Rami Bitar Lab 4 Answers

3.1: From my results I conclude that when processes are created at the same time the results are fair, the cpu usage for each process are roughly equal.

3.2: In the case where processes are not all created at the same time the results show that cpu time is fair with later processes having higher usages but because they started with a higher initial usage than previous processes

3.3: After testing subtracting the initial cpu usage from its total run time I have found that they shared nearly identical run times about 10000 ms

10149 - 150
11348 - 1374
12185 - 2185
12859 - 2858

Conclusion: fair

3.4: From my analysis the total cpu usage of each of the iobnd processes are equal, and they are equal to their initial values so they ended up using 0 time but their initial values increased.

150 - 150
342 - 342
514 - 514
686 - 686

Conclusion: fair, but odd?

3.5 The way my code handles the null process is inside resched.c when I check the processes cpu time vs the top of readylist/minheap I would not add its new usage to the total whether or not it was more or less than the top of the list/heap. In case I do insert it I do not increase the total number of ready processes. If I pull the null entry from the list/heap I check if it is the null process if so I put it back into the list and the pull again if it is still null process I go with it but I do not subtract from the total number of ready processes.

4.1 Testing with 2 io processes and 2 cpu processes yields similar results to lab 3 both the I/O bound processes took up barely any cpu, the two cpu bound processes took significantly more cycles but again were even when compared to each other.

150 -150
342 - 342
11605 - 1601

10510 - 514

Conclusion: As expected the cpu bound values are smaller than before and still both roughly take 10000 ms to complete +- 4 ms

4.2 Testing it for the cpu bound processes as done in lab 3 the results are the practically the same but each total cpu usage and initial usage are smaller than before by about 1 second or 1000 ms.

10149 - 150 = 9999
10761 - 10761 = 10000
11301 - 1301 = 10000
11844 - 1844 = 10000

Conclusion: fair and slightly more efficient

BONUS: Some advantages red-black trees are better at finding arbitrary elements than min-heaps (log(n) vs (n)). They are are equal to min-heaps at deleting elements (log(n)) and both have equal space usage (n). Some disadvantages of red-black trees is that they have much greater overhead than min-heaps and inserting into the trees takes longer ((1) vs log(n)).