Bryan Worrell

CS 446/646

Dispatcher Simulator Design

04/18/2011

Introduction

The design of this dispatcher uses several different operating system concepts in regards to CPU scheduling. One real time and three user priority queues to store and retrieve processes based on what their priority is. The real time queue acts in a round robin fashion as does the lowest level user priority queue when all of the other queues are empty. Using the concept of aging, jobs with a priority initially at one or two will eventually be lowered to three if they do not finish in the time quantum allowed. Finally, average turnaround, normalized turnaround and waiting times will be calculated to test program efficiency.

The Process Structure

The object being queued will be a process struct with several attributes that will be described in this section. The first attribute is the time the job arrives. This is needed because for each time quantum tick, the file containing a list of jobs will be searched and if the arrival time of a job is equal to the current run time, it will be enqueued. The priority aspect will then help determine whether it needs to be inserted into the real time queue or one of the three user queues.

The amount of CPU time it takes for the job to finish will also be stored as part of the struct. A time remaining variable will be initially set to this value since we do not want to change it as it is needed later on for calculations of turnaround time and because we also need a way to determine when the job has finally finished. This way the time remaining can be ticked during each time quantum it runs so that once it is zero, we know it is done and the known total CPU execution time remains unchanged.

Lastly, two integers and one bool variable will be used for time stamping purposes. These are important because the exact time a process starts executing and the time that it actually ends are also required for the data calculations later on. The bool variable helps with time stamping the correct start time because it lets us know if it has yet to be run once and if it has not, we can simply set the bool to true and time stamp it.

Program Implementation

The first step will be to determine how many processes there are in the dispatch list input file. This allows the setup of a loop that will not end until all jobs have finished. As soon as this loop is entered, the algorithm which has two major parts to it begins. The first part entails searching the entire dispatch list file for any jobs that have arrived at the current run time. If any jobs have arrived, their priority is determined, which is then used to place them in the correct queue. Once all jobs are placed appropriately, a bool is checked. The purpose of this bool is to ensure that any two jobs with the same priority are enqueued in the correct order; in operating system terms this is preempting. Any new job takes precedence over one that has been executed once, but is still not finished.

The next part covers executing whichever job is next for one time quantum. First, the real time queue is checked and until this queue is empty, the jobs in here will be cycled through. If the real time queue is empty, the user priority one is examined. Any jobs in this queue are retrieved, ticked and then enqueued to priority two if they are not done. The same thing happens to any jobs retrieved from priority two, except they are enqueued to priority three if they have time remaining. Finally, priority three functions the same as the real time queue such that all jobs are cycled through until it is empty, unless a job of higher priority suddenly appears in the upper level ones. As soon as any job finishes, it is enqueued to a list of finished processes to be used in the final calculations.

Once all the jobs have been successfully executed, the average of the overall turnaround time, normalized turnaround time and wait time for all jobs are calculated. This is the part where the time stamped characteristics for each process struct stored in the finished list are used. The data are then displayed with the precision set to two decimal places.

Comparison to Real Dispatcher Functionality

There are many similarities this project has to actual operating system dispatchers. Most of the concepts used are implemented in interactive systems, such as the round robin algorithm. When it comes to CPU scheduling, one must carefully decide on what type of algorithms to use. Whether the system is a batch or interactive one is a major factor for deciding the overall structure of the dispatcher. The real time queue in this design corresponds to “response time” and the processes that a user expects to complete immediately. The three level user priority queue structure is a basic design that outputs fairly good turnaround times. Several of the problems in the field of CPU scheduling are clearly addressed in the design. Overall, the structure is a very fair simulation of how an interactive operating system can work.