EJ Lilagan, Kyrstn Hall

Dr. Xiaoyu Zhang

CS433 – Operating Systems

13 November 2022

Programming Assignment 4

Submitted Files:

- buffer.h
- buffer.cpp
- main.cpp

How to Compile/Run the Program:

To compile: make
Execute: ./prog4 (number of seconds) (number of producers) (number of consumers)

Features Implemented (via. buffer)

The buffer will include two files, buffer.h and buffer.cpp, to run and track items that are in the buffer array. It is designed to . Furthermore, it will come with private and public variables/functions within the header file to be soon accessed in the buffer.cpp file to reuse the variables needed.

1. Constructor: passes in a size parameter to declare the max_size of buffer. As mentioned in zybook, it defines the BUFFER_SIZE of 5, however as the size variable is used, we will not use BUFFER_SIZE as much. With the following variables (count, front, buffer[0]), they will all include a default value of zero and rear will be -1. Without including buffer[0], it will fail the cppcheck test. It will also include a variable length to be used from the private class for the size of 5 to be accessed.
2. Destructor: Temp will get whatever is in front of the buffer, and it will remove that item until the buffer is empty.
3. insert_item(): Requires two conditions. If there is space inside the buffer, add the item, and return true since it satisfies the condition. Else (if the item is not inserted successfully or if it's full) return false.
4. remove_item(): Requires two conditions. If there is space inside the buffer, remove the item and return true as it satisfies the condition. Else (if the item is not deleted successfully or if it’s empty) return false.
5. get_size(): returns the value of the length.
6. get_count(): returns the value of count within the size of the buffer.
7. is_empty(): checks the count within the buffer. Set true if count is equal to 0, and false if count is greater than zero or less than to 5.

8. is_full(): checks the count within the buffer. Set true if count is equal to 5 (or the length), and false if count is less than 5 or greater than zero.
9. print_buffer(): Prints starting from the front to the rear. If the buffer is empty, it will display “Buffer is empty”. Else, it will display all the items in the buffer, separated by commas.

Features Implemented (via. main)

In main.cpp, it will require a mutex lock (mutex) and two semaphores (full and empty) to be initialized. With the necessary adjustments, it will be beneficial to reduce redundancy of the variables and will be primarily used with the following functions listed down below. Furthermore, the main is imperative to synchronization of the producer/consumer threads given the necessary steps to be implemented.

1. *producer(void *param)
    a. Initializes an item that passes the fourth parameter of the pthread_create() function, and to be inserted into the buffer inside of the while loop.
    b. while(true), proceed to enter the item.
    c. Set a random period of time.
    d. For synchronization, set a sem_wait(&empty) for semaphore when not full.
    e. Next, let the mutex access the critical section using the mutex_lock func from the pthread library.
    f. Thirdly, two conditions will be checked for insertion whether if it has been inserted or when an error is fetched.
    g. Once all has been completed, the critical section ends and will unlock the mutex and release the semaphore (increment.)

2. *consumer(void * param)
    a. Initializes item to be removed within the buffer.
    b. while(true), proceed to read the item.
    c. Set a random period of time.
    d. Similarly, set the synchronization, but set a sem_wait(&full) for semaphore when it’s not empty.
    e. Next, let the mutex access the critical section.
    f. Thirdly, two conditions will be checked for removal whether if it has successfully been deleted or when an error is fetched.
    g. Lastly, the critical section ends and will unlock the mutex and release the semaphore (decrement.)

3. main(int argc, char *argv[])
    a. Get command line arguments argv[1], argv[2], argv[3] (via execute line.)
        i. Initialize three integer values that will be typed on the command line. atoi() will be used for all three integer values to convert the char (or string argument) into an integer.
        ii. Assign sleep_time with argv[1] for step 6.
        iii. Assign pthread with argv[2] for step 3 (number of producers.)

iv. Assign cthread with argv[3] for step 4 (number of consumers.)

b. Initialize buffer and synchronization primitives.
   i. Initialize semaphore empty to buffer size (5).
   ii. Initialize semaphore full to 0.
   iii. Initialize the mutex to NULL.
   iv. Assign producer thread id (tid1) in step 3.
   v. Assign consumer thread id (tid2) in step 4.
c. Create producer thread(s) that passes a unique ID to each producer thread ranging from 1 to 5.
   i. Set integer value 'i' for producer and for loop.
   ii. Set a for loop to loop around the producer number that is entered from step 1 to restrict using other values.
   iii. Initialize arg to a new integer → set a condition to see if it cannot allocate memory, but if not fetched, set arg as the passed value to be sent into the *producer().
   iv. Set attr to prevent default attributes when NULL.
   v. With the pthread_create function, the parameters to be passed will include be tid1 for producer, attr as mentioned, producer (function) to create the thread, (void*) arg to pass the value as the parameter onto the producer function [syntax must be followed as shown for it to work.]
d. Create consumer thread(s)
   i. Set integer value 'j' for consumer and for loop.
   ii. Set a for loop to loop around the consumer number that is entered from step 1 to restrict using other values.
   iii. Set attr to prevent default attributes when NULL.
   iv. In contrast to producer, the fourth parameter will be NULL as it will not change the value but read what will be deleted.
e. Main thread sleep
   i. Set a sleep function to notify the user that the time has been completed.
   ii. Additionally, the sleep_time value that has been inserted from the user will be set as seconds in the sleep function.
f. Exit
   i. Simply destroy the mutex lock and semaphores (empty and full) to close the program.
   ii. Call exit(0) to terminate the program.

Additional:

- BUFFER_SIZE → defined in buffer.h to prevent redundant variables for tracking size, but would then be commented out to make other variables such as size and length significant.
- buffer → global object to use the necessary functions to be used in the buffer.h.
- buffer_item → declare the variable object to be an integer value.

Choice of Data Structure: For this assignment, the buffer will be used similar to a queue. Given that we are given an array of the following functions to keep track of BUFFER_SIZE, it must be

held into account. As mentioned on the *Features Implemented (via. buffer)*, each function used is a prime indication as to what is required to satisfy the outcomes.

Lessons Learned/Re-learned

Initially, we used a multitude of methods to create our pthread_create functions to access the consumer and producer functions, but would get different results from gradescope as it would not be consistent to read in the respective producers/consumers. We relied on rand() % BUFFER_SIZE to keep track of the id's that will track the producers that will be generated onto the compiler, however it will be overwritten with the passed element of the pthread_create, which resulted in us to comment out rand(), and will run successfully. In addition, this assignment gave us a fresh reminder to implement a queue.

References

- .cpp check test
- Initializing an array
- atoi for main using argv[1], agrv[2], argv[3]
- pthread_create from producer function example
- How to allocate memory
- How to pass create threads