A Knapsack Based CPU Process Scheduling Using Neelsack Algorithm

Neelakantagouda Patil[1],
[1] Torry Harris Business Solutions, Bangalore, India

Abstract

Scheduling concept has gained much popularity since the advent of the operating systems scheduling policies, networking packets scheduling and also because of many research and analysis in data mining and many simulations based systems. Specific to the operating systems, it becomes very important to reduce the load on processor, which is critical resource. Thus utilizing the CPU to maximum possible extent and ensuring that all the processes get served by the CPU. There are many process scheduling algorithms like FIFO, priority, round robin which individually address on different performance measures like waiting time, turnaround time, throughput etc., with the fact that no one are proved to be the best with all the performance measures. This new proposed technique is designed on the base of the optimization technique and proved to show improvements in the different performance measures with appropriate results and experimental facts with different cases. The main agenda here is to show case that optimization technique can be used to schedule processes.

Keywords: *CPU Scheduling, Round Robin, Dynamic programming, knapsack, turnaround time, waiting time, starvation.*

1. Introduction

The necessity of scheduling has seen drastic change in the last few decades and the reason behind is now we are in the information age, where the data feeding, storing, processing, retrieving has become a vital concept. This process of doing things needs lot of processing and computational capabilities. Again thinking on the edge of technological advancement we can see laptops, tablets and smart phones are getting more compact to capture market and attract customers from their micro designs, using advanced technologies and hardware designs, thus making handy and lesser weighed devices. But looking at the other face of this, challenge is not to compromise on the performance which is also equally critical to retain customers. So this is where the advanced science needs the touch of basic science and its methodologies. In this context the scheduling process for operating system is helpful in getting advanced and updated techniques to adjust and serve the current technological expectations in the fields like data mining technology, networking, operating system and many more.

Scheduling generally in any field means making a set of tasks of same kind work together sharing same resource at same time, fixing or allocating the appropriate time or resource such that each task can complete like others. In the design of computers each and every single parts are scheduled, and CPU must be scheduled, as it is the core part for operations. Scheduling is implemented at operating system level for serving the set of processes that come into it with the request. There exist many techniques for scheduling[1] the processes in operating systems, it may be like FIFO(First Come First Serve) or others like SJF(Shortest Job First),Priority based, Round robin[2] and many more. These are designed to schedule processes in CPU and in particular case one will be better applicable than others in few performance measures[3]. The most advanced scheduling can be less efficient than other in some cases.

This new approach can be proposed as more efficient scheduling algorithm than existing ones(at least in few cases, it's based on the operating system to select for the situation given). The approach followed is applying optimization technique, which has its own vast application. The aim of optimization technique is to get a maximum optimized solution for any given problem. To be more specific in this new approach of scheduling, knapsack algorithm[4], has been applied which is a dynamic programming technique[5].

2. Overview

Any operating system works on the basic principle of serving one or more process at the same time, but it is mere illusion that the operating system does that. Instead it will serve the process in a given short time slice and since it is very small time that we can`t even notice. In the same instance OS also needs to handle multiple process that arrive with the request. As the processes have different burst time, different arrival time, so such all complexities make the design of scheduling really challenging.

Since the need for scheduling process was very basic, there were many researches took place. As a result of which new algorithms were invented. Then regarding quality of the scheduling algorithms, it is measured with different approaches and also based on how it behaves with the increase in number of processes that arrive to the CPU. The scheduling algorithms quality or efficiency can be measure by many parameters like CPU utilization, through put, waiting time, response time, turnaround time, context switching, which considers different aspects to measure the CPU scheduling, and also it is known that, no one scheduling algorithm alone is proved best in all the mentioned parameters. In this project the waiting time(Amount of time spent as ready to run) and turnaround time(Mean time from submission to completion of process) are considered for measuring the newly proposed algorithm with other existing ones.

As we know the schedulers can be classified as long-term scheduler(LTS), Medium term scheduler(MTS) and short term scheduler(STS)[6]. In which LTS is executed less frequently and which is responsible for admission of new process to the system, MTS is executed more frequently than LTS and is responsible to control the number of processes that are in main memory and also to control the temporary removal of process from memory. Finally STS is most frequently executed scheduler and it is actually responsible for assignment of CPU to ready process. We can also observe that most of the research and developments are on the study of short term schedulers, since they are the schedulers which can decide the process scheduling at the leaf level. The newly proposed algorithm for scheduling will also falls under the category of STS.

As a part of discussion about the most integral part of this Neelsack algorithm, the discussion is on the knapsack algorithm which is the vital part of this algorithm. Knapsack is basically a dynamic computer programming technique, which is based on the analogy for the problem that a robber has a bag with capacity(C kg) and he has to chose and fill that bag with the available items, so that he gets maximum profit. Among the n items with different values(V1,V2,V3,......,Vn) and different weights(W1,W2,W3,....,Wn) the decision should be made. The knapsack algorithm is the solution for this classical problem. It has two kinds, one is fractional, where robber can break the item and can take the fractional part of it and other is 0-1 knapsack, where robber can't break the item and the only option he has is to take the full item or drop idea of taking it. The Neelsack algorithm is using the first kind of knapsack(fractional). The reason for using the fractional knapsack is that, the process can be served by dividing its burst time, since there is no restriction that the process should be served completely at once(allocating CPU for its full burst time). It's clearly explained in the later sections.

3. Working Principle

As discussed, the basic ideology is completely based on using the knapsack technique in process scheduling. The Neelsack scheduling relates the knapsack terminologies into process scheduling terminologies. The weight of the bag in knapsack is assumed to be the capacity of the CPU here, then the items in knapsack problem are assumed to be the processes in this case and the weights of the items are assumed to be the burst time of that respective process and also items value is considered as priority given to the process. The main theme of using this special analogy is that the purpose of knapsack algorithm is that it chooses the best possible item for the robber so that he gets maximum profit and the purpose of the scheduling is also selecting the right process for CPU by considering its priority, so since the purpose is same but application is for different purpose, hence blending of the one of the dynamic technique for this new scheduling makes this technique of process scheduling very efficient.

There are few steps the Neelsack algorithm follows.

3.1 Read input for processing.

For this algorithm we need few parameters like number of process in CPU queue, their respective burst time and priority.

3.2 Calculation of capacity in each iteration

This step is the decision making step of entire algorithm, as the important parameter i.e., capacity is decided here.
It is calculated as the ratio of sum of BT of all the processes and number of processes at current iteration. It is noteworthy that, at each iteration capacity for the knapsack algorithm is decided dynamically, and also it keeps on changing as the BT of the each process and also the number of processes need to be served at current iteration keeps on changing. This usually goes on decreasing for each iteration. Unlike round robin where the quantum(capacity in Neelsack case) is constant, the capacity keeps changing dynamically based on the service it needs to provide. This dynamic nature also boosts the performance.

3.3 Application of Knapsack.

In each iteration after the calculation of capacity knapsack technique is applied as the part of this algorithm. This is done with the analogy on matching parameters. It is as follows.
i)Number of process: Number of items.
ii)Burst time of process: Weights of items.
iii)Priority of process: Cost of items.

3.4 Inversion of priority.

This step also has its own significance. In this step the priority is inverted(multiplied by -1) in the sense positive number to negative and vice versa(value remains same). This step is important because it ensures that each process is given importance in case of allocation, and thereby avoiding the starvation[7] of least priority process unlike priority based scheduling. If the process has been served completely than it will nullify the priority of that process, so it will not participate in the next iterations.

There after again the iteration repeats the step 3.2, step 3.3 and step 3.4 until all the processes gets CPU allocation for execution.

4. Algorithm

Pseudo Code: Neelsack Scheduling Algorithm.

```
Input: a)Number of processes(n).
       b)Burst time(BT) and
       c)Priority(PR) for each process.
Output: CPU allocation pattern for each
        process.
Step 1:Read number of processes.
Step2 :Read burst time and priority of each
       process respectively.
Step 3:Calculate initial β value(Sum of BT
       of all the processes/Number of
       processes).
Step 4:Apply Knapsack algorithm(With
       Replacement assumptions of input
       parameters:
       a) Number of processes(n) as Items in
       the knapsack.
       b) Burst time(BT) of each process as
       the weight of the each process.
       c) Priority(PR) of each process as the
       cost of the each process).
Step 5: Invert the priority of processes which
        got CPU for current iteration
        (multiply priority value by -1) and
        nullify if that process is completely
        served.
Step 6:Goto Step 3(Until all the processes
       have CPU and gets completed)
Step 7:Display result with CPU allocation
       pattern for each process.
```

Neelsack algorithm is an iterative scheduling algorithm. Iteratively this uses knapsack algorithm. The execution or process flow of Neelsack can be as depicted in the Fig 1.

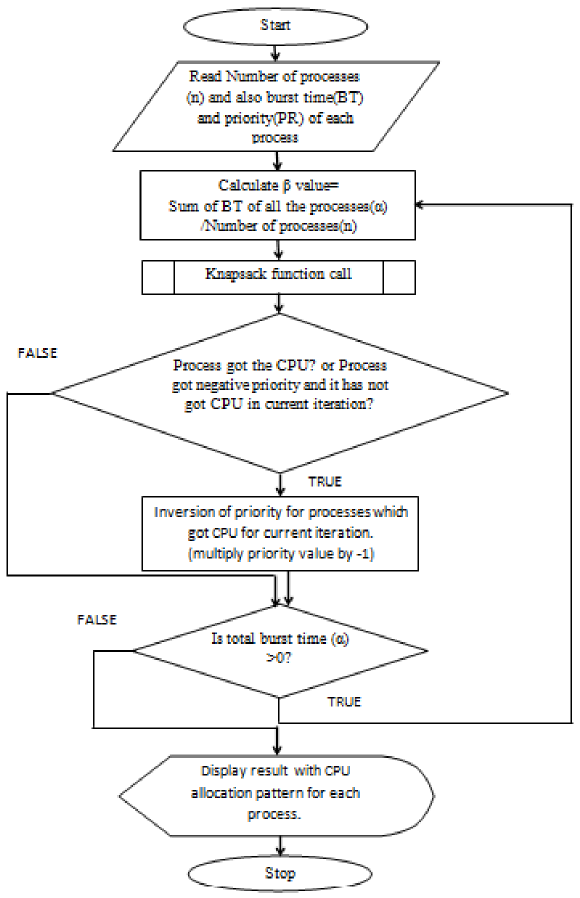
Start
Read Number of processes (n) and also burst time(BT) and priority(PR) of each process
Calculate β value= Sum of BT of all the processes(α) /Number of processes(n)
Knapsack function call
Process got the CPU? or Process got negative priority and it has not got CPU in current iteration?
FALSE
TRUE
Inversion of priority for processes which got CPU for current iteration. (multiply priority value by -1)
Is total burst time (α) >0?
FALSE
TRUE
Display result with CPU allocation pattern for each process.
Stop

Fig. 1 : Process Flow chart for Neelsack algorithm.

Process starts with accepting the input, that includes the number of process, their burst time(BT) and priority(PR) and followed by the calculation of capacity(β) as mentioned in the working principle section. The knapsack function called internally, using the logic and benefit of dynamic programming to find the best process(es) for which the CPU has to be allocated for the current iteration. Inversion of priority has to be done in few cases like when process gets CPU for execution, the clear reason behind this is to avoid starvation of any process. In case of the process has negative priority and also it has not got CPU in the current iteration for execution, the inversion of priority is applied for again the same reason of avoiding starvation.
This process iterates till the check of total burst time(β) is lesser than or equal zero.

5. Results

Measuring parameters like turnaround time and waiting time are considered. These parameters are measured in different cases and scenarios and also compared with the existing scheduling algorithms like FCFS and Round robin. For the purpose of depicting the quality of this algorithm few cases are demonstrated. In these random sample cases, the output from each iterations are noted and analyzed and the final result is compared with the FCFS and Round robin technique of scheduling.

Case 1: Random burst time and random priority

In this case as we can see the input in the table 1, we have 5 processes which have different burst time and also different priority . In fourth iteration all process will have 0 burst time left with null priority.

Table 1: Input for case 1

Process ID	***Burst Time(ms)***	***Priority***
P0	12	3

P1	2	1
P2	3	3
P3	2	4
P4	6	2

Initially calculation is for capacity

$$(\beta)=\alpha/n \qquad (1)$$

where $\alpha=(12+2+3+2+6)=25$.
i.e., $\beta=25/5=5$.

The knapsack analogy is applied using the Eq. (1) and it selects P2 and P3 with the capacity 5. Since these two process are completely served there priority has been nullified as in the table 1(a). This is the common process which continues till all process are served. Other major discussion is on rounding off the calculated value to the lower value, this is because the capacity can be in fractional and but burst time can't be. We can't make rounding to the higher value because capacity can't be increased. Hence we are making the rounding of the fractional value of capacity to lower value. This rounding is done iteration 2 and 4 of case 1.

Table 1(a): After 1st iteration(with β=5)

Process ID	***Burst Time(ms)***	***Priority***
P0	12	3
P1	2	1
P2	0	Null
P3	0	Null
P4	6	2

Table 1(b): After 2nd iteration(with β=6.66≈6)

Process ID	***Burst Time(ms)***	***Priority***
P0	12	3
P1	0	Null
P2	0	Null
P3	0	Null
P4	2	-2

Table 1(c): After 3rd iteration(with β=7)

Process ID	***Burst Time(ms)***	***Priority***
P0	5	-3
P1	0	Null
P2	0	Null
P3	0	Null
P4	2	2

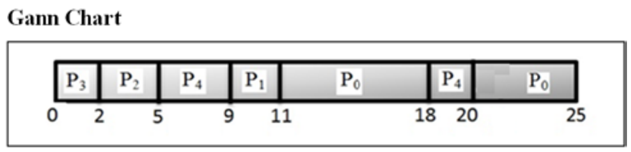

Fig. 2 : Gann chart depicted for case 1

Table 2(a): Case 1 turnaround time result comparison

Process ID	***Turnaround time(ms)***		
	Round robin	***Priority based***	***Neelsack***
P0	25	20	25
P1	7	2	11
P2	12	25	5
P3	23	8	2
P4	23	8	20
Avg. Turnaround time	15.4	15.6	12.6

Table 2(b): Case 1 waiting time result comparison

Process ID	***Waiting time(ms)***		
	Round	***Priority***	***Neelsack***

	robin	*based*	
P0	13	8	13
P1	5	0	9
P2	7	20	2
P3	10	23	0
P4	17	2	14
Avg. Waiting time	10.4	10.6	7.6

As in the Gann chart depicted in the Fig. 2. The P3 and P2(fully) are served first iteration and that the P4(partially) and P1(fully) in second iteration and in the third iteration P4(partially) and P0(fully). We can note that process are severed partially as you can see in case of P4, so that`s why it is using fractional knapsack as mentioned in the overview section.

Case 2: Random burst time and same priority.

In this case also working of algorithm is same as case 1. The input has different burst time but same priorities. In fifth iteration all process will have 0 burst time and left with null priority.

Table 3: Input for case 2

Process ID	*Burst Time(ms)*	*Priority*
P0	11	2
P1	18	2
P2	4	2
P3	13	2

Table 3(a): After 1st iteration(with β=11.5≈11)

Process ID	*Burst Time(ms)*	*Priority*
P0	4	-2
P1	18	2
P2	0	Null
P3	13	2

Table 3(b): After 2nd iteration(with β=11.67≈11)

Process ID	*Burst Time(ms)*	*Priority*
P0	4	2
P1	18	2
P2	0	Null
P3	2	-2

Table 3(c): After 3rd iteration(with β=8)

Process ID	*Burst Time(ms)*	*Priority*
P0	0	Null
P1	14	-2
P2	0	Null
P3	2	2

Table 3(d): After 4th iteration(with β=8)

Process ID	*Burst Time(ms)*	*Priority*
P0	0	Null
P1	8	2
P2	0	Null
P3	0	Null

Table 4(a): Case 2 time around time result comparison

Process ID	*Turnaround time(ms)*		
	Round robin	*Priority based*	*Neelsack*
P0	11	11	26
P1	44	29	46
P2	26	33	4
P3	37	46	32
Avg. turnaround time	29.5	29.75	27

Table 4(b): Case 2 waiting time result comparison

Process ID	***Waiting time(ms)***		
	Round robin	***Priority based***	***Neelsack***
P0	0	0	22
P1	26	11	28
P2	22	29	0
P3	33	33	19
Avg. waiting time	20.25	18.25	17.25

Gann Chart

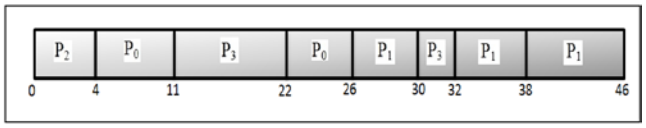

Fig 3: Gann chart depicted for case 2

In case of same burst time and same priority or same burst time different priority, all the three algorithms have same turnaround time and same waiting time(assuming the quantum for round robin is same as burst time). So in these cases we can conclude that all the three algorithms are equally efficient.

Result Comparison:

Case 1:

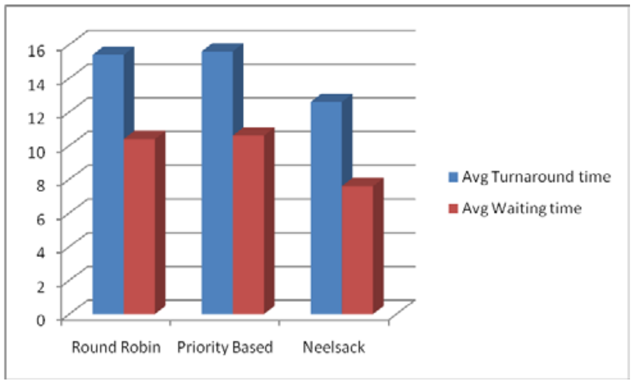

Fig 4: Result for Case 1

Case 2:

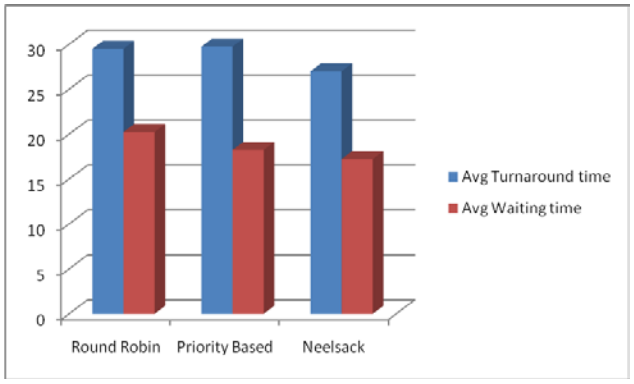

Fig 5: Result for Case 2

As in the result for both the cases as shown in the above Fig 4 and Fig 5, the result for both turnaround time and waiting time of Neelsack is better than the other two. So this can depict that the Neelsack algorithm can be implemented and also used as operating system process scheduler.

6.Scope

On successful implementation and deployment of this algorithm to the operating system work environment, it will have the direct impact on performance, as it is shown it is better than few other algorithms. Because of the drawbacks that no algorithm is proved best to use in all the scenarios. This newly designed algorithm can be used in some cases chosen by implementers, as in most of the cases it is better than the round robin and FCFS, which are individually good in some cases. It can have a major hand by playing a role of short term scheduler in the operations of measuring devices and can serve in many fields where processing time and performance is the prime factor.

7.Conclusion

As discussed in the abstract of this paper there are many existing short term operating systems schedulers, and they are efficient and powerful in certain cases(based on the measuring parameters). Any single algorithm can't be declared as the best at all cases. So through this new algorithm it is shown a new dimension for research on scheduling, observing the cube of problem in different dimensions. Mean while it's not possible to simulate any scheduling algorithm`s working accurately and exact performance can only be observed in live operating system operation. Solution to scheduling can be provided using lots of classical problems from the field of computing and it`s just one other to show case one useful technique. The results that are captured are just positive scenarios that shows this algorithm can be useful in few cases like others in few other cases. Since there are many factors like dynamic capacity selection and priority inversion, it showed significance impact on performance and it can be operating system implementers choice based on the hardware configuration and process handing mechanism used.

8.References

[1] International Journal of Advanced Research in Computer Science and Software Engineering, Volume 3, Issue 5, May 2013, ISSN: 2277 128X.

[2]International Journal of Computers and Distributed Systems Vol. No.3, Issue I, April-May 2013 by ANKUR BHARDWAJ.

[3] International Journal of Scientific Engineering and Research (IJSER): Volume 2 Issue 3, March 2014, by Mahima Shrivastava.

[4] David Pasinger Phd. thesis, February 1995,Dept. Of Computer Science, University of Copenhagen, Universitetsparken 1,DK-2100 Copenhagen, Denmark.

[5]http://community.topcoder.com/tc?module=Static&d1=tutorials&d2=dynProg - A TopCoder.com article by Dumitru on Dynamic Programming.

[6] A Comparative Study of CPU Scheduling Algorithms Neetu Goel: www.ifrsa.org.

[7]https://en.wikipedia.org/wiki/Starvation_(computer_science)

First Author Neelakantagouda S Patil, Bachelor of Engineering(2009-13) in Information Science and engineering from Visvesvaraya technological university(VTU, Belgaum), currently working as software engineer with Torry harris business solutions, Bangalore. My major research interest lies in the field of operating systems, data mining and image processing.