Homework 5

Erik Brakke

March 17, 2016

Answer 1

This is an M/G/1 system
$\lambda = 100$, $T_s = .0095$, $\rho = .95$, $A = 1/2 * [1 + (.0055/.0095)^2] = .668$, $q = (.95^2 * .668/.05) + .95 = 13$,
$T_q = 13/100 = .13$
This means that in the worst case, the number of requests serviced before R is $1/2 * 100 * .13 = 6.5$

Answer 2

See code in the 'src' folder

FCFS: The average slowdown for CPU bound jobs was 3.5 and the average slowdown for IO bound jobs was 83.79
From the perspective of IO bound jobs this is pretty bad because they only need the CPU for a few milliseconds on average, but because they have to wait for CPU bound jobs to finish, they experience a massive slowdown.

RR: The average slowdown for CPU bound jobs was 3.3 and the average slowdown for IO bound jobs was 17.02
RR with quantum 100ms does a much better job allowing IO bound jobs to finish. The slowdown they experienced compared to FCFS was much better. Though IO bound jobs would still prefer SRTN as this means they will almost always get serviced before a job that will take longer than theirs.

Answer 3

a. A request for f_n may get served, and then replace f_k in the cache, where $k < n$. Now if a request for f_k comes in, it will never be chosen off of Q2 because the max ID in the cache is n, so requests for f_k will sit in Q2 indefinitely

b. This is an N-batched SCAN scheduling algorithm

c. A larger N is better for increasing the effectiveness of the cache. A large N will allow more requests to enter Q1, and having more reqeusts in Q1 means that the cache is being utilized more. If N were small, then when a batch reaches its limit, a request r that may have been able to be served in $Q1$ will be pushed to another batch. By the time the latter batch is served, the file f_r may not be in the cache anymore

d. A smaller N will make the system more fair. When N is small, then only a small number of requests will be able to 'cut' in front of a bigger request

e. (a) N = 100. It is unlikely that a request will come in for a file that won't be in the cache, so it is better to serve as many files from the cache as possible per batch.

(b) N = 100. Again as above, it is unlikely that a request will come for a non-cached file, and because it is only a medium load, it is unlikely that many requests will 'cut' the request for the non-cached file

(c) N = 1. There is no use in using the cache as the files requested will most likely not be in the cache. This way we favor fairness as it doesn't make sense to serve a smaller file first (which would happen if N ¿ 1)

(d) N = 10. We want $N > 1$ because now the cache will be of use, however we do not want N = 100 because there is a heavy load on the server. It is likely that a request for a non-cached file will get served last as a lot of requests will end up in Q1. To strike a balance between speed and fairness, N = 10 will ensure that only 10 requests can be processed ahead of a request for a non-cached file.

Answer 4

a. Starvation is possible. A class C job is most susceptible to starvation as class B jobs could continue to arrive and interrupt the class C job. Becuase class B jobs are frequent, then it's possible that many class B jobs will 'cut' class C jobs and this could eventually lead to starvation

b. (a) Worst case is 5 seconds as class A jobs are of highest priority and are served as soon as they arrive

(b) Worst case is class A is being served, so it waits 5 + 0.5 = 5.5 seconds

(c) Worst case, A,B, and C jobs all arrive at the same time. It will take 10.5 seconds to clear our the A job and the queued B jobs. At time 10.5, job C can start working in half second increments (because B jobs arriving will interrupt it). It will take 40 seconds to process C with the half second increments, plus 10.5 seconds of 'startup', therefore it will take 50.5 seconds to serve CJK

c. If C took greater than 123.75 seconds of CPU time, then the C queue will stack indefinitely. This is because every minute, there is 10.5 seconds where C is idle. in 5 minutes, that 52.5 seconds of idle time. There is a 300 second window between C arrivals, which means that C can process for (300 - 52.5) = 247.5 seconds. Because C can only processes for half of that time, if C took 123.75 to process, then it would finish right as the next C arrived. Thus if it took longer, the C jobs would start to stack in the queue

d. If C arrived before A, then C will be using the resource R. Because A will not request CPU without having access to R, then it must wait for C to finish its entire process(using both the CPU and R) until it can make its first request to the CPU. Thus the high priority job will be waiting on a lower priority job.

d. If C arrives before A and B, then A will be blocked by C needing to use R, but C will also have to compete with B for resources. It will take 40 seconds for C to finish it's job, and then A will be able to go for it's 5 second time. So worst case it will take 45 seconds

e. Priority inversion is when the medium priority task can preempt the high priority task thus inverting the priorities of the two tasks. In this case the medium task was able to get the lock while the high priorty task was blocked, thus the medium task was able to run before the high priority task.

f. Priority inheritence means that if a lower priority task is blocking other tasks (medium and high), then the lower task will get 'promoted' to that of the highest priority being blocked. Thus, the medium priority task cannot preempt the high priority task

g. Worst case would now be 25 seconds. This is because as soon as A requests R, C is promoted higher than B and thus C will not have to wait for every B task to finish.