10/25/2018

CptS 223

**PA5 – Report**

Freya Varez, James Ponce

The purpose of this assignment is to design and implement a ‘Job-first’ scheduler, the specifications of this scheduler allow multiple jobs to be completed at once, dependent of the number of free processors, all of which are operating in parallel. The scheduler must complete a list of jobs in an optimal amount of time. Each job is introduced to the scheduler in the form where is the number of parallel processors necessary to complete the task in amount of time. In this implementation, the jobs are read from a utility file and assigned a respective unique ­.

For each job accepted, is called; the details and design of this function are mentioned below and in the attached *Design Document.* However, in simple terms, this function takes the job from the utility file, validates the job and adds it to a *Wait Queue* (implemented with a priority queue, see *Design Document*). The mentioned validation checks that and , displaying an error message to the console and ‘deleting’ the job should these conditions not be met.

Additionally, the following functions are implemented:

* : Finds the job that takes the shortest amount of time, or . This job is denoted and respective number of processors denoted . This function is implemented indirectly through the function, which gets the shortest job from the heap in constant time with a basic function.
* : Checks if there are a sufficient number of free processors to complete the given job. Implemented indirectly through the function, at the time of each insert, (respective processors required for the job) is evaluated. At evaluation, the job is either moved into the priority queue, or deleted (where ) with a respective error printed to the console.

Else, if another job is running, the function evaluates if () is true. In both cases, runs in constant time.

* : Deletes from priority queue, and temporarily delegates to the given job. This function is implemented such that, if enough processors are available through , another job is moved. Thus, runs in time, where is the number of available jobs.
* : Indirectly implemented through the and function(s). Dependent on the number of available processors, each job is moved into the simulation. Runs in time, where is the number of available jobs, as theoretically, all jobs are able to run in parallel.
* : Implemented via function. Simulates running the job, initiating a countdown timer for respective while the job is ‘running’. Time complexity is .
* : Utility function simulating a timer for the above function. Initiates the function below after timer decrements to zero. Constant time complexity.
* : Implemented indirectly through the function. Releases the number of processors associated with the job back into the free processor pool. Constant time complexity.
* : A member function of the scheduler class, records the number of iterations necessary to complete all jobs within the priority queue. Implemented indirectly through .