A Project Report

On

EDF scheduling algorithm based on heap sort with information theory concepts

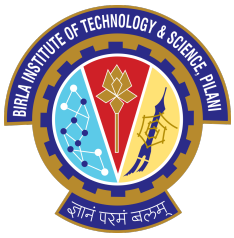
By

Hariharasudhan K
2018H1240083H

Under the supervision of

Dr. Soumya J

Submitted in Partial Fullfillment of the Requirements of
BITS G553 : Real Time Systems



BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI
HYDERABAD CAMPUS
November 2019

Acknowledgement

I would like to express my special thanks of gratitude to my teacher(Dr.Soumya J) as well as our Head Of Department (Sanket Goel) who gave me this opportunity to do a project on the topic (EDF scheduling algorithm based on heap sort with information theory concepts) which helped me in doing a lot of Research and I came to know many new things.I am really thankful to them.

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI
Hyderabad Campus

Certificate

This is to certify that the project report entitled "EDF scheduling algorithm based on heap sort with information theory concepts" submitted by Mr. Hariharasudhan K (ID No. 2018H1240083H) in partial fulfillment of the requirements of the course BITS G553, Real time systems Course, embodies the workdone by him under my supervision and guidance.

Date:

Dr. Soumya J

BITS- Pilani, Hyderabad Campus

Abstract

Development of a modified Heap Sort algorithm based Earliest Deadline First Scheduling. First, the Earliest Deadline first scheduling is done with the conventional heap sort and it is made as compact as possible. The new term called switch over is introduced in the context and to remove it, a modification has been done for the existing heap sort. The new Earliest Deadline first Scheduling with the modified heap sort is checked for switch over and compared with the conventional algorithm. A new library called libscheduler is created with the algorithms and yet to be extended with RMA and other different scheduling algorithms.

Contents

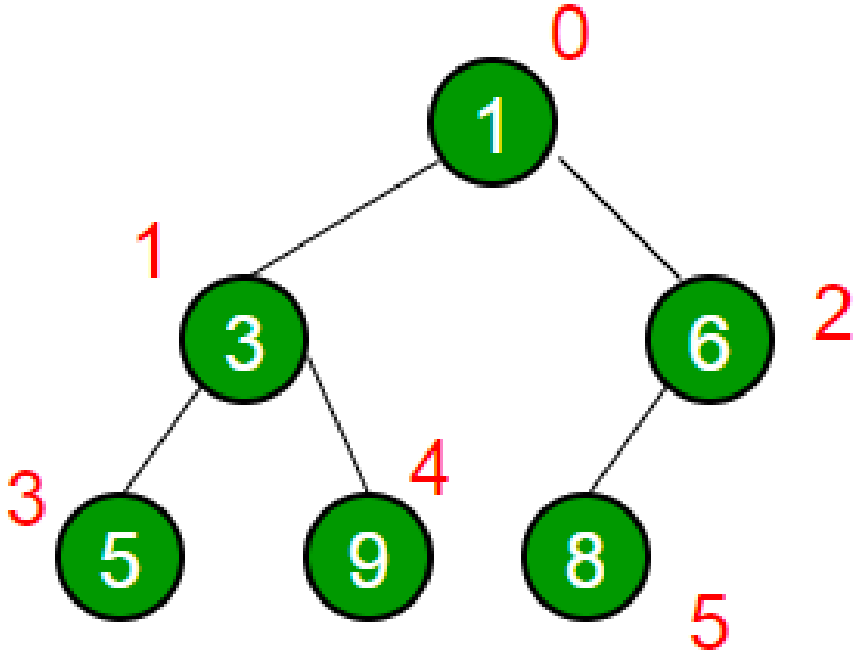
Chapter 1

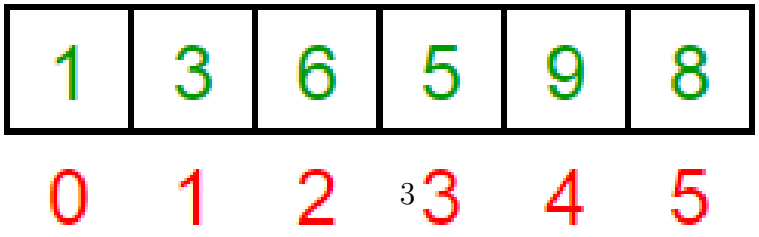
Introduction

Earliest Deadline first(EDF) scheduling is the best scheduling algorithm for the real time scheduling provided that the system is not overloaded. Overloaded in the sense the CPU utilisation should be lesser the one for the single processor system. For the non overloaded system, EDF scheduling will be surely done and there is no issues with that. But the issue comes from the fact that the time required to schedule the processes are more. The processes should be sorted in terms of their relative deadline and based on that it should be scheduled. The process with minimum releative deadline will be scheduled first. Here, on analysing we dont need sorted array all the time. It is required to find the process with minimum relative deadline. The structure that will always return the minimum value in the ready queue is the min Heap structure. Based on the min Heap structure, we can push elements into the array queue in run time. So this algorithm is dynamic in nature. Dynamic in the sense the heap should be made in such a way that it should be accessed in the unit time scale. The situtuation of switching over problem will come in the proposed algorithm and it is rectified by modified heap sort algorithm which will be discussed later.
zxfchzcvhs

Chapter 2

Min Heap Structure





Min Heap is a structure that will give the minimum value in the array queue. The minimum element in the array will be taken out and the structure will alter itself in such a way that the next smallest element will be pointed out at the top. The main task here is we should make count that which deadline belong to which process. So in our algorithm two heap structures are made and both structure will work simultaneously. One change in the deadline heap will affect the elements in the index heap in such a way that both are related to each other always.

Chapter 3

Algorithm

3.1 Algorithm for heapify down

```
Algorithm reheapDown (heap, root, last)
Reestablishes heap by moving data in root down to its
correct location in the heap.
Pre
        heap is an array of data
        root is root of heap or subheap
        last is an index to the last element in heap
Post
        heap has been restored
Determine which child has larger key

1 if (there is a left subtree)
          1 set leftKey to left subtree key
        2 if (there is a right subtree)
          1 set rightKey to right subtree key
        3 else
          1 set rightKey to null key
        4 end if
        5 if (leftKey > rightKey)
          1 set largeSubtree to left subtree
        6 else
          1 set largeSubtree to right subtree
        7 end if
        Test if root > larger subtree
        8 if (root key < largeSubtree key)
          1 exchange root and largeSubtree
          2 reheapDown (heap, largeSubtree, last)
        9 end if
2 end if
```

3.2 Algorithm for heapify up

```
Algorithm reheapUp (heap, newNode)
Reestablishes heap by moving data in child up to its
correct location in the heap array.
Pre
        heap is array containing an invalid heap
        newNode is index location to new data in heap
Post
        heap has been reordered

1 if (newNode not the root)
1 set parent to parent of newNode
2 if (newNode key > parent key)
1 exchange newNode and parent)
2 reheapUp (heap, parent)
3 end if
2 end if
end reheapUp
```

3.3 Algorithm for pushing heap

```
Algorithm minHeapPush (heap, last, data)
*       Inserts data into heap.
        Pre
                heap is a valid heap structure
                last is reference parameter to last node in heap
                data contains data to be inserted
        Post
                data have been inserted into heap
        Return true if successful; false if array full
/
        1 if (heap full)
        1 return false
        2 end if
        3 increment last
        4 move data to last node
        5 reheapUp (heap, last)
        6 return true
end minHeapPush
```

3.4 Algorithm for poping heap

```
Algorithm minHeapPop (heap, last, dataOut)
Deletes root of heap and passes data back to caller.
Pre
        heap is a valid heap structure
        last is reference parameter to last node in heap
        dataOut is reference parameter for output area
Post
        root deleted and heap rebuilt
        root data placed in dataOut
Return true if successful; false if array empty

1 if (heap empty)
1 return false
2 end if
3 set dataOut to root data
4 move last data to root
5 decrement last
6 reheapDown (heap, 0, last)
7 return true
end minHeapPop
```

3.5 Octave code for moving plot

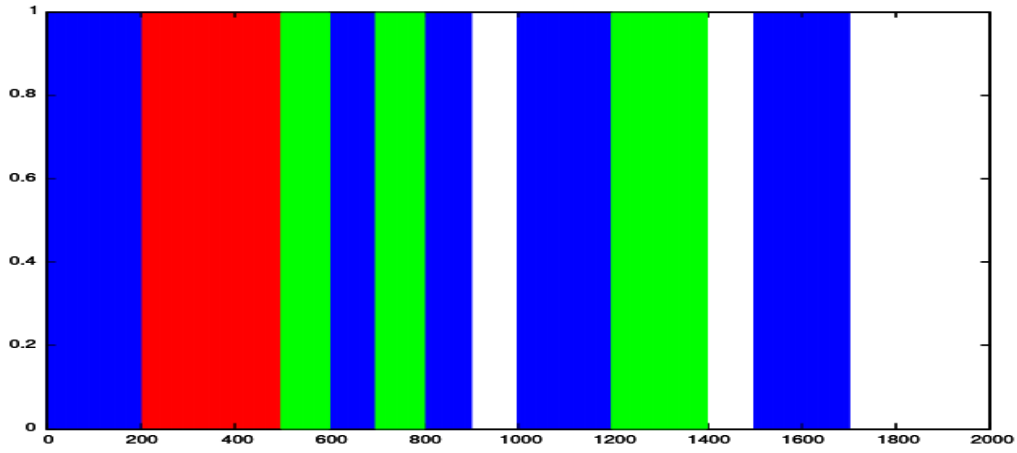
```
a =dlmread('polar.dat')
s = a(:,1) + 1;
b = a(:,2);
c = a(:,3);
d=a(:,4)


for  i=1:20
  stem(s(i*100-99:i*100),b(i*100-99:i*100),'r');
  axis([1 2000 -1 2])
  stem(s(i*100-99:i*100),c(i*100-99:i*100),'b');
    stem(s(i*100-99:i*100),d(i*100-99:i*100),'g');
  hold on;
  pause(1);

end
```

Chapter 4

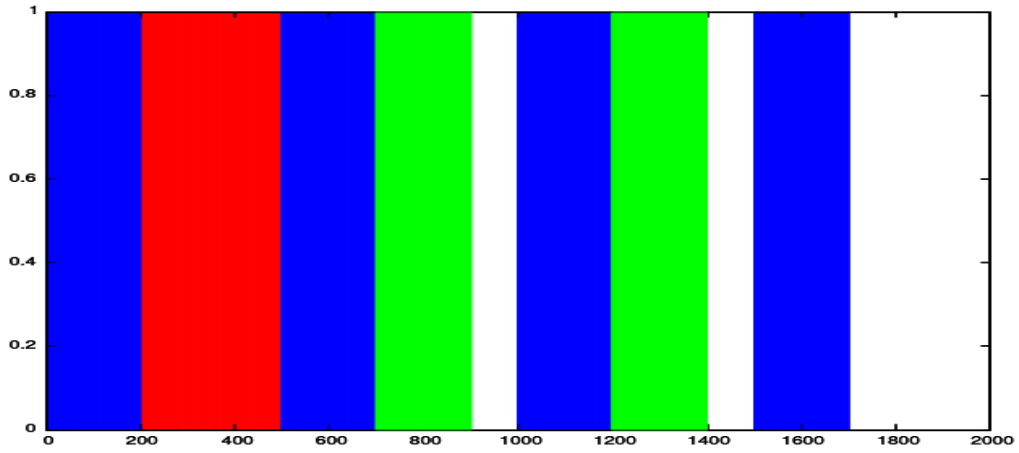
Novelty of the algorithm

Task sets: T1 = (3,20,7), red
T2 = (2,5,4), blue
T3 = (2,10,9), green

4.1 Conventional Heap Sort



4.2 Modified Heap Sort



4.3 Explanation

The conventional heap sort is the best ever algorithm found till now to find the minimum number in an array multiple times which can work on changing array size. The complexity of finding the minimum number is O(logn) provided the heap program is inserted properly. But, the thing is, the algorithm is executed for each timescale and we need to insert the deadline of the tasks inserted from the back.
This algorithm will work fine provided that the relative deadline of each of the task are different. But when the relative deadline of any two tasks are similar at that time we will face a problem of switching over.

4.4 Switch over

First of all, the term Switching over is used first here only and there is no literature on that term. I am defining switching over as the term which is used to denote a situation of two or more processes having the same relative deadline and since heap sort is used it will switch among themselves as all them after completing its unit execution time, it is fed at the bottom of the heap. By the time the other process which have the same deadline will be on top and it will get executed.

4.5 Modification to avoid switching over

There is no other way but we need to break the rules of the conventional heap sort. It will make the sort less efficient ofcourse but it will avoid switching over and proves to be more efficient than the efficient heap sort with switching over. The algorithm is simply changed that swap will occur in heap sort for less than or equal to instead of less than alone. Obviously the last process that is entered will go to top provided there is no job that is coming in the next instance with comparatively less deadline than the task that is executed just before.

Chapter 5

Conclusion

The implementation of earliest Deadline First algorithm using heap sort is written in c language. The modification algorithm to EDF is designed based on the information theory concepts that is studied. The GUI for the scheduler is created by means of GNUplot library. A library for c language libscheduler have been created.

Chapter 6

Future Work

Until now, the library is made with EDF algorithm and its modification algorithm. Our next mission is to make RMA and round robin schedulers for libscheduler and publish this work to the open source community for further development.

References

[1] arlos A. Rincón C. , Student Member, IEEE, Xingliang Zou, Student Member, IEEE, and Albert M. K. Cheng , Senior Member, IEEE Real-Time Multiprocessor Scheduling Algorithm Based on Information Theory Principles IEEE EMBEDDED SYSTEMS LETTERS, VOL. 9, NO. 4, DECEMBER 2017

[2] HANSHAN MENG, QIANG ZHU , AND FEI XIA Improvement of the Dynamic Priority Scheduling Algorithm Based on a Heapsort IEEE Access, VOLUME 7, 2019

[3] . Lin, Y. Wang, N. Chang, and M. Pedram Concurrent task scheduling and dynamic voltage and frequency scaling in a real-time embedded system with energy harvesting, IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 35, no. 11, pp. 18901902, Nov. 2016.

[4] . E. Ghor and M. Chetto, Overhead considerations in real-time energy harvesting systems Proc. Int. Conf. Pervasive Embedded Comput.Commun. Syst. (PECCS), Feb. 2015, pp. 358362.

[5] . L. Liu and J. W. Layland, Scheduling algorithms for multiprogramming in a hard-real-time environment, J. ACM, vol. 20, no. 1, pp. 4661,1973.

[6] . K. Baruah, M. Bertogna, and G. C. Buttazzo, Multiprocessor Scheduling for Real-Time Systems (Embedded Systems) Cham, Switzerland: Springer, 2015.

[7] . E. Shannon, A mathematical theory of communication, Bell Syst. Tech. J., vol. 27, no. 3, pp. 379423, 1948.