COSC 1285 Algorithms and Analysis

RMIT University | Semester 1, 2020

Assignment 1: Process Scheduler | Task B: Evaluate your Data Structures

Kamlesh Chag s3764712

Henry Chin s3774122

Introduction

Scheduling, also known as process scheduling, is a method that is used in computing to assign and distribute computing resources. An operating system uses scheduling to ensure processes are executed efficiently but equally and potentially reduce wait times. Computational elements such as processing time and memory are allocated to the various processes, threads, data flows or other applications that need them. Scheduling concerns itself mainly with throughput, latency and response time. Modern computers with large processing power no longer rely heavily on scheduling but still prioritizes managing requests.

Active processes, traditionally placed in an array, are called runqueue and contain priority values that determine which process to run next. Processes that have lowest vruntime are brought forward and once executed, are removed from the runqueue. Otherwise, if the process is interrupted, it is reinserted based on its new vruntime while processes that have similar vruntime follow a First-In-First-Out (FIFO) order. In this assignment, the runqueue is defined as an abstract data and is represented by both sequential and tree representation and later used to compute average speeds of various operations.

Theoretical Analysis

Here, a theoretical analysis of the performance is conducted based on our implementations. Best case, worse case running time estimate and the asymptotic complexity (Big O) is reported based on a given *runqueue* of size *n* on three different scenarios. For Scenario 1, the *runqueue* is growing as increasing processes are added and performance is evaluated in terms of the enqueue operation (EN). In Scenario 2, the *runqueue* is shrinking as increasing processes are dequeueing and performance is evaluated in terms of the dequeue operation (DE). Lastly, Scenario 3 assumes a non-changing *runqueue* and performance is evaluated in terms of the PT operation by calculating total vruntime. The results are shown in the tables below:

Theoretical Analysis			
Scenario 1 **(Enqueue)**	Best Case	Worse Case	Big O
Ordered Array	If the new element is the largest then the loop will break on the first comparison to sort it thus creating the best case for growing the runqueue.	If the new element has the smallest vruntime then we have to go through the whole array to place it on the first position thus creating a worst case.	O(n)
Ordered Linked-List	If the new element has the smallest vruntime then it directly gets added in front of head making it the new head thus making it the best case.	If the element is with the largest vruntime then we need to traverse the whole list and add it to the end thus making it the worst case..	O(n)
Binary Search Tree	When inserting a mid priority element, it will always be placed in the main root.	For inserting element 0, it must be inserted as left child of 1. Therefore, we need to traverse all elements (in order 3,2,1) to insert 0.	O(n)

Scenario 2 **(Dequeue)**	Best Case	Worse Case	Big O
Ordered Array	The best case for dequeue is when there's only 1process in the runqueue as it won't require any traversal or shrinking of the array.	The worst case would be if there are a large number of processes in the array because then we have to dequeue the process and then change the whole array which can take longer time making it the worst case.	O(n)
Ordered Linked-List	The best case for dequeue for the ordered linked list is every case because the head is always going to be same thus having O(1)	There is no valid worst case available for linked lists as dequeue removes the node from the head.	O(1)
Binary Search Tree	The best case is when removing an element	The worst case would be when removing an	O(n)

	would not throw off the balance of the tree.	element, it would throw off the balance of the tree and it would need rebalancing.	

Scenario 3 **(Proceeding Process)**	Best Case	Worse Case	Big O
Ordered Array	Taking Processing time for first element as we don't have to traverse anything	Proceeding time of last element as we need to traverse the whole runqueue	O(log n)
Ordered Linked-List	Taking Processing time for first element as we don't have to traverse anything	Proceeding time of last element as we need to traverse the whole runqueue	O(n)
Binary Search Tree		When searching element 1, we have to traverse all elements to find 1 (in order 3,2,1).	O(n)

On an overall basis, LinkedList was found to be the quickest to perform all tasks on an average making it our prefered choice for process scheduler. Linkedlist has the best possible way to perform dequeue. For enqueue, all the 3 data structures had a similar complexity.

Data Generation & Experimental Setup

A code was written which generated an input file. This code was run on cmd with the below syntax :
Java DataGenerator <number of elements> <filename(without extension)> <EN | PT | DE>.
This creates a file names <filename> which has <number of elements> of the process<EN | PT | DE>

THe experimental setup had a program which gave us the runtime of the process when we had provided the RunqueueTester with the files created above. This test was done on a MacBook Air 2018, with no programs running in background.

Empirical Analysis

500 Processes (Averaged)			
	Dequeue	Enqueue	Proceeding Time
Ordered Array	0.0071	0.0212	0.0141
Ordered Linked-List	0.0045	0.0075	0.0184
Binary Search Tree	0.0068	0.0109	0.0204

1250 Processes (Averaged)			
	Dequeue	Enqueue	Proceeding Process
Ordered Array	0.0168	0.0509	0.0307
Ordered Linked-List	0.0074	0.0169	0.0404
Binary Search Tree	0.0148	0.0243	0.0520

2500 Processes (Averaged)			
	Dequeue	Enqueue	Proceeding Process
Ordered Array	0.0310	0.0860	0.0419
Ordered Linked-List	0.0146	0.0352	0.0895
Binary Search Tree	0.0137	0.0727	0.1330

From the above analysis we conclude that linked list is the best way to implement a process scheduler as it is fastest in most of the scenarios and test cases. This also supports our theoretical analysis thus supporting our claim on the linked list to be used for process scheduling.

Recommendations

Dynamic Array can be used for a better handling of the work to be done on it making deletion easier and maintaining the array length without keeping a nore of the length varying throughout the process.

Linked List can use quicksort to sort every time a dequeue happens as it is a faster option to sort out of quick sort bubble sort and insertion sort.

Binary search tree (BST) is a non-cyclic graph with sorting mechanism. Most operations on a BST takes time proportional to the height of the tree, hence the height plays an important role for searching, insertion and deletion. Similarly to a dictionary, BST is maintained in a way such that words are sorted alphabetically with key values compared with the root node. When it is greater, the search moves to the right, else it goes to the left. For each level, the key is compared with only one node hence the maximum number of comparisons is equal to the tree's height (Rehman, Mehta & Elahi 2012). A self-balancing binary search tree is used to solve this problem by performing transformations on the tree to reduce the height. The minimum height of a binary tree with n nodes can be achieved using $log_2 n$.

An AVL tree (named after inventors Adelson-Velsky and Landis) is a type of self-balancing binary search tree with additional property that the difference between height of left subtree and right subtree of any node cannot be more than 1. If at any time they differ by more than one, rebalancing or rotations are done to restore its property. AVL trees are often compared to hash tables, with advantages such as persistency and fast enumeration of items in key order, which hash tables do not provide.

Summary

Out of the available options of data structures given to us for the process scheduler it Linked List was best suited to carry out the purpose of process scheduling. Linked Lists can handle the large number of data easily. Dequing the processes from the run queue is easiest that way and even rest all methods or functions of a process scheduler. The data structure Proc enables it to be used in different ways. Having a constant head pointer helps the linked list to perform the task. Thus it makes our best choice for this task.

References

Rehman, I, Mehta, Z & Elahi, M 2012, ‘An Improved Algorithm to Maintain the Binary Search Tree Dynamically’, *Multitopic Conference (INMIC), 2012 15th International*, pp. 253 - 257.