1. $\lambda = 100$ rps

$T_s = 0 - 19$ms uniformly distributed

bound on F-Scans unfairness

$$\frac{1}{2}\lambda T_q = \frac{1}{2} q \quad \text{by littles Law}$$

To solve for q we must use M/G/1 formula

$$q = \frac{\rho^2 A}{1-\rho} + \rho \qquad A = \frac{1}{2}\left[1 + \left(\frac{\sigma T_s}{T_s}\right)^2\right]$$

$$\rho = \frac{\lambda}{\mu} \qquad \mu = \frac{1}{T_s}$$

$$T_s = \frac{0+19}{2} = 9.5\text{ms} = .0095\text{s}$$

$$\sigma T_s = \sqrt{\frac{1}{12}(b-a)^2} = \sqrt{\frac{1}{12}(19)^2} = 5.5\text{ms} = .0055\text{s}$$

$$A = \frac{1}{2}\left[1 + \left(\frac{.0055}{.0095}\right)^2\right] = .67$$

$$\mu = \frac{1}{.0095} = 105$$

$$\rho = \frac{100}{105} = .95$$

$$q = \frac{.95^2(.67)}{1-.95} + .95 = 12.96$$

$$\boxed{\text{So the bound is } \left(\tfrac{1}{2}\right)12.96 \text{ or } 6.5}$$

3.

a. The most obvious example would be if Q1 always has requests in it then requests in Q2 will never be serviced.

b. n-batch scan

c. A larger N makes the cache more efficient

d. A larger N is less fair though.

e.

- ☐ N = 100 The load on the system is low so we don't care about fainess. Setting N to 100 will also increase our systems performance

- ☐ N=10 The load is moderate so we should be more concerned about fainess. N=10 provides a good balance on efficiency and fairness.

- ☐ N = 1 high load so fainess is a concern, caching provides little benefit here because there is little locality

- ☐ N = 10 because the load is extremely high we should be concerned with fainess, however extremelly high locality means caching will be effective, 10 provides a good balance between the two.

4.

a. yes starvation is possible. Class C jobs are the most susceptible because class b jobs are scheduled more frequently then class C jobs and have a higher priority.

b.

A event	1/minute	highest priority	5 seconds
B event	1/second	medium priority	.5 seconds
C event	1/5 minutes	lowest priority	20 seconds

A worst case 5 seconds. it has the highest priority

B worst case 5.5 seconds. an A event arrives during the B events execution.

C worst case 50 seconds. I used geometric series to think about this

event C takes 20 seconds to do work which will generate 20 B events, Those 20 B events take 10 s to service so they will generate an additional 10 B events those 10 B events take 5s to service so they will generate 5 more B events, This is a geometric series so the 20 seconds of work on A create 20 seconds of work on B events. If an A event arrives its 5 seconds of work will generate 5 seconds of work on B events. Total is 20+20+5+5 = 50s.

4

c. $2x + \left(\left\lceil \left(\frac{2x}{60}\right) \right\rceil 5\right) 2 = 300$

$x = 125$ so 125 is the maximum amount of cpu time for C.

d. if a Job C arrives just before a Job A the Job A will have to wait for Job C to finish using the resource R before it can begin to be serviced. Therefore the higher priority Job A must wait for the lower Priority Job C.

e. A Job C arrives before Job A. Then The C Job is interrupted by Job Bs causing Job C to take 40 seconds to complete Job A then Runs for a total of 45 seconds.

f. The above is an example of priority inversion even though Job A has higher priority then Job B Job B's are serviced before Job A. A higher priority Job is preempted by a lower priority one.

g. Priority inheritance attempts to solve the problem of priority inversion. The priority of a low priority task is increased to the highest priority task waiting for the shared resourse being consumed by the low priority task.

h. Job Bs can no longer interrupt JoB Cs using the resource so As worst case service time is 25 s