10/25/2018

CptS 223

**PA5 – Design Documentation**

Freya Varez, James Ponce

Note: The below summery of functions was taken from the attached PA5-report for simplicity, as many of the functions used are under different name, or were implemented indirectly with other functions:

* : 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 .

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Free Processors**

**Simulator**

(Job destructor call)

**Wait-Queue**

**Trash**

**Scheduler**

**File**

|  |  |
| --- | --- |
|  | constant |
|  | Constant time |
|  |  |
|  |  |
|  | Constant time |
|  | Constant time |

The main shortcoming of using the shortest-job first strategy, is that, assuming the rest of the implementation is the same, each deletion is at the front of the struct, thus it’s required that a linked-list type structure (list, queue, stack etc.) is used to increase the time efficiency. This means that at any point, if an object must be searched, there is an runtime. If deletions occurred at the end of the list, a different implementation could be used, such as an array, making search time (with other function implementations remaining the same) with a simple binary search.