COP 5615: Distributed Operating Systems Principles
Fall 2016
Professor Sumi Helal

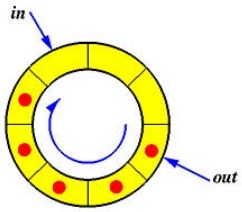
Lab 1: Process Coordination using Sempahores & Mutexes

Due:

I. Objectives

- Learn about a classical synchronization problem known in concurrent processing. Specifically, you will learn about the consumer-producer problem.
- You will implement solutions to this problem under Xinu semaphores, along with mutexes of your own design.

II. The Producer-Consumer Problem



Two processes are sharing a circular buffer (queue), one produces at the tail of the circular queue (the Producer) and the other consumes from the head (the Consumer).

(a) Analyze what can go wrong considering that the two processes are sharing access to the buffer, and are running at possibly different speed (priorities), for varying bursts of CPU processing times. Then, based on your analysis, consider all possible problems or faulty situations that may arise and impact the Producer, the Consumer, or both.

(b) You are asked to implement a mutex in Xinu, using only Xinu semaphores (the semaphore's count must always be either 0 or 1 to be a proper mutex). A functional mutex consists of two methods: acquire and release. A mutex is acquired to start a critical section; this is when no other processes can run code that also requires that mutex – they will wait instead. Once the critical section ends, the mutex is released, and other processes can continue if they were waiting to acquire the mutex themselves.

(c) Using the mutex functionality you just created, implement the producer-consumer problem correctly (meaning, no problems or faulty situations). You are to use the provided *main.c* to define and allocate a shared circular buffer, and then create both the producer and the consumer processes. Note that more than one mutex may be required. You should program both processes with "simulation code" to actually produce or consume. Both processes should output to the console what they are producing or consuming.

(d) Use Xinu semaphores *properly* (counting up and down higher than 1) to implement the producer-consumer problem correctly and effectively. You are to use a copy of your *main()* form part (c), replacing the *producer* and *consumer* functions. Note that a mutex may still be required along with the proper use of semaphores. Be sure that your implementation isn't

faulty to critical case scenarios (locking and the like). You should program both processes with "simulation code" to actually produce or consume as in part (c). Both processes should output to the console what they are producing or consuming.

(e) Consider the differences between the two implementations. How efficiently are the processes able to cooperate before one is required to wait?

(f) You may have noticed that there is a timing function within *main.c*. You are asked to compile a graph plotting the amount of samples consumed against the amount of time taken for part (c) and (d), with 100, 200, 300, 400, and 500 samples, to compare the performance of the mutex version and the semaphore version. By uncommenting the *time_and_end()* function, at the end of execution, the amount of time taken for each amount of samples can be seen. Consider the performance of each method; why might one be faster than the other?

III. Grading

Submit a tar or zip of your two *main.c* files, and a short pdf report to canvas by the deadline, including:

- A list of potential problems and faulty situations that may arise in a producer-consumer scenarios without using any coordination or synchronization mechanisms (1 to 2 lines each)
- The timing graphs for parts (c) and (d) above (one chart with two plots one for c and one for d, against X axes being 100-500.
- Conclusion.