DALHOUSIE UNIVERSITY
FACULTY OF COMPUTER SCIENCE

CSCI 3120 Operating Systems

Assignment 5: Synchronization

Due: 16:00, Dec. 8, 2023

- **Teaching Assistants:**
    - Kamran Awaisi (km521977@dal.ca)
    - Hui Huang (huihuang@dal.ca)
    - Arka Ghosh (arka.ghosh@dal.ca)
    - Yitong Zhou (yt760204@dal.ca)
    - Shiyun Wang (sh776410@dal.ca)
- **Help Hours: FCS Learning Center (Goldberg 233) or via the Channel "Office Hour - TAs" on MS Teams:**
    - Monday: 1:00pm-2:00pm, Kamran Awaisi
    - Tuesday: 1:00pm-2:00pm, Hui Huang
    - Wednesday: 1:00pm-2:00pm, Arka Ghosh
    - Thursday: 1:00pm-2:00pm, Yitong Zhou
    - Friday: 1:00pm-2:00pm, Shiyun Wang

1. Assignment Overview

In this assignment, you need to design and implement a C program that solves the bounded buffer problem. Semaphore and mutex lock are used to synchronize the producer and consumer threads.

2. Important Note

There is a zero-tolerance policy on academic offenses such as plagiarism or inappropriate collaboration. By submitting your solution for this assignment, you acknowledge that the code submitted is your own work. You also agree that your code may be submitted to a plagiarism detection software (such as MOSS) that may have servers located outside Canada unless you have notified me otherwise, in writing, before the submission deadline. Any suspected act of plagiarism will be reported to the Faculty's Academic Integrity Officer in accordance with Dalhousie University's regulations regarding Academic Integrity. Please note that:
1) The assignments are individual assignments. You can discuss the problems with your friends/classmates, but you need to write your program by yourself. There should not be much similarity between your program and others' programs.
2) When you refer to some online resources to complete your program, you need to understand the mechanism, then write your own code. Your code should not be similar to the online resources. In addition, you should cite the sources via comments in your program.

3) You may use AI-driven tools to assist your learning, but you are not allowed to use them to produce work to be submitted for course evaluations. Your submitted program should not be highly similar to others' programs.

3. Detailed Requirements

1) Overview: In Chapter 3, the bounded buffer problem was first discussed. In Chapter 6, a solution based on "circular array" was described. However, with this solution, producers and consumers do not work in the correct manner if they are executed concurrently. Namely, producers and consumers are not properly synchronized. In Chapter 7, a semaphore-based method was presented to solve the synchronization problem associated with the solution in Chapter 6. In this assignment, you need to design and implement a C program that combines the solution in Chapter 6 and the semaphore-based method in Chapter 7 to tackle the bounded buffer problem. The detailed requirements of the C program are described in the following sections.

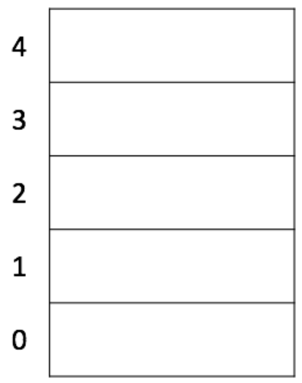
Note that:

a) In this assignment, producers and consumers, running as separate threads, move items to and from a buffer respectively.
b) The method presented in Chapter 7 uses three semaphores: *empty* and *full* (which count the number of empty and full slots in the buffer), and *mutex* (which is a binary semaphore that protects the actual insertion or removal of items in the buffer). In this assignment, you need to use two POSIX unnamed semaphores to implement *empty* and *full*. In addition, you need to use a POSIX mutex lock (instead of a binary semaphore) to implement *mutex*.

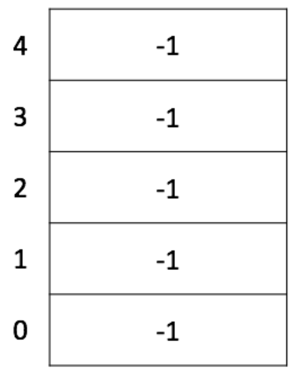
2) Buffer: In this assignment, we assume that the buffer includes 5 slots for items. The bounded buffer should be implemented as a fixed-size array. Obviously, the array consists of 5 elements. You should use the following statements to define the size of the buffer, the data type of buffer item, and the buffer:

```
#define  BUFFER_SIZE  5
typedef  int  buffer_item;
buffer_item buffer[BUFFER SIZE];
```

According to the above statements, each item in the buffer is an integer. The integer is generated by producer threads, which will be discussed later. The following figure shows the structure of the buffer. Obviously, each buffer item corresponds to an index number, which is in the range of 0 to 4. In this assignment, we use *buffer[N]* to refer to an item in the buffer, where *N* is an index number.



In this assignment, we assume that each item generated by producers is a non-negative integer. Furthermore, we assume that when a buffer slot is empty, "-1" should be placed into the corresponding buffer slot. Note that before producer and consumer threads are created, the buffer is empty; therefore, each buffer item should be initialized with "-1". The following figure shows the status of the buffer in this situation:



In this assignment, the buffer is implemented as a circular array (note that you can refer to the solution in Chapter 6 to understand how circular array works). And the buffer can be manipulated with two functions, *insert_item()* and *remove_item()*, which are called by the producer and consumer threads, respectively. Specifically, *insert_item()* is used to insert an item into the buffer. Let us consider an example situation in which the C program is started and no producer/consumer thread has been created. In this situation, the buffer is completely empty. Thereafter, if *insert_item()* is invoked, it will insert an item into the buffer slot corresponding to *buffer[0]*. The following figure shows the status of the buffer after "125" (note that "125" is an example item generated by a producer) is inserted into the empty buffer:

4	-1
3	-1
2	-1
1	-1
0	125

The following figure shows the status of the buffer after another item, "569", is further inserted into the buffer:

4	-1
3	-1
2	-1
1	569
0	125

In this assignment, *remove_item()* is used to delete an item from the buffer. Suppose that *remove_item()* is invoked after "569" is inserted into the buffer. The following figure shows the status of the buffer after *remove_item()* is invoked:

4	-1
3	-1
2	-1
1	569
0	-1

3) main(): The *main()* function in the C program initializes the buffer and creates the producer/consumer threads. Once it has created the producer and consumer threads, the thread corresponding to the *main()* function will sleep for a period of time and, upon awakening, will terminate the C program. During this sleep period, the producer threads

attempt to add items to the buffer and the consumer threads try to consume the items in the buffer.

When a user runs the C program, three numbers (as command-line arguments) are passed to the *main()* function. Assume that the name of the executable file is "BoundedBuffer", here is an example command to run the executable:

```
BoundedBuffer  10  2  3
```

The command-line arguments indicate:

a) How long the thread corresponding to the *main()* function will sleep before terminating the C program
b) The number of producer threads to be created
c) The number of consumer threads to be created

With the example command, the thread corresponding to the *main()* function will sleep for 10 seconds; 2 producer threads and 3 consumer threads will be created. In this assignment, you can assume that the <u>command-line arguments are always positive integers</u>.

A skeleton for the *main()* function can be found here:

```
int main(int argc, char *argv[]) {
  1. Get command-line arguments argv[1],argv[2],argv[3]
  2. Initialize semaphores and mutex lock
  3. Initialize buffer
  4. Create producer thread(s)
  5. Create consumer thread(s)
  6. Sleep and thereafter terminate the C program
}
```

In this assignment, <u>when multiple producer threads are created, we assign an index number to each producer thread</u>. The index numbers are integers and start from 0. Namely, the index number of the first producer thread is 0, the index number of the second producer thread is 1, etc. In a similar manner, <u>index numbers are assigned to consumer threads</u>. Namely, the index number of the first consumer thread is 0, the index number of the second consumer thread is 1, etc.

There are different methods to associate a unique index number with a created thread. One possible method works in the following manner: When the thread corresponding to main() creates a thread using pthread_create(), an argument corresponding to a unique index number is passed to the created thread via pthread_create().

4) <u>Producer and Consumer Threads</u>: After a producer thread is created, it sleeps for a random period of time (in the range of 0 to 4 seconds). After it wakes up, it attempts to insert a random integer (in the range of 0 to RAND_MAX) into the buffer. Note that RAND_MAX is a constant related to the C library function rand(). The details of rand() will be presented later. Thereafter, the producer sleeps for a random period of time (in the range of 0 to 4 seconds), then attempts to insert another random integer (in the range of 0 to RAND_MAX) into the

buffer. This alternating behavior continues till the thread corresponding to the main() function terminates the C program.

The alternating pattern also applies to the consumer threads. Namely, after a consumer thread is created, it sleeps for a random period of time (in the range of 0 to 4 seconds) and, upon awakening, attempts to remove an item from the buffer by assigning "-1" to the corresponding buffer slot. This alternating behavior continues till the thread corresponding to the main() function terminates the C program.

In this assignment, random numbers are generated with the rand() function, which produces random integers between 0 and RAND_MAX. For the random numbers used to set the sleep time of producer/consumer threads, you can use the modulus operator (i.e. %) to convert the original random number into an integer in the range of 0 to 4. Note that rand() does not work well for a multi-thread program on MS Windows. Therefore, if you have been using MS Windows to debug your assignments and rand() produces the same number for different threads, you could try to run your program on csci3120.cs.dal.ca (which is a linux machine).

Here is a tutorial on rand():
https://en.cppreference.com/w/c/numeric/random/rand

An outline of the producer thread can be found below:

```
void *producer(void *param) {
    buffer_item item;
    while (true) {
        /* sleep for a random period of time: 0-4 seconds */
        sleep(...);
        /* generate a random number */
        item = rand();
        /* insert an item */
        insert_item();
    }
}
```

An outline of the consumer thread can be found below:

```
void *consumer(void *param) {
    while (true) {
        /* sleep for a random period of time: 0-4 seconds */
        sleep(...);
        /* remove an item */
        remove_item();
    }
}
```

5) Output: When your program is executed, each time an item is inserted into the buffer, a statement in the following format should be displayed on the screen:

Producer X inserted item Y into buffer[Z] where X, Y, and Z represent the index number of the inserting producer, the non-negative integer corresponding to the inserted item, and the index number of the corresponding buffer slot.

Each time an item is removed from the buffer, a statement in the following format should be displayed on the screen:

Consumer X consumed item Y from buffer[Z]

where X, Y, and Z represent the index number of the removing consumer, the non-negative integer corresponding to the removed item, and the index number of the corresponding buffer position.

The output corresponding to the following command used to run your C program (assuming the name of the executable is BoundedBuffer) can be found in the appendix at the end of this document. Note that, due to randomness, the output might be different each time the following command is executed. However, the behavior of the bounded buffer should be correct during each execution. For example, an item needs to be produced first before it is removed; a producer always inserts a new item into the next free slot; a consumer always consumes the earliest item in the buffer.

```
BoundedBuffer 10 2 1
```

6) Additional Requirements:

a) Command-line Arguments: In this assignment, you can assume that:
   a. The command-line argument for the sleep length of the thread corresponding to main() is in the range of 1 to 100.
   b. The command-line argument for the number of producer threads is in the range of 1 to 5.
   c. The command-line argument for the number of consumer threads is also in the range of 1 to 5.

b) Use of POSIX semaphore and mutex lock:
   a. With the solution in Chapter 6, “in” is used to indicate the next free slot in the buffer and “out” is used to indicate the slot corresponding to the earliest item in the buffer. In addition, “count” is used to check whether the buffer is full or empty.
   b. In this assignment, “in” and “out” can still be used. However, “count” should NOT be used. Instead, two POSIX semaphores (i.e. empty and full) are used to check whether the buffer is full/empty; one POSIX mutex lock (i.e. mutex) is used to ensure that only one thread can revise the buffer at a time. The details of the synchronization method based on semaphore and mutex lock can be found in Chapter 7.
   c. In Chapter 7, we learned that unnamed semaphores can be initialized using sem_init(). This function works well on csci3120.cs.dal.ca. However, it does not work on macOS. When you work on your assignment, please make sure that csci3120.cs.dal.ca is the programming platform.

c) How to terminate producer/consumer thread: As mentioned in Chapter 4, each C program starts with at least one thread, which corresponds to the main() function.

The maximum lifetime of every thread belonging to the same program is the lifetime of the main() thread. When the main() thread terminates, all threads belonging to the same program will terminate. In this assignment, the main() thread does not need to include pthread join() to wait for child threads to terminate. Instead, once the sleeping period of the main() thread is over, the main() thread will terminate and all other threads will terminate too. More details can be found on the following webpage:

https://www.thegeekstuff.com/2012/04/terminate-c-thread/

d) Compiling and running your program on csci3120.cs.dal.ca should not lead to errors or warnings. To compile and run your program on csci3120.cs.dal.ca, you need to be able to access the command-line interface of csci3120.cs.dal.ca. In addition, you need to be able to upload a file to or download a file from csci3120.cs.dal.ca.
   a. To access the command-line interface of "csci3120.cs.dal.ca", several different methods could be used. Here are two widely used methods:
      i. MS Windows Computer: You can use the software tool "putty" on MS Windows computers. With "putty", the connection type should be set to "SSH". Note that "putty" can be downloaded via the following link: https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html .
      ii. Mac and Linux Computer: On Mac and Linux computers, you can use the command "ssh" to access "csci3120.cs.dal.ca" via the program called "Terminal".
   b. To transfer files between your computer and "csci3120.cs.dal.ca", several different methods could be used. Here are two widely used methods:
      i. MS Windows and Mac Computer: Cyberduck is popular tool used to transfer files between two computers via SFTP (i.e. SSH File Transfer Protocol). You can download Cyberduck from the following webpage: https://cyberduck.io/ . This document includes an appendix that provides an introduction to Cyberduck.
      ii. Linux Computer: On Linux computers, you can use the command "scp" to transfer files. Here is a tutorial on the command "scp": https://www.linuxtechi.com/scp-command-examples-in-linux/.
   c. You should upload your program to "csci3120.cs.dal.ca" to verify that it works well on "csci3120.cs.dal.ca". After verifying that your program works on "csci3120.cs.dal.ca". You should submit your assignment in brightspace. For submission details, please proceed to the submission section of this document. Note that the TA will NOT look at your files on "csci3120.cs.dal.ca". Instead, the TA will retrieve your submission from brightspace.

7) Readme File: You need to complete a readme file named "Readme.txt", which includes the instructions that the TA could use to compile and execute your program on csci3120.cs.dal.ca.

8) Submission: Please pay attention to the following submission requirements:

a) You should place "Readme.txt" in the directory where your program files are located.

b) Your program files and "Readme.txt" should be compressed into a zip file named "YourFirstName-YourLastName-ASN5.zip". For example, my zip file should be called "Qiang-Ye-ASN5.zip".
c) Finally, you need to submit your zip file for this assignment via brightspace.

Note that there is an appendix at the end of this document, which includes the commands that you can use to compress your files on csci3120.cs.dal.ca.

4. Grading Criteria

The TA will use your submitted zip file to evaluate your assignment. The full grade is 20 points. Your submission will be evaluated from the following perspectives. The specific rubric used by the marker is included in a separate file.

- "Readme.txt" with the correct compilation/execution instructions is provided. [1 Point]
- Initialization: Buffer is declared, then each buffer item is set to -1. In addition, the semaphores and mutex lock are properly initialized. [2 Points]
- Producer threads:
  - POSIX semaphores are used to check whether the buffer is full/empty. POSIX mutex lock is used to ensure only one thread is allowed to revise the buffer. [3 Points]
  - A number of producer threads are created. The number is equal to the number specified by the user. [1 Point]
  - A producer always sleeps for a random amount of time (0-4 seconds) before generating a new item. [1 Point]
  - Once the sleep is over, a non-negative integer (0 to RAND_MAX) is generated. [1 Point]
  - When allowed, a producer inserts the non-negative integer into the buffer. A producer always inserts a new item into the next empty slot in the buffer. [2 Points]
- Consumer threads:
  - POSIX semaphores are used to check whether the buffer is full/empty. POSIX mutex lock is used to ensure only one thread is allowed to revise the buffer. [3 Points]
  - A number of consumer threads are created. The number is equal to the number specified by the user. [1 Point]
  - A consumer always sleeps for a random amount of time (0-4 seconds) before attempting to remove an item from the buffer. [1 Point]
  - When allowed, a consumer removes an item from the buffer. A consumer always removes the earliest item in the buffer. When an item is removed from the buffer, "-1" is placed into the buffer slot corresponding to the removed item. [2 Points]
- The thread corresponding to the *main()* function sleeps for a period specified by the user and then terminates the C program. [1 Point]
- Proper programming format/style (e.g. release the memory that is dynamically allocated when it is not needed any more; proper comments; proper indentation/variable names, etc.). [1 Point]

Please note that when "Readme.txt" is not provided or "Readme.txt" does not include the compilation/execution instructions, your submission will be compiled using the standard command `gcc -o A5 A5.c (or your filename) –lpthread -lrt`, and you will receive a zero grade if your program cannot be successfully compiled on csci3120.cs.dal.ca.

5. **Academic Integrity**

At Dalhousie University, we respect the values of academic integrity: honesty, trust, fairness, responsibility and respect. As a student, adherence to the values of academic integrity and related policies is a requirement of being part of the academic community at Dalhousie University.

1) What does academic integrity mean?

Academic integrity means being honest in the fulfillment of your academic responsibilities thus establishing mutual trust. Fairness is essential to the interactions of the academic community and is achieved through respect for the opinions and ideas of others. Violations of intellectual honesty are offensive to the entire academic community, not just to the individual faculty member and students in whose class an offence occur (See Intellectual Honesty section of University Calendar).

2) How can you achieve academic integrity?

- Make sure you understand Dalhousie's policies on academic integrity.
- Give appropriate credit to the sources used in your assignment such as written or oral work, computer codes/programs, artistic or architectural works, scientific projects, performances, web page designs, graphical representations, diagrams, videos, and images. Use RefWorks to keep track of your research and edit and format bibliographies in the citation style required by the instructor. (See http://www.library.dal.ca/How/RefWorks)
- Do not download the work of another from the Internet and submit it as your own.
- Do not submit work that has been completed through collaboration or previously submitted for another assignment without permission from your instructor.
- Do not write an examination or test for someone else.
- Do not falsify data or lab results.

These examples should be considered only as a guide and not an exhaustive list.

3) What will happen if an allegation of an academic offence is made against you?

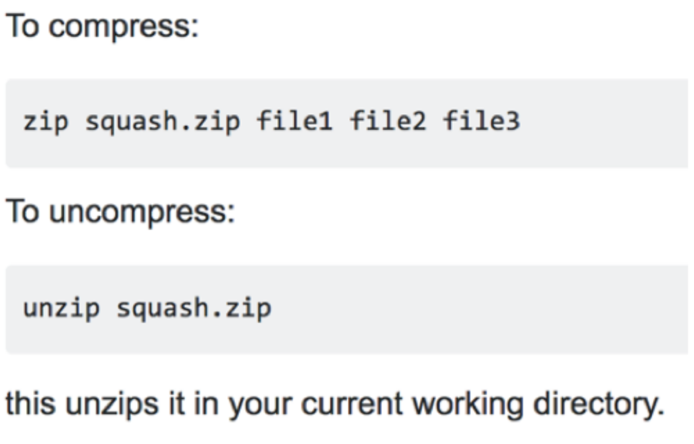
I am required to report a suspected offence. The full process is outlined in the Discipline flow chart, which can be found at:
http://academicintegrity.dal.ca/Files/AcademicDisciplineProcess.pdf and includes the following:
a. Each Faculty has an Academic Integrity Officer (AIO) who receives allegations from instructors.

b. The AIO decides whether to proceed with the allegation and you will be notified of the process.
c. If the case proceeds, you will receive an INC (incomplete) grade until the matter is resolved.
d. If you are found guilty of an academic offence, a penalty will be assigned ranging from a warning to a suspension or expulsion from the University and can include a notation on your transcript, failure of the assignment or failure of the course. All penalties are academic in nature.

4) Where can you turn for help?

- If you are ever unsure about ANYTHING, contact myself.
- The Academic Integrity website (http://academicintegrity.dal.ca) has links to policies, definitions, online tutorials, tips on citing and paraphrasing.
- The Writing Center provides assistance with proofreading, writing styles, citations.
- Dalhousie Libraries have workshops, online tutorials, citation guides, Assignment Calculator, RefWorks, etc.
- The Dalhousie Student Advocacy Service assists students with academic appeals and student discipline procedures.
- The Senate Office provides links to a list of Academic Integrity Officers, discipline flow chart, and Senate Discipline Committee.

Appendix 1: How to Use Zip and Unzip on "csci3120.cs.dal.ca"



To compress:

```
zip squash.zip file1 file2 file3
```

To uncompress:

```
unzip squash.zip
```

this unzips it in your current working directory.

Appendix 2: How to View/Kill Your Processes on "csci3120.cs.dal.ca"

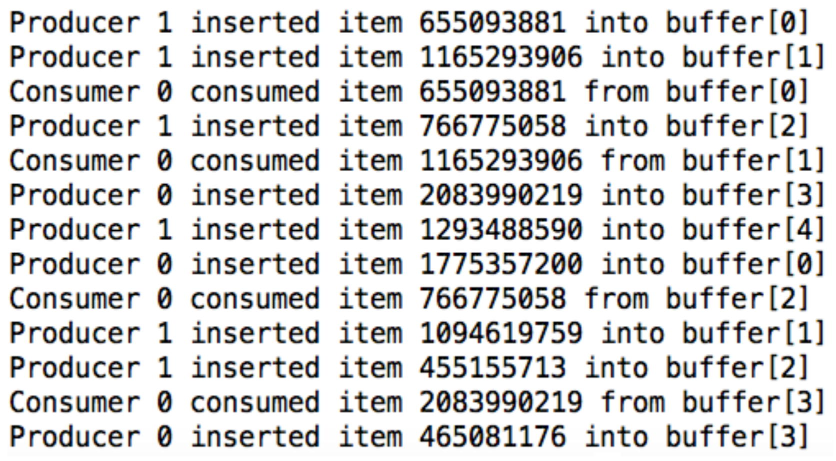
Due to programming mistakes, you might leave a number of running processes on "csci3120.cs.dal.ca". When you leave too many running processes "csci3120.cs.dal.ca", your CS

ID could be locked temporarily (then you need to talk to CS Help to unlock your account). To view/kill your processes on “csci3120.cs.dal.ca”, you can use the following commands. Please keep an eye on your processes on “csci3120.cs.dal.ca”.
1) Command to display the processes belonging to a specific ID (i.e. UID): **ps -u UID**
2) Command to kill a process using process ID (i.e. PID): **kill -9 PID**
3) Command to kill all processed belonging to a specific user ID (i.e. UID): **pkill -u UID**

Appendix 3: Introduction to Cyberduck

1. Optional Registration: Cyberduck is free software that supports file transfers via SSH File Transfer Protocol (i.e. SFTP). You can choose to pay for the registration to avoid seeing the donate/buy window when you quit the program. But it is perfectly ok to use Cyberduck without the registration. The only drawback is that you will see the donate/buy window when it is terminated (in this case, you can click on “Later” to completely terminate the program).
2. Set Up Connection: SFTP (SSH File Transfer Protocol) is used to set up the connection
    a. Start Cyberduck
    b. Click on the icon “Open Connection”
    c. Set Protocol to “SFTP (SSH File Transfer Protocol)”
    d. Set Server to “csci3120.cs.dal.ca”
    e. Enter your CSID and password
    f. Check “Add to keychain” on macOS or check “Save Password” on MS Windows
    g. Click on “Connect”
3. Upload/Download Files: “drag-and-drop" is supported
    a. Uploading: select a file on your computer and thereafter drag the file into the Cyberduck interface
    b. Downloading: select a file in the Cyberduck interface and thereafter drag the file to the proper destination

Appendix 4: Example Output



```
Producer 1 inserted item 655093881 into buffer[0]
Producer 1 inserted item 1165293906 into buffer[1]
Consumer 0 consumed item 655093881 from buffer[0]
Producer 1 inserted item 766775058 into buffer[2]
Consumer 0 consumed item 1165293906 from buffer[1]
Producer 0 inserted item 2083990219 into buffer[3]
Producer 1 inserted item 1293488590 into buffer[4]
Producer 0 inserted item 1775357200 into buffer[0]
Consumer 0 consumed item 766775058 from buffer[2]
Producer 1 inserted item 1094619759 into buffer[1]
Producer 1 inserted item 455155713 into buffer[2]
Consumer 0 consumed item 2083990219 from buffer[3]
Producer 0 inserted item 465081176 into buffer[3]
```