 17 October 2016, 11:59 PM
Assignment Worth: 30% of total marks

NB: This is a team or individual assignment: The assignment is intended to be a team assignment for teams of two. However, if you feel it inconvenient to do this as a team, you are free to choose to submit an individually worked assignment. For those who feel a bit "challenged" with programming due to their major's nature, this is a chance to team up with somebody more comfortable with programming.

NB: Assignments will be accepted up to five (5) days late, but a penalty of 5% per day (or part of a day) late will be imposed on either the team or the individual depending on the circumstances.

NB: Students are referred to the school's policy on plagiarism. A confirmed case will incur zero mark to all the involved students.

1. Assignment Goal

The goal of this assignment is to develop a better understanding of multithreading and process/thread synchronization using semaphores. You can do this assignment either using **Java** (`Runnable` interface and `java.util.concurrent.Semaphore`) or **C/C++** (`Pthreads`).

2. Assignment Overview

In Lecture 5 and Lab 5, we have discussed a semaphore-based solution to the producer-consumer problem using a bounded buffer. In this assignment we will design a programming solution to the **bounded-buffer problem** using the producer and consumer processes. The solution presented in Lecture 5 uses three semaphores: `empty` and `full`, which count the number of empty and full slots in the buffer, and `mutex`, which is a binary (or mutual exclusion) semaphore that protects the actual insertion or removal of items in the buffer. For this assignment, standard counting semaphores will be used for `empty` and `full`, and a binary semaphore will be used to represent `mutex`. The producer and consumer--running as separate threads--will move items to and from a buffer that is synchronized with these empty, full, and mutex structures. You can solve this problem using either Java concurrent API or POSIX Pthreads API.

3. Assignment Tasks

Task 1: The Buffer (or Buffer.java)
Internally, the buffer will consist of a fixed-size array of type `buffer_item` (which will be defined using a `typedef`). The array of `buffer_item` objects will be manipulated as a circular queue. The definition of `buffer_item`, along with the size of the buffer, can be stored in a Java class file or C header file such as the following:

```
 /* buffer.h */           // Java: constant.java
typedef int buffer_item; // Java: omit it in Java or use wrapper class
#define BUFFER_SIZE 5  // Java: public static final int BUFFER_SIZE = 5;
```

The buffer will be manipulated with two functions, `insert_item()` and `remove_item()`, which are called by the producer and consumer threads, respectively. A skeleton outlining these functions appears as:

```
#include <buffer.h> //Java: import
/* the buffer */
buffer_item buffer[BUFFER_SIZE];
int insert_item(buffer_item item) {
     /* insert item into buffer
     return 0 if successful, otherwise
     return -1 indicating an error condition */
}


int remove_item(buffer_item *item) {

     /* remove an object from buffer
     placing it in item
     return 0 if successful, otherwise
     return -1 indicating an error condition */
}
```

The `insert_item()` and `remove_item()` functions will synchronize the producer and consumer using the algorithms outlined in the lecture slides. The buffer will also require an initialization code section (which is part of the `main()` function) that initializes the `empty`, `full`, and `mutex` semaphores.

Task 2: The `main()`**Function** (or Main.java)

The `main()` function will initialize the buffer and create the separate producer and consumer threads. Once it has created the producer and consumer threads, the `main ()` function will sleep for a period of time and, upon awakening, will terminate the application. The `main ()` function will be passed three parameters on the command line:

1. How long to sleep before terminating
2. The number of producer threads
3. The number of consumer threads

A skeleton for this function appears as:

```
#include <buffer.h>
int main(int argc, char *argv[]) { //Java: public static void
                                  //      main(String[] args)
/* 1. Get command line arguments argv[1], argv[2], argv[3]*/
/* 2. Initialize buffer */
/* 3. Create producer threads */
/* 4. Create consumer threads */
/* 5. Sleep */
/* 6. Exit */
```

Task 3: Producer and Consumer Threads (or Producer.java and Consumer.java)

The producer thread will alternate between sleeping for a random period of time and inserting a random integer into the buffer. Random numbers will be produced using the `rand()`function, which produces random integers between `0` and `RAND_MAX` (Java: `Math.random()`). The consumer will also sleep for a random period of time and, upon awakening, will attempt to remove an item from the buffer.

Please print both the `process id` and the integer item produced/consumed as output message.

Hint: In Linux C, we can get thread id using: `pthread_self()`. In Java, inside the `run()` method, we can get thread id using:
`long threadId = Thread.currentThread().getId();`

An outline of the producer and consumer threads appears as:

```
#include <stdlib.h> I/* required for rand() */
#include <buffer.h>
#include <unistd.h>
void *producer(void *param) {
     buffer_item item;

     while (TRUE) {
          /* sleep for a random period of time */
          sleep( ... );
          /* generate a random number */
          item = rand() ;
          if (insert_item(rand))
            fprintf("report error condition");
          else
            printf("producer %lu produced %d\n",
            pthread_self(), rand);
     }
}

void *consumer(void *param) {
     buffer_item item;
     while (TRUE) {
          /* sleep for a random period of time */
          sleep ( ... ) ;
          if (remove_item(&item))
            fprintf("report error condition");
          else
           printf("consumer %lu consumed %d\n",pthread_self(),
     item);
     }
}
```

Task 4: Test Run

Please create a section called "*Running result analysis*" in a text file `readme.txt` and include both the running output and your explanation to the result in this section. Please allow your main function to run (sleep) for 10 seconds.

1. Run your program using 1 producer, 1 consumer, and buffer size 5. Explain your result.
2. Run your program using 5 producer, 5 consumer, and buffer size 1. Explain your result.

3. Run your program using one producer, 5 consumer, and buffer size 5. Explain your result.
4. Run your program using 5 producer, 5 consumer, and buffer size 10. Explain your result.

Submission Requirements

1. You should ensure that all files used for the assignment sit in a directory called "*assignment*".
2. For Java code, five files need to be in this directory: `Main.java, Buffer.java, Producer.java, Consumber.java, Constants.java`. For C code, two files need to be in this directory: `buffer.h`, and `buffer.c`.
3. In addition, a file `readme.txt` that includes:
   a. State if it's a team or individual work. If it's a teamwork, please give the name and student number of your team members.
   b. A list of all the files in the directory
   c. A brief instructions on how to compile and run the program
   d. Your answer to "Task 4: Test Run.

For submission, please compress the whole "*assignment*" directory as `assignment.zip` and <u>submit individually</u> using the Assignment link in AUTonline (inside the Assignment area).

For each submitted program file, we require clear comments including student information, description of the file, and description of each function defined in this file.

Assignments that fail to follow "submission requirements" will <u>NOT</u> be assessed.

Marking Scheme: See next page.

The assignment will be marked out of 100 and will contribute 30% towards assessment of this course.

NB: If your code cannot pass compilation or has runtime error, you will get zero mark for the assignment.

Assessment item	**Marks**
0.1: Comment and readme.doc	5
0.2: Quality of code	4
Task 1: Buffer	
1.1: Variable Declaration	3
1.2: The `insert_item()` function	12
1.3: The `remove_item()` function	12
Task 2: The `main()` Function	
2.1; Get command line arguments	3
2.2: Initialize buffer	5
2.3: Create producer thread(s)	10
2.4: Create consumer thread(s)	10
2.5: sleep and exit code	2
Task 3: Producer and Consumer Threads	
3.1: The `producer()` function	10
3.2: The `consumer()` function	10
Task 4: Test Run	
4.1: Running output and explanation	14
Total	100
Compilation Error	**Get zero total mark**
Runtime Error	**Get zero total mark**