Name : Dhruv Subramanian

CS 354 Lab 3 – Time share scheduling in a modern OS

Changes I have made in my code :

main(): I just added code from my previous lab, have called create to create processes of iointensive and cpuintensive.

Resched.c : I declared the variables I needed use (qarray and tsdtab).

I set up how to handle the iointensive and cpuintensive processes. I saved the old process priority, then I got the new process priority and updated the old one. I made sure the highest priority process was running at any given time.

For iointensive I made sure that they gave up cpu to more cpu intensive processes until the latter finished.
Then in both cpu and intensive, I would call deque for the multilevelfeedback queue and would update the process being executed to the current process. From then, I would context switch after updating the value of preempt based on the quantum time parameter of the dispatch table.

Initialize.c : I set up and printed out the dispatch table

ready.c: I made a call to enqueue after updating the state of the process to ready based on the process ID of the process. Once enqueue is called, reschedule is called.

multilevelfbq.c :made a multilevelfeedback dequeue function which takes a qid16 array of queues and calls the dequeue function on the ith index that isn't empty. As long as isn't empty, the dequeue function removes and returns the ith element of the queue array on the feedback queue.

multilevelfbq.h :
declared the qid16 array of queues.

create.c: handled the null process and set the default process priority.

ts_disptb.h : defined the ts_disptb structure.

xinu.h: added the .h files to this main file.

Problem 3.2:

I implemented the Solaris TS dispatch table in initialize.c and updated the value of each element of the structure based on which ever level it was on. The values updated were the new priority of the CPU bound process, the new priority of the I/o bound process and the value of the new time-slice.

I have taken care of the null process in the code in create. We need to take of the null process because when resched is called to to execute a new process, it assumes that the readylist isn't empty. It assumes that there is atleast one process in it at that time. This is the null process. I've checked to see if the process is null, and if it is I've set that priority to 0. If there is no other process other than the null process, then it is the one that executes. Otherwise I've set a default priority of 37.

Problem 3.3:

My decision on the initial priority was 37. this was because by choosing this value it would neither have too high a quantum time value nor would it be too low. This average value would create little bias either towards cpu intensive or io intensive.
The multilevel feedback queue that I have implemented is acceptable because we have a constant time insertion into the queue and a constant time dequeue operation. This is because iterationg through a known number of priority levels is constant. Once we know the level, in order to insert into the front or back of the queue we have a pointer to the tail and head of the queue at the level. This will allow us a constant time operation in the implementation of the feedback queue.

So as to implement the multilevel feedback queue to effect constant time complexity I used an array of queues of type qid16 defined in multilevelfbq.h . In the multilevelfbq.c, I implemented the dequeue function for the multilevel feedback queue. I looped backward through the priority levels and then I called on dequeue function, which removes and returns the first process on the multilevel feedback queue.

Problem 4:

Analysis of my findings:

a) CPU intensive processes:

For purely CPU intensive processes I found that they hog the CPU. As they use the CPU, the priorities of those processes will decrease as the time slice decreases. And the CPU processes will sort of execute in a round robin fashion. In this way the Solaris scheduler will perform based on different workloads.

b) IO intensive processes: For the IO intensive processes, each process tries to give up CPU to the other processes thereby decreasing the time slice and increasing priority. This will proceed until the all processes have been executed and finish and will continue to proceed in a round robin manner.

c)Both CPU and IO intensive processes: For both together, the IO intensive processes will give up CPU to cpu intensive and will start getting blocked so initially while executing we can't see it. Once the the CPU processes keep executing, the priority will decrease, and cause the
IO intensive blocked process to start executing and then, we can see them execute.

Bonus problem:

On carrying out a test scenarion to implement the constant time overhead gauge, I checked to see if the clktimewas the same before and after resched, which it was. I used only cpu bound processes and varied them to a point that measured the time difference and overhead. This waqs one way to estimate overhead .