Homework Assignment № 2

Steven Seppala 26 October 2014

Homework assignment 2

1. Problem 6.14

   (A) With α and τ_0 both equal to zero. This will imply τ_{n+1} = τ_0 = 100 ms. This therefor implies that the recent history has no effect and that the formula will always predict 100 ms for the next CPU burst.

   (B) The most recent behavior of the process is given much higher weight than the past estimated value. Consequently, the scheduling algorithm is almost memory-less, and simply predicts the length of the previous burst for the next quantum of CPU execution.

2. Problem 6.16

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

a. The four Gantt charts are


 _____________________________________________________
 |           P1            | P2|  P3  |P4 |     P5      | FCFS
 |_________________________|___|______|___|_____________|
 0                        10  11     13  14            19

 _____________________________________________________
 |P2| P4|  P3  |      P5     |            P1             | SJF
 |__|___|______|_____________|___________________________|
 0  1   2      4             9                          19

There is a tie between the priorities of P1 and P3. If P1 is scheduled
first:
 _____________________________________________________
 |P2|      P5      |           P1            |  P3  | P4| Priority13
 |__|______________|_________________________|______|___|
 0  1              6                        16     18  19

or if for the tie between P1 and P3 priorities, P3 is scheduled first:

 _____________________________________________________
 |P2|      P5      |  P3  |           P1            | P4| Priority31
 |__|______________|______|_________________________|___|
 0  1              6      8                        18  19

 ___________________________________________________________________
 |P1| P2| P3| P4| P5| P1| P3| P5| P1| P5| P1| P5| P1| P5|     P1     |RR
 |__|___|___|___|___|___|___|___|___|___|___|___|___|___|____________|
 0  1   2   3   4   5   6   7   8   9  10  11  12  13  14           19
```

b. The turnaround time of each process for each of the scheduling algorithms in part a:

Turnaround_time = Finish_time - Arrival_time

	FCFS	SJF	Priority1	Priority3	RR
P1	10	19	16	18	19
P2	11	1	1	1	2
P3	13	4	18	8	7
P4	14	2	19	19	4
P5	19	9	6	6	14

c. The waiting time of each process for each of the scheduling algorithms in part a:

Waiting time (turnaround time minus burst time) =
Finish_time - Arrival_time - Burst_time

	FCFS	SJF	Priority13	Priority31	RR
P1	0	9	6	8	9
P2	10	0	0	0	1
P3	11	2	16	6	5
P4	13	1	18	18	3
P5	14	4	1	1	9

d. The schedule in part that results in the minimal average waiting time (over all processes):

	FCFS	SJF	Priority1	Priority3	RR
Ave. wait	9.6	3.2	8.2	6.6	5.4

So the answer is Shortest Job First.

3. Problem 6.23

   With $\beta > \alpha > 0$; this implies a FIFO.

   With $\beta > \alpha > 0$; this implies a LIFO.

4. Ch 6. About UNIX scheduling

   Given that base = 60:
   $P_1 = \frac{40}{2}$+ 60=80
   $P_2 = \frac{18}{2}$+ 60=69
   $P_3 = \frac{10}{2}$+ 60=65

Thus the new priorities are : P_3 first P_2 second, and then P_1 last.

5. Problem 5.23

1.Initialize semaphore to $MAX\_SOCKETS$

2.A new incoming connection will wait() on the semaphore

a.If the semaphore's value is less than 0, block

b.If the semaphore's value is greater than or equal to 0, accept connection

3.When a connection is closed, signal() the semaphore

4.Go back to 2 for possible new connections

6. Problem 5.23

```
1  int guard = 0;
2  int semaphore value = 0;
3  wait ()
4  {
5      while (TestAndSet(&guard) == 1);
6      if (semaphore value == 0)
7      {
8          atomically add process to a queue of processes waiting for ↩
               the
9              semaphore and set guard to 0;
10     }
11     else
12     {
13         semaphore value--;
14         guard = 0;
15     }
16 }
17 signal()
18 {
19     while (TestAndSet(&guard) == 1);
20     if (semaphore value == 0 && there is a process on the wait queue)
21         wake up the first process in the queue of waiting processes ↩
               else
22             semaphore value++;
23     guard = 0;
24 }
```

7. Problem 5.17

For a short duration lock, it would be better to use a spinlock.

If the lock is going to be held for a long time, it should be a mutex lock which blocks while it sleeps waiting for it to become available.

If a thread may be put to sleep while holding the lock, it is better to use a mutex lock so that another thread may not be accidentally allowed into the sensitive code.

8. Problem 7.16

    (a) Safe anytime.

    (b) Safe only when Max $\leq$ Available.

    (c) Safe only when Max $\leq$ Available.

    (d) Safe anytime.

    (e) Safe anytime.

    (f) Safe anytime.

9. Problem 7.17

    Proof by contradiction:

    Suppose the system is deadlocked. This implies that each process is holding one resource and is waiting for one more. Since there are three processes and four resources, one process must be able to obtain two resources. This process requires no more resources and, therefore it will return its resources when done.

10. Problem 7.23

    (a) The values of Need for processes P_0 through P_4 respectively are:
    (0, 0, 0, 0), (0, 7, 5, 0), (1,0, 0, 2), (0, 0, 2, 0), and (0, 6, 4, 2).

    (b) Yes. With Available being equal to (1,5, 2, 0), either process P_0 or P_3 could run. Once process P_3 runs, it releases its resources, which allow all other existing processes to run.

    (c) Yes, it can. This results in the value of Available being (1, 1, 0, 0). One ordering of processes that can finish is P_0, P_2, P_3, P_1, and P_4.