CSE 330 - Operating Systems Summer 2021
Project #2

Due: June 6th 2021, 23:59 PM

No late submission accepted

Strictly Individual Projects

Implementing Semaphores

Using the threads, you have implemented, implement semaphores. Since the threads are non-preemptive, you do not need to ensure atomicity of the semaphores (they are already atomic).

Implement the following: (you can either create file called sem.h or write it in your main c file)

1. **Semaphore data structure:** A value field and a queue of TCBs.

2. **InitSem(semaphore, value):** Initializes the value field with the specified value.

3. **P(semaphore):** The P routine decrements the semaphore, and if the value is 0 or less than zero then blocks the process in the queue associated with the semaphore.

4. **V(semaphore):** The V routine increments the semaphore, and if the value is 0 or negative, then takes a PCB out of the semaphore queue and puts it into the run queue. Note: **The V routine also "*yields*" to the next runnable process.** //this is important.

5. Implement a solution to the Producer Consumer Problem with the following settings:
   There are P producers all in while loops that loop N times, each producing 1 item in one loop.
   There are C consumers all in while loops that loop N times each consuming 1 item in one loop.
   Every consumer or producer yields at the end of a loop to the next process.
   Consider that the buffer size is B items.
   You will be given a Queue of numbers, where positive numbers denote the producer ID and negative numbers denote consumer ID
   Arrange your ready queue accordingly
   Start with the first thread in the queue and run it.
   At the end of each loop before yield a consumer should print the following:
   if consumer X is consuming an item then print
   printf"\n Consumer %d is consuming item generated by
   Producer %d",X,Y) else print

printf("\n Consumer %d is waiting\n",X)

At the end of each loop before yield the producer should print the following:
if producer X is producing an item then print
printf("Producer %d is producing item number %d", X,Y)
else print
printf("\n Producer %d is waiting \n", X)

How will you implement P(S) and V(S)?

P(S) requires two operations: a) block when S<= 0 and b) let go when S > 0;

The task (b) is easy as nothing needs to be done except for decrementing the semaphore value.

To perform task (a) for every semaphore you declare you have to create a new queue semQ. Whenever a thread executed P(S) and S <= 0 the TCB of the thread gets deleted from the ReadyQ and inserted at the **end** of the semQ. Then the process calls yield(). In this way if the thread remains in the semQ it never gets executed again.

V(S) requires two operations a) increment S and b) unblock **one** process. To unblock a process, you will deleted the **head** of semQ and add at the **end** of the ReadyQ. This way the processes can perform yield and the unblocked process will execute again.

Test cases

Similar to project 1, we will test your code using input redirection. The first line of each test case file will be of the form:

B,P,C,N

Where B is the buffer size, P is the number of producers, C is the number of consumers, and N is the number of times producers and consumers run their while loop. (Note all producers and consumers run the same number of while loop).

This line will be followed by P+C numbers (one number in each line). Each number denotes process ID. Positive numbers indicate producer process ID, while negative numbers indicate consumer process ID.

Example test case
2,3,1,2
1
-1
2
3

Example test case output

For this particular test case we have buffer size 2, 3 producers and 1 consumer, and each producer or consumer executes their while loops 2 times.
We have two semaphores here: a) Full (for the Consumer) and b) Empty (for the Producer).

Initially the ready queue looks like the following

ReadyQ → 1 -1 2 3

Our Buffer is initially empty

The Full and Empty queues remain empty

The first producer produces an item and prints

Producer 1 is producing item number 1

After this print the producer yields so the new ready queue looks like the following

ReadyQ → -1 2 3 1

Buffer → P1 __

The first consumer then executes and prints

Consumer 1 is consuming item generated by Producer 1

The ready queue now looks like the following

ReadyQ → 2 3 1 -1

Buffer → __ __

The second producer then executes.

Producer 2 is producing item number 1

Then it yields

ReadyQ → 3 1 -1 2

Buffer → P2 __

Producer 3 then executes

Producer 3 is producing item number 1

ReadyQ → 1 -1 2 3

Buffer → P2 P3
Producer 1 then attempts to produce but the buffer is full so has to wait

Producer 1 is waiting

EmptyQ → P1

ReadyQ → -1 2 3

Consumer 1 then executes. It unblocks P1 from empty queue and hence P1 joins the end of the ready queue. It then prints:

Consumer 1 is consuming item generated by Producer 2

After printing it exits because it is done.

EmptyQ → empty

ReadyQ → 2 3 1

Buffer → P3 __

Process P2 executes

Producer 2 is producing item number 2

Then producer 2 is done so it exits
ReadyQ → 3 1
Buffer → P3 P2

Producer 3 then executes but is blocked because buffer is full

Producer 3 is waiting

EmptyQ → P3

ReadyQ → 1

Producer 1 then executes

Producer 1 is waiting

The Ready Q is empty so

The program ends

SO overall output is as follows

Producer 1 is producing item number 1

Consumer 1 is consuming item generated by Producer 1

Producer 2 is producing item number 1

Producer 3 is producing item number 1

Producer 1 is waiting

Consumer 1 is consuming item generated by Producer 2

Producer 2 is producing item number 2

Producer 3 is waiting

Producer 1 is waiting

Submission and Grading:

Your project must consist of 5 files

1. TCB.h (uses ucontext.h)
2. q.h (includes TCB.h)
3. threads.h (includes q.h)
4. sem.h (includes threads.h) if you have written one.
5. proj-3.c (includes threads.h)
   (make sure the compile command, "gcc proj-3.c" does the correct compilation).

All 5 files are to be ZIPPED into one zip file and uploaded in Canvas. The name of this file should reflect the name of the student (abbreviated, do not make it too long).

Note: Grading is on Ubuntu. It is in your interest to make sure the program compiles and runs in the target test platform.

Testing your code on grading test cases

Grading test cases are different from the test cases provided.

To test your code against grading testcases follow these steps:

Put the four files q.h tcb.h threads.h and proj-3.c in one zip file. The resulting file should be YOUR LAST NAMEProject2.zip

Make sure you do not have a folder inside the zip file

You can test your code in Gradescope. Just upload the zip file with all four files in it. Make sure your main code is written in proj-3.c. Then upload the zip so that all four files are uploaded. Gradescope will test your code by using the following compilation method:

gcc proj-3.c

It will then test against the grading test cases and provide you with test results