

Concurrent Programming CW2 Submission Overview

In this coursework I attempted and completed stages 1 and 2. I tried to make headway with the disk option of stage 3 but was making very little progress for the amount of time I was spending on it so discontinued that section.

Stage 1a

This stage was relatively simple given that the lab work covered most of the work that needed to be done here.

Stage 1b

This stage required more effort. Initially I implemented a round-robin that iterated over my PCB table to accomodate a flexible max number of initial processes for the generalised implementation of stage 1a.

For a priority scheduling algorithm to improve this, I chose to use a multi-level feedback queue. This comprised of 3 linked lists of differing priorities, with the nodes in the lists associated with processes in the PCB table that are ready to be executed. The process priority can be edited but is defaulted to 1, so all processes are initially placed in the highest priority queue. Lower priority queues allow programs more time to execute before switching. If a program does not finish executing before its allocated time slot it is moved to a lower priority queue. The top two queues are FCFS queues and the bottom operates as a round robin.

In theory, this scheduling system allows for very quick processes like I/O to be initialised and get executed very quickly, staying in the highest priority queue, whilst slower processes are given longer time slots at a lower priority. In the case of this coursework the processes are fairly slow and so the scheduler quickly evolves into a round-robin scheduler if no new processes are added.

Stage 2

This stage took me a very long time. Dealing with the technicalities of forking processes and assigning values to the correct context and register spaces was a large hurdle for me to overcome. However, once I understood the use of memory I was able to finish this section, with the Dining Philosophers problem not too difficult to implement once I had the forking and executing system calls working properly.

My choice of IPC was semaphores; this IPC lends itself nicely to the implementation of Dining Philosophers in the form of mutexes being used to lock and unlock access to the forks. My solution to Dining Philosophers uses the rule that a philosopher must only pick up the lowest index fork first. By putting the forks (read semaphores) in an array with a lowest and highest index, it infers that all philosophers except one can pick up a fork straight away, thus preventing deadlock. Starvation is avoided by using randomised thinking and eating times.

In the first image below you can observe the use of the execute and terminate console instructions to dynamically start and end the execution of processes. Note that P5 stops without the need of a prompt because it finishes execution on its own.

In the second image you can observe the Dining philosophers problem working, with all 16 sub-processes getting initialised and getting their turn to “eat” - this is signified by the printing of the philosopher’s index.

[illegible]

Activities Terminal

File Edit View Search Terminal Help

pulseaudio: set_sink input_mute() failed

pulseaudio: Reason: Invalid argument

Concurrent Programming CW2 Submission Overview

01233567881011112131414111133555881010101212121414111133333555888810101010121212121414141415151515153

333377779999912121213131313131515151533336666688881111111111212121214141414143333355555777771010

101010121212121313131314141414333344444666669999111111111121212121214141414142222444455555888881

0101010101212121212141414141111133333444447777799991212121313131313111122222444466666888881111

111113313131314141414140000002222444466667777711111111113131313131414141414000044445555577771111111

111414141400000333335556666111111113131313141414141400002222444410101010121212121313131313151

515151511114444999991111111131313131300000444888881010101013131313141414141515151515333388

88999991212121213131313131515151522227779999111111111313131315151515222666668888101010101

101313131315151515111666669999131313151515156666888812121212151515156666777711111111

414141414555577771010101010131313141414141444446666610101010121212121214141414444499999121

2121214141414444888881111111114141414333338888101010101313131314141414143333888812121212121

3131313141414141422222888811111111114141414141422227777101010101111111114141414222266

6669999101010101313131314141414142225555888889999912121212131313131414141411114444477778

888811111111121212121414141411113333366667777101010101111111114141411115555666669999913131

1010101313131313111444448888101010101313131414141414000004447777999991212121213131313130000

3333377788881111111111313131315151515333366667777101010101313131322225555666669999913131

31313111114444666688881313131311113333666677771313131311115555666661313131311144444131313

13131414141414000004441212121212141414151515153333121212121313131313222211111111131313131

41414141414111101010101212121214141414000009999911111111131313131414141400008888101010101

0121212121313131314141414888811111111121212121414141488881010101011111111414141415151

51515888810101010131313131414141488881212121214141414888811111111114141414888811111111

1131313131414141414

Output/messages

This preventing deadlock. Starvation is avoided by using randomized times.

tmux

Stage 3

Although I didn't make much material progress in this section, attempting to create a file structure was a fun project and I will probably continue to explore it outside of the coursework.