Simon Owens
Operating System CS470
Thread Project

High-level Design

The python script creates multiple ‘producer’ and ‘consumer’ threads. Producer threads add random numbers to a list. Consumer threads attempt to remove random numbers from this same list. Since all threads are attempting to remove and add elements to the list at potentially same time, a mutex lock is used so that the buffer_item can only be altered by one thread at a time. This program loops for a given amount of time specified by the user. The usage of this program is the following:

python3 thread.py <sleep-time> <producer-threads> <consumer-threads>

The main design components of this program are:

- Main
- Buffer_item class
  - Insert Item function
  - Remover Item function

The main function must receive three arguments from the user. It then must create the producer and consumer threads specified by the user. Once the producer threads are created, they will execute the producer function. Likewise, the consumer threads will execute the consumer function after creation. Both of these functions loop until the thread is terminated by the main program. The producer function calls the “insert_item” function and the consumer function calls the “remove_item” function.

The Buffer_item class holds a list and a value for its maximum length. This class has an “insert_item” function that attempts to place a random element in the last if there is enough space. The class has a “remove_item” function that attempts to remove a random element, if it exists, from the list. A mutex lock exists in both functions so that the list can be altered by several different threads but not get into any race conditions.

Additional Questions

1) It was most difficult for me to understand how to use “thread” library functions when having the thread execute different functions. I may want the thread in another function to print out its name, but couldn’t figure out how to do this unless I pass that function the threads name (or making the thread a global variable). It was also weird that threads don’t share memory (but makes sense).
2) We just went over processes, so threads were pretty easy to understand.
3) It was difficult for me to understand the different ways the “locks” worked. Especially the binary semaphore. I’m still learning how to initialize and use different locks correctly.
4) It was easy for me to understand that once in the Critical section, no other thread should be allowed in. There could be a race conditions if two threads were attempting to remove the same element and add the space element to the list, all at the same time.

5) My current design passes around too many parameters. Also, the input should be sanitized. I only checked that the user input the correct amount of arguments. My program also shouldn't just call "sys.exit". Sometimes a thread gets killed in the middle of an operation which throws a lot of errors upon exit. I need to find out some way to safely kill all of the processes.
6) I found it dealing with the threading library extremely frustrating and surprising. I figured you could just use like "this", "threading.get_id" or "self" to easily identify properties about the thread executing the current code.