CSC 4103: HW 2

Class	CSC 4103
Date Created	@March 3, 2023 5:39 PM
Status	Done 🙌

Ashwin Nair

Dr. Feng Chen

CSC 4103

1. Describe the differences between process and thread.
    a. Starting off, we have memory allocation. A process has its own memory space and system resources, while a threads share the same space and resources **within** a process. Threads also context switch faster since they only have to save then restore the state of the thread. However, processes have to save the current state of the process and then restore the state of the thread. Similarly, thread synchronization is also easier since threads have the same memory space.
2. Explain the advantages and disadvantages of user threads and kernel threads.
    a. User threads are lightweight and flexible but not that much parallelism. They can block the entire process if blocked. Kernel threads have better parallelism but also have more overhead and less customization overall.
3. Consider the exponental average formula used to predict the length of the next CPU burst. What are the implications of assigning the following values to the parameters used by the algorithm?
    - Formula:
        - $tn+1 = a * tn + (1-a) * bn$
        - tn is the predicted value of the next CPU burst time after the nth burst
        - bn is the actual length of the nth cpu burst
        - a is a weight factor, which we will use to adjust the influence of past values on the predicted value.
        - t0 is the initial predicted value before any CPU burst times are known
    1. a = 0 and t0 = 100 milliseconds
        a. If a = 0 and t0 = 100 milliseconds, then the prediced value (or t0) is 100 milliseconds. The past CPU history isn’t taken into account, and the t0 remaining constant the entire time does not help produce accurate predictions and may not yield the best scheduling decisions
    2. a = 0.99 and t0 = 10 milliseconds
        a. If a = 0.99 and t0 = 10 milliseconds, then the CPU burst time will be taken into account, and will be heavily influenced, by the past history of CPU burst times. Weight (a) of 0.99 is being assigned to the most recent burst time, and a weight of 0.01 is being assigned to the initial predicted value (t0). The predicted value will adjust over

time in the pattern of burst times, and will be well versed to deal with short-term issues. This is viable for accurate predictions over a longer period of time, but not for process with burst times with no overall pattern that it abides to.

4. Consider the following set of processes, with the length of the CPU burst time given in milliseconds:

Process	Burst Time	Priority
P1	2	2
P2	1	1
P3	8	4
P4	4	2
P5	5	3

The processes are assumed to have arrived in the order P1, P2, P3, P4, P5, all at time 0.

a. Draw Four Gantt charts that illustrate the execution of these processes using the following scheduling algorithms: FCFS, SJF, non-preemptive priority (a larger priority number implies a higher priority), and RR (quantum = 2).

a. FCFS (First come, first serve):

P1	P2	P3	P4	P5
2	1	8	4	5

b. SJF (Shortest Job First):

P2	P1	P4	P5	P3
1	2	4	5	8

c. RR (Round Robin) with a quantum of 2:

P1	P2	P4	P5	P5	P4	P3
2	1	2	2	2	2	6

d. Non-preemptive Priority:

P1	P4	P5	P2	P3
2	4	5	1	8

d. What is the turnaround time of each process for each of the scheduling algorithms above?

A. Turnaround Times

Process	FCFS	SJF	Non-Preemptive Priority	RR
P1	2	2	2	6
P2	3	1	3	5
P3	16	8	21	13
P4	20	5	18	10
P5	25	10	23	12

e. What is the waiting time of each process for each of these scheduling algorithms?

A. Waiting Times

Process	FCFS	SJF	Non-Preemptive Priority	RR
P1	0	0	0	4
P2	2	0	2	3
P3	6	0	13	0
P4	10	1	14	14
P5	14	5	18	5

5. Different synchronization methods have advantages and disadvantages. In the following scenarios, describe which is a more suitable synchronization method, spinlock or semaphore. Explain the reason for your choice and why the other is not chosen.
   a. Increment and decrement a shared 16-bit counter in memory.
      i. Here, I would use a ***spinlock***. Since we're dealing with a short critical section, a spinlock would help us avoid context switching since we are using busy-waiting to implement (this means a thread constantly checks the lock status in a tight loop). We would not use a semaphore because for short critical sections, spinlocks are better, since it does not consume a relevant number of CPU cycles.
   b. Write a message to the log file and use *fsync* to wait for the data written to a magnetic disk.
      i. Here, I would use a ***semaphore***. Writing to a log fiie and then waiting for it to be written onto the magnetic disk could be a time-taking task. It could take many miliseconds to finish, so using a spinlock would consume unnecessary resources and would consume a significant number of CPU cycles. This would be because using a spinlock would occupy the CPU by constantly checking if the lock status, instead of allowing it to go and perform other tasks.
6. Consider the following snapshot of a system with 5 processes and 4 resource types:
   a. What is the content of the Matrix *Need*?
      i. Matrixes

Allocation ABCD	Max ABCD	Need ABCD
0012	0012	0000
1000	1750	0750
1354	2356	1002
0632	0652	0020
0014	0656	0642

   b. Safe State Check
      i. To check if the system is in a safe state with the banker's algorithm, we will first initialize the Available vector to the values given:
         `Available = 15 20 20 15`
      ii. Then create a work matrix, which we will also initialize to the values given for the Available section:
         `Work = 15 20 20 15`
      iii. Next, we make a boolean array called Finish to track what processes are done:
         `Finish = false false false false false`
      iv. Since P1 has no outstanding needs, we can start with it. We add its allocation to the Work matrix, mark p1 as finished in the `Finish` array, then release its resources:
         `Work = 15 20 21 17`
         `Finish = true false false false false`
         `Available 0 1 -1 -2`
      v. Next, we can do the same thing with P4, since it has a need of 0 0 2 0, and the Work matrix has atleast 2 units of resource C. We will now add its memory allocation to the Work matrix, then release its resources:
         `Work = 15 26 24 19`
         `Finish = true false false true false`
         `Available = 0 7 -1 -2`
      vi. We can keep doing this until the all of the entries in the boolean array called `Finish` are marked `true`. The safe order of execution would be P1, P4, P4, P5, P2. Since there is a safe order of execution, the system is in a ***safe state.***