Nicholas Everekyan, Kevin Gallagher

CS433: Operating Systems

Professor Zhang

November 9th, 2024

Programming Assignment 4 Report

Problem Description:

We were tasked with implementing a semaphore solution to the Producer Consumer problem. The program itself consisted of a small buffer to which a differing number of producing and consuming processes would attempt to access and modify. The difficulty in this was ensuring that no process would attempt to remove from an empty buffer, or add to a full buffer, or access it at the same time as some other process, which would result in corrupted data.

Program Design:

We went about solving this issue using the PThread Library, as well as condition variables, which are essentially semaphores, for our solution. As such, this is the monitor solution to the Producer Consumer problem, as all synchronization occurs within the buffer functions itself rather than the producers or consumers. When attempting to add an item to the buffer, the thread locks the mutex lock, waits if the condition variable is full, and once it isn't, pushes a new item to the buffer. Finally, it ends by signaling that the buffer is not empty, and unlocks the mutex lock. The remove process is essentially the same, only waiting if the condition variable is empty, updating the not full condition, removing the item from the buffer and passing it by reference to the input variable, and unlocking the mutex lock. For safety as well, the get_count() function also locks a mutex lock to use the buffer.size() function. As all synchronization is done within the buffer itself, there is no need to implement any synchronization within the main file itself.

Analysis:

We chose the monitor approach to help alleviate busy waiting, as with conditional variables, the pthread_cond_wait() function will release the mutex and suspend the thread until it is signaled later on; were we to approach this through doing all the synchronization within the producer and consumer processes, busy waiting could have resulted from the semaphores in essentially similar while statements. Furthermore, the monitor approach minimizes the amount of code needed, as instead of having code in both producer and consumer processes running, all the overhead is instead taking place in the buffer itself. In this scale, the differences are negligible, however, if this was scaled to tens of thousands of processes, it would not be as negligible.

Conclusion:

Our program successfully implements and solves the Producer Consumer Problem through a use of monitors and conditional variables. This assignment helped highlight the nuances and complexities of running synchronized processes together, as well as properly implementing a concise way of managing access to shared data. A possible area of improvement could be to scale this up to a much larger scale. A buffer with 5 slots doesnt seem very practical in today’s world.