Chris Banci

CS433: Operating Systems

Prof. Zhang

February 9, 2017

Assignment 1 - Report

**Choice of Data Structure:**

The data structure that I chose to implement the priority queue is a *Min-Heap*. A *Min-Heap* is a complete binary tree, such that, each node contains the heap property. The heap property is where each node in the tree is less than or equal to the data in its childs node. This allows for the minimum value to be at the root of the tree. Since the root value of a min-heap is always the smallest value, it makes sense to use this data structure, because in a priority queue, the front element that is to be dequeued, is the element with the highest priority.

Initially, I started off the implementation of the priority queue using a sorted linked-list as it was easier to implement without needing to look at resources. However, after deciding to compete for the fastest run time, a sorted-link list wasn't the most efficient approach in terms of time complexity, so I switched to using a Min-Heap which is the ideal data structure to implement a priority queue.

**Implementation:**

To enqueue/insert a process into the ready queue…

1) The function *getProcess()* is called which returns the address of a PCB object from the vector called process\_list. This vector holds all processes in the system.

2) The reference returned from *getProcess()* is passed into *insertProc()* which is then passed into *insert()*. The function *insert()* puts the PCB into its correct position within the heap according to its priority. The process's state is then set to READY.

To dequeue/remove a process from the ready queue…

1) The function *removeHighestProc()* is called which calls the *deleteMin()* function. The *deleteMin()* function removes the process with the highest priority, which is located at the root of the heap. The process's state is then set to RUNNING.

**Time Complexity of Implementation:**

When inserting an element into a min-heap, it is placed at the end of the tree and shifted up to its correct position. Inserting an element takes an average case of *O(log n)* operations due to the shift up procedure and the height of the complete binary tree.

When deleting the minimum element of a min-heap, the root element is swapped with the last element and shifted down to its correct position. Like insertion, removing the front node takes an average case of *O(log n)* operations due to the sift down procedure and the height of the complete binary tree.

Retrieving the min element of a min-heap, takesan average case of *O(1)* operations since the root of a min-heap is always the smallest value and no operations are needed. Simply accessing the root of the complete binary tree returns the minimum value.

**Timing Results:**

Here are my top 3 timing results for test 2:

* Run 1 = 35.843 ms
* Run 2 = 35.891 ms
* Run 3 = 35.875 ms

Average = 35.870

The timing results exceeded my expectations. I knew my implementation using min-heap would be fast, but I didn't expect it to be this fast. What I also didn’t expect was the vast difference in performance from the optimizations I made to get a faster runtime.

**Additional:**

To optimize performance, I have done the following:

* *-O2* over -O code optimization flag.
* *Mt19937* over *Rand* when generating random numbers.
* random\_device over time(nullptr) when seeding mt19937.
* reserving memory for vectors to prevent dynamic memory reallocation.
* *#pragma once* in headers files.
* *“\n”* over “*std::endl*” when creating a new line.
* “*emplace\_back*” over “*push\_back*” when using vectors.
* *prefix* over *postfix* notation.
* *auto* over explicit types.

The biggest bottleneck for test 2 was generating random priorities and setting it to a process every time a process was to be added to the ready queue. To reduce this runtime overhead, I made it so that numbers were randomly generated beforehand and placed them into a vector where it was iterated to get a random priority for a process. Along with using the random generator mt19937, which is more faster than rand(), it had brought down the runtime down from around 55ms to around 35ms.

**Resources:**

* Time complexity from <http://bigocheatsheet.com/>
* MinHeap data structure from CS311 homework with modifications.