Design

Our intention in this program was to create an efficient means for process management. In our best estimation (in comparison to other projects) our process management algorithm is more efficient.

Often efficient code results in a lower amount of coding thus improving readability. This is the general layout of our code:

Main():
1. Argument check
2. Open file
3. While unless exit
   a. Input command (var:option)
   b. Course of action dependant upon command
      i. Case 'c'
         1. Input pid and burst
         2. Push the process on the Queue
         3. Write std output of process on Ready Queue
         4. Write std output of process Running
      ii. Case 'd'
         1. Input pid
         2. Run Delete function on both Ready and Wait Queue
            a. Check for existent processes
            b. Cycle to check queue for marked kill
            c. Call delete recursively to kill children
            d. Write std output of process killed
         3. Write std output of process running
      iii. Case 'I'
         1. Write std output of process running
      iv. Case 'w'
         1. Input event number
         2. Write std output of process to send to Wait Queue
         3. Move process to temp then push to Wait Queue
         4. Std output of process running
      v. Case 'e'
         1. Input event number
         2. Assign waitID for process to return to Ready Queue
         3. Write std output of process running
         4. Write std output of process placed on Ready Queue
      vi. Case 'x'
         1. Check for running, ready, and waiting processes
         2. Write std output of all remaining processes
            a. Running process = ReadyQ[0]
               i. If it exists
                  1. Output process with burst
               b. Ready Queue = ReadyQ[1:ReadyQ.size()]
                  i. If a process exists
                     1. For all processes left
                        a. Output process with burst
               c. Wait Queue = WaitQ[0:WaitQ.size()]
                  i. If a process exists
                     1. For all processes left
Output with burst

1. Execute Program
2. Insert processes into Ready Queue
   a. Assign waitID (to be assigned in process)
      i. Move process to back of Ready Queue
      ii. Find process
   b. Output to a file
5. Exit program

Analysis

It’s safe to say not all of our programming practice is what it should be if we were to publish it as an actual process management program.

The greatest cause for inefficiency in most projects is the cycle to find a process and delete it or send it to the wait queue. Using a <vector> STL instead of a <queue> STL is quite advantageous because the vector STL has a function that allows you to delete from any point in the queue and move all cells forward without losing them in memory space. This is important because with a <queue> STL one would have to pop off from the front and push to the back to check all processes that appear in head for a marked kill. It’s obviously less processor intensive to run a FOR loop cycle to find the marked kill than to run two functions and check for the marked kill. This can be seen at Case ‘d’: 2(a,b.c.d).

Our class definition is simple in that all parts are public. This made programming easier for us but probably could have been made more secure by declaring variables as private within the class. We only have 1 member function of the process class, the constructor. We could have included the Delete and DeleteW functions but chose not to for simplicity.

The delete function was the toughest of adversaries to overcome. We had two ideas of ways to accomplish this task. The first was to create a “kill” queue. Here we would push the ID of the process to be killed and search both queues for the processes that were children of the process to be killed. When another one was pushed on the process was repeated until no more children were found. In essence we found this to be a recursive function, and creating a queue was more work than needed to be done. Thus we decided to opt for choice two which was to just use a recursive delete function.

In looking at our Design and thinking back to the programming it seems to us that simplicity ended up being our highest priority. It may be by chance that we stumbled onto efficient code. As we think more about the <vector> STL it seems to us that we owe our efficiency and simplicity both to it.