Solution

CO 502 – Operating SystemsTutorial: Deadlock & Scheduling

Morris Sloman

Deadlock

1. Suppose that there is a resource deadlock in a system. Give an example to show that the set of processes deadlocked can include processes that are not in the circular chain in the corresponding resource allocation graph.

   Consider three processes A, B and C, and two resources R and S. Suppose A is waiting for R that is held by B, B is waiting for S held by A, and C is waiting for R held by B. All three processes, A, B and C are deadlocked. However, only A and B belong to the circular chain.

2. Consider a system that uses the banker's algorithm to avoid deadlocks. At some time a process P requests a resource R, but is denied even though R is currently available. Does it mean that if the system allocated R to P, the system would deadlock?

   No. An available resource is denied to a requesting process in a system using the banker's algorithm if there is a possibility that the system may deadlock by granting that request. It is certainly possible that the system may not have deadlocked if that request was granted.

3. Two processes, A and B, each need three records, 1, 2, and 3, in a database. If A asks for them in the order 1, 2, 3, and B asks for them in the same order, deadlock is not possible. However, if B asks for them in the order 3, 2, 1, then deadlock is possible. With three resources, there are 3! = 6 possible combinations each process can request resources. What fraction of all combinations is guaranteed to be deadlock free?

   Answer: Suppose that process A requests the records in the order 1,2, 3. If process B also asks for 1 first, one of them will get it and the other will block. This situation is always deadlock free since the winner can now run to completion without interference. The other four combinations can be similarly reasoned about and shown to lead to possible deadlock:

   (1) 1 2 3*: deadlock free*

   (2) 1 3 2*: deadlock free*

   (3) 2 1 3*: possible deadlock*

   (4) 2 3 1*: possible deadlock*

   (5) 3 1 2*: possible deadlock*

   (6) 3 2 1*: possible deadlock*

   So only one third of the cases are guaranteed to be deadlock free.

4. Can a single-processor system have no processes ready and no process running? Is this a deadlocked system? Explain your answer.

   In theory, we might have a single blocked process waiting for an I/O completion; when the completion occurs, the blocked process proceeds to the ready state (or even directly to the running state). However, in practice, most architecturess require a valid process to execute at all times. Therefore, many operating systems assign an idle process created by the kernel to execute when no other processes are ready. Although this system may appear not to be doing anything, it is not deadlocked in the sense that as soon as the I/O completion interrupt arrives, the system will indeed resume operation.

Scheduling

5. (a) Give an example showing why FCFS is not an appropriate scheduling scheme for interactive users.

   The assumption is that with FCFS, once a process is initiated it runs to completion. Such a scheme would prevent the system from guaranteeing good interactive response times. For example, suppose a large batch process enters a uniprocessor system. While that process executes, no other processes can execute. A user that attempts to load a web page or send an instant message must wait until the batch process completes before the system will respond to those requests.

   (b) Using the example from (a), show why round-robin is a better scheme for interactive users.

   Round-robin is a preemptive scheme that makes use of the interrupt clock. Long processes cannot delay shorter ones, because the shorter ones are assured of getting the processor periodically. Interactive users will thus receive the processor frequently enough to maintain good response times. In our example, the large batch job will be interrupted to service the processes that try to load a web page or send an instant message.

6. In this problem you are to compare reading a file using a single-threaded file server and a multithreaded server, running on a single-CPU machine, using Kernel level threads.
   It takes 15 ms to get a request for work, dispatch it, and do the rest of the necessary processing, assuming that the data needed are in the block cache. If a disk operation is needed, as is the case one-third of the time, an additional 75 ms is required, during which time the thread sleeps. How many requests/sec can the server handle if it is single-threaded? If it is multithreaded?
   Assume that thread switching time is negligible and consider:

   (a) a non-preemptive scheduler,

   (b) a preemptive round robin scheduler with a small quantum.

   a) Processing time = 15 ms if data in cache or 15+75 = 90 ms, if disk access needed.

   Cache hit probability = 2/3, and probability disk access required = 1/3.

   *In the single-threaded case, weighted average access time = 2/3*15 + 1/3*90 = 40 ms. Thus the mean request takes 40 ms and the server can do 25 req/s.*

   For a multithreaded server, try to overlap I/O time with other threads accessing cache, so require minimum 75/15 + 1 = 6 threads. For at least 6 threads and we assume a cooperative (non-preemptive) scheduler, all the waiting for the disk can be overlapped, so every request takes 15 ms, and the server can handle 1000/15 = 66 2/3 requests per second.

   *b) Assume a preemptive round-robin scheduler with a small quantum > 25ms (not critical as not really used in solution). Each request needs, on average, 15ms CPU time + 1/3*75 = 25 ms I/O time. Total time = 15 + 25 = 40ms. The probability that all n threads are waiting for I/O is $(25/40)^n = (5/8)^n$. Thus the CPU utilisation is $1 - (5/8)^n$.*
   During 1000ms, the CPU handles $(1 - (5/8)^n) * 1000/15$ requests.
   *For n = 1 (i.e. single threaded) we get 3/8 * 1000/15 = 25 req/s, as expected.*
   For n = 2 we get 40.62 req/s, for n = 6 we get 62.69 req/s, etc.

7. Five jobs are waiting to be run. Their expected run times are 9, 6, 3, 5, and X. In what order should they be run to minimize average turnaround time? (Your answer will depend on X.)

   Shortest job first is the way to minimize average turnaround time:

   - $0 < X \leq 3$: X, 3, 5, 6, 9
   - $3 < X \leq 5$: 3, X, 5, 6, 9.
   - $5 < X \leq 6$: 3, 5, X, 6, 9.
   - $6 < X \leq 9$: 3, 5, 6, X, 9.
   - $X > 9$: 3, 5, 6, 9, X.

8. Five batch jobs, A through E, arrive at a computer centre at essentially the same time. Their estimated running times are as follows: A = 15min, B = 9min, C = 3 min, D = 6 min and E = 12 min. Their (externally defined) priorities are: A = 6, B = 3, C = 7, D = 9 and E = 4, with a *lower* value corresponding to a higher priority. For each of the following scheduling algorithms, determine the turnaround time for each job, and the average turnaround time for all jobs. Ignore process switching overhead and assume all jobs are completely CPU bound.

(a) Non-preemptive priority scheduling.

Jobs are run to completion in the order B, E, A, C, D. The turnaround time for each job is B = 9 min, E = 21 min, A = 36 min, C = 39 min, D = 45 min.
The average turnaround time is: (9+21+36+39+45) / 5 = 30 min.

(b) FCFS (run in order 15, 9, 3, 6, 12).

Jobs are run to completion in the order A, B, C, D, E. The turnaround time for each job is A = 15 min, B = 24 min, C = 27 min, D = 33 min, E = 45 min.
The average turnaround time is: (15+24+27+33+45) / 5 = 28.8 min.

(c) Shortest job first (SJF).

Jobs are run to completion in the order C, D, B, E, A. The turnaround time for each job is C = 3 min, D = 9 min, B = 18 min, E = 30 min, A = 45 min.
The average turnaround time is: (3+9+18+30+45) / 5 = 21 min.

(d) Round robin with a time quantum of 1 minute.

Jobs are run in round robin order:

Time	
0	*A,B,C,D,E.*
5	*A,B,C,D,E,*
10	*A,B,C,D,E,*
15	*A,B,D,E,*
19	*A,B,D,E,*
23	*A,B,D,E,*
27	*A,B,E,*
30	*A,B,E,*
33	*A,B,E,*
36	*A,E,*
38	*A,E,*
40	*A,E,*
42	*A,A,A*

The turnaround time for each job is A = 45 min, B = 35 min, C = 13 min, D = 26 min, E = 42 min. The average turnaround time is = (45+35+13+26+42) / 5 = 32.2 min.

9. An operating system requires 1 ms to handle interrupts, 2 ms to schedule the next process and 1 ms to dispatch the next process. Assume that the clock chip produces interrupts with a frequency of 50 Hz.

   (a) Calculate the percentage of overhead processing time within the operating system

   overhead = 50 x (1 + 2 +1) ms = 200 ms = 20%

   (b) Calculate the percentage of overhead processor time within the operating system if the time for interrupt handling, scheduling and dispatching doubles but the clock chip produces interrupts with a frequency of 5 Hz.

   overhead = 5 x (2 + 4 + 2) ms = 40 ms = 4%

Mentimeter Questions

Deadlock

1) Can deadlocked processes be outside allocation circular chain?

   Yes see 1.

2) Which of the following sequence is a safe sequence ?

   D: P1, P0, P2

3) What fraction of combinations is deadlock free

   1/3 See 3)

Scheduling

1) Which of the following state transitions is not possible ?

   Waiting to running. Note you can kill a blocked process.

2) FCFS Throughput & Turnaround time for 3600s, 1, 1, 1. Which is correct?

   Throughput 4 jobs / 3603 s = 0.0011 jobs/s
   Ave turnaround time (3600+3601+3602+3603)/4 = 3601.5s

3) FCFS Throughput & Turnaround time for 1, 1, 1 3600s, . Which is correct?

   Throughput 4 jobs / 3603 s = 0.0011 jobs/s
   Ave turnaround time (1 + 2 + 3 +3603)/4 = 902.25s

4) Which of the following are true

   weakness of priority scheduling is that priorities may not be meaningful.

   Pre-emptive scheduling suspends a running process before time slice expires

5) Which one of the following can not be scheduled by the kernel?

   User level thread