Assignment 2, Operating Systems

Jay R Bolton

October 13, 2011

Set 2: Exercise 2.1, 2.2, 2.6, 2.10, 2.14, 2.22

2.1 Let j be the number of jobs, t be total time. I'll try to do it for all j.

a. *Turnaround*
$t = ceil(\frac{j}{2})$
As soon as the first job hits IO, an interrupt is called and the actual process overlaps the IO, so that there are two processes per time cycle. With an odd number of jobs, we have to wait a whole time cycle on the last one, thus the "ceil" function.
Throughput
j/t
That's just the def of average.
Proc Utilization
$(\frac{j}{2*t}) * 100$ We need two processes per T to make 100%, but due to the odd numbered jobs it'll be a bit less.

b. I believe that it's going to be identical to a

2.2 Those jobs that have a history of using little processor time per total time are more likely to be IO bound. I assume I don't need to show this mathematically. Since the algorithm looks at history, it will also tend to favor neglected programs, so that if a process-bound job is pushed aside for too long, then it will inevitably be pushed to the top.

2.3 So for time sharing, response time is priority, while for multiprogramming, processor usage is priority. With multiprogramming, we'd want to always have access to as many possible jobs to fill up available resources, but then we wouldn't want to spend a lot of time switching jobs in memory or writing the state of a program out to a disk. I don't really have any specifics to offer on this question.

2.4 System calls allow communication between processes and kernel services and provides a limited bridge between user and system mode.

2.5 This could measure the load on the processor(s) and somehow the average could be set in a config so that the admin could have a really low level way of limiting processes. Or maybe it has to do with increasing performance. I guess if the average is high, then we have very little process and resource switching, so things will be faster.(?)