Comparing Priority and Round Robin Scheduling Algorithms

Alban Allkoci[1], Elona Dhima[2], Igli Tafa[3]

[1]Information Engineering Department
Polytechnic University of Tirana
Faculty of Information and Technology

[1]Information Engineering Department
Polytechnic University of Tirana
Faculty of Information and Technology

[1]Information Engineering Department
Polytechnic University of Tirana
Faculty of Information and Technology

Abstract

Computer usage has come increasing with time. Nowadays everybody needs to work with lots of programs at the same time and unfortunately this leads to slowing down the work. So software developers have to find a way how deal with workflows. At first, when I chose this theme, I was interested in how scheduling was done and with work I came to like and understand these algorithms. I decided to compare these two specific ones because they are more heard. In this article we describe scheduling algorithms in multitasking environments

Keywords: Scheduling, algorithm, multitasking environments

1. Introduction

Nowadays, most modern systems require to work with multitasking. This means that there will always be problems when having to process more than one process at a time. That's why it is used scheduling and Scheduling algorithms. It gives threads, processes the access to the resources they need. The scheduler chooses the next one to be admitted and run. There are tree types of schedulers:[1]

A)*Long-Term Scheduling* that decides which jobs to be admitted in the queue, waiting to execute when their turn comes.

B)*Medium-Term Scheduling* which removes processes from secondary memory. This may swap processes that have not been used or active for a long time or that have low priority.

C)*Short-Term Scheduling* decides which of the processes that are ready to execute should be run. So these decisions are made more frequently time after time.

Scheduling algorithms are the ones to decide which process waiting can access the resource
they are requesting and which not. This depends on the type of the algorithm and the way they work. I have studied only two of them, *priority scheduling and*

round robin. For those who have no idea about the way this algorithms work, I'll do a little explaining for each of the two.

I. Priority Scheduling [8]

This algorithm gives each process a priority well defined. This way every process has its own priority on which will depend if it is going to be run or wait. The first one to run is going to be the process with highest priority, while others will wait for their turn. This algorithm can be:

- Pre-emptive when the priority of the newly arrived process is compared to the process running and if its priority is higher it will occupy the CPU.
- Non-preemptive when the recently arrived process is positioned at the head of the queue.

Anyway, the drawback of this algorithm is indefinite blocking for lower priority processes which seem to never have the chance to be run. This is solved with a technique called *aging* which gradually increases the priority of the processes that have been waiting for a long time.

II. Round Robin Scheduling [6]

This is a priority free algorithm. Processes are given an equal time slice when they can be executed. The execution is done on a circular order, one after another. So each job has a quantum, time when it can be run. If this quantum is not enough for the process to finish its execution, it is stopped and the next process will be run. After a full round is completed, will come its turn again and so on. If a process is finished, it will go off the list and if another one comes, it will be positioned at the end of the list to wait its turn. This algorithm doesn't have starvation, but it can be often too long. All the problem is the time quantum given to each process. It should not be too large nor too short or the algorithm might degrade to FCFS(First Come First Served) or it just might take too much time to finish.

We are going to make some really easy and understandable testing to compare 3 or more processes execution time.

2. Related works

Since scheduling algorithms are very important for a good job, so it is normal that there have been earlier works about them. Since there ate not only these three scheduling algorithms, there ate lots of papers that deal with problems like the one I'm dealing. I will mention only few of them, the ones which in my opinion are more interesting.

One by Dan McNulty, Lena Olson and Markus Pelaquin compares these algorithms for multiprocessors. They have chosen to compare three other algorithms, Earliest-Deadline First, P-Fair algorithm and LLREF, which is an empiric improvement of the first two. [12]

Another one that attracted my attention was a paper from Amity University in India. It is about a mixture of Priority and Round Robin Algorithms in order to meliorate the drawbacks of one another. [7]

One more I read was from M.Kaladevi, M.Phil, S.Sathiyabama which deals with real time tasks. So it is important not only to finish the job as soon as possible, but the most important part is finishing it within the deadline.

One very interesting paper is one about lottery algorithm. It was written by O.Moonian and G.Coulson. The authors try to work with a lottery algorithm which isn't so propabilistic but is more for a high response time solution.

3. Theory of experiment

In this paragraph will be described the work I have done with the scheduling algorithms. I have experimented with two very simple algorithms respectively for priority and round robin scheduling.

3.1Necessary conditions for scheduling algorithms

In order to experiment with the algorithms, I had to do several tests. As each one of them has its own specifications, it is normal that the conditions they require are different. So for Priority Scheduling I tried to give as input three different processes and with their respective priorities. *I want to make clear that the processes I have used are just random names and numbers that crossed my mind and not real ones.*
For Round Robin algorithm were used three processes and the time they need be processed.

3.1.1 The environment and programming languages

The operating system where the experiment was held is **Windows7 OS .**
The programming language is a very important choice. I chose C language for pure personal preference. I might have chosen Java for programming but for the moment C seemed a more appropriate one.

4. Experimental phase

In this section I'll present a quick view of the experiments I have done. Everything consists in some tests I have realized with each of the algorithms. All the experiments were performed in the same computer, under the same conditions, so that nothing could affect the results. In order to get a more reliable result I have done different experiments with each of the algorithms, the results of which I will show summarized in a table below.

4.1 Priority Algorithm

The priority algorithm first, takes the names of the processes and their respective priorities. After that, it compares the priorities with each other and catches the process with the highest priority and executes it, while the others wait. The waiting time increases with every process executing. So at the end the waiting time for the process with the lowest priority will be the larger one. When all the calculations will be done, will be printed the **waiting time, turn around time, the average turn around time and the average waiting time.**
Below it is a simplified diagram of this algorithm.

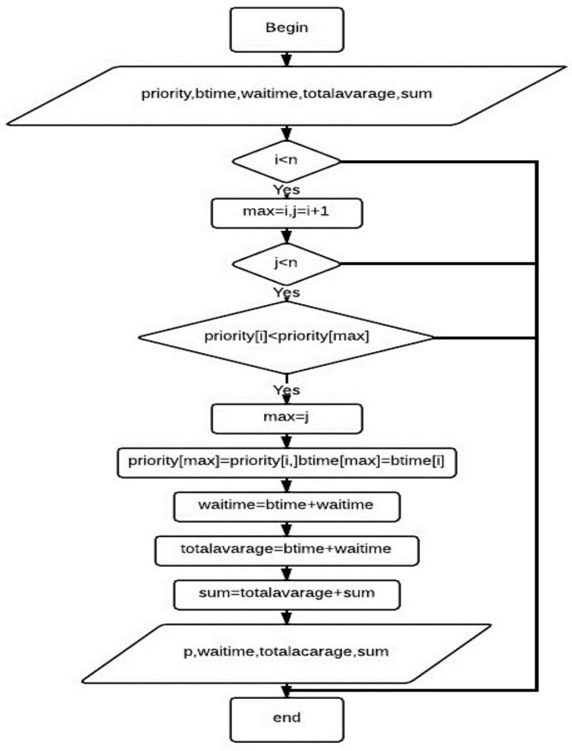
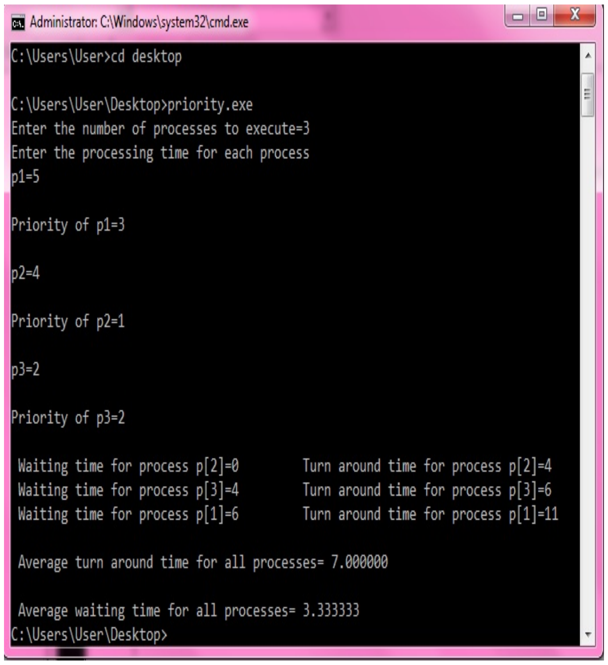

Begin
priority,btime,waitime,totalavarage,sum
i<n
Yes
max=i,j=i+1
j<n
Yes
priority[i]<priority[max]
Yes
max=j
priority[max]=priority[i,]btime[max]=btime[i]
waitime=btime+waitime
totalavarage=btime+waitime
sum=totalavarage+sum
p,waitime,totalacarage,sum
end

Figure 1: Priority Algorithm scheme



```
Administrator: C:\Windows\system32\cmd.exe
C:\Users\User>cd desktop

C:\Users\User\Desktop>priority.exe
Enter the number of processes to execute=3
Enter the processing time for each process
p1=5

Priority of p1=3

p2=4

Priority of p2=1

p3=2

Priority of p3=2

 Waiting time for process p[2]=0        Turn around time for process p[2]=4
 Waiting time for process p[3]=4        Turn around time for process p[3]=6
 Waiting time for process p[1]=6        Turn around time for process p[1]=11

 Average turn around time for all processes= 7.000000

 Average waiting time for all processes= 3.333333
C:\Users\User\Desktop>
```

Figure 2: Screenshot of Priority Algorithm

4.2 Round Robin Algorithm

The way this algorithm works is nearly the same as the priority algorithm. First, it takes the name of the processes and their processing times. After that for every 3 seconds all processes will be executed until they have finished. For every process will be calculated the execution time and the time they have to wait to finish executing. When is all over, it will be calculated the **total waiting and the average waiting time**. All this is going to be printed. I have made a simple (as simplified as possible) diagram of this algorithm in order to be understood better.

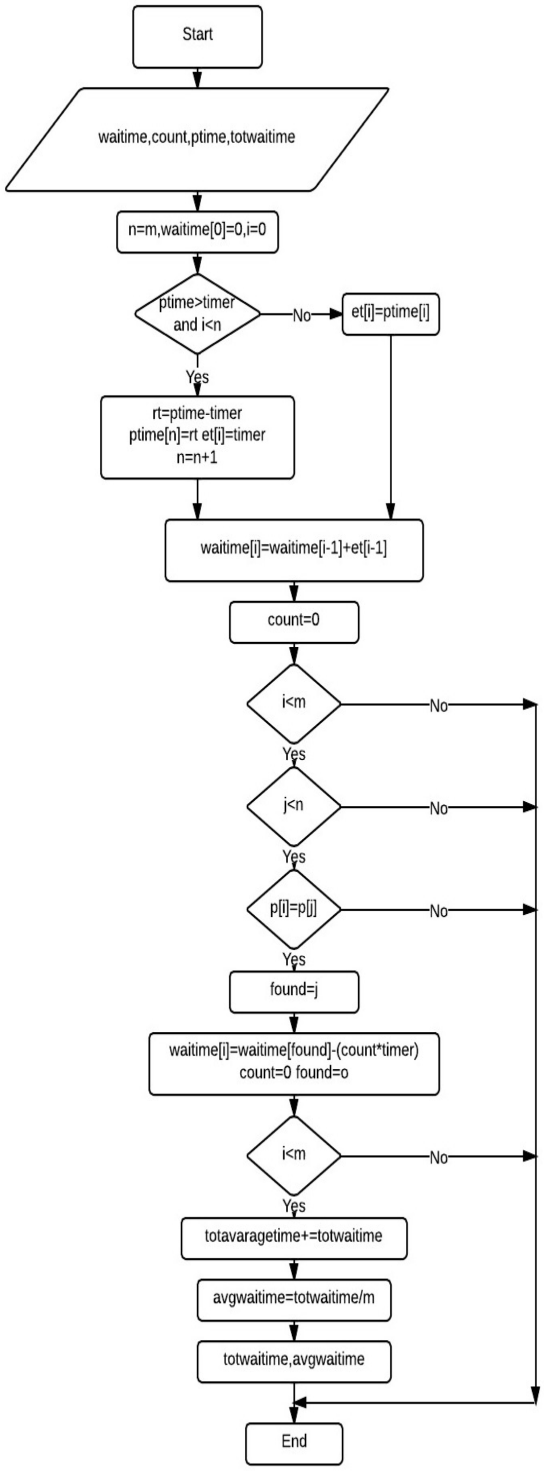
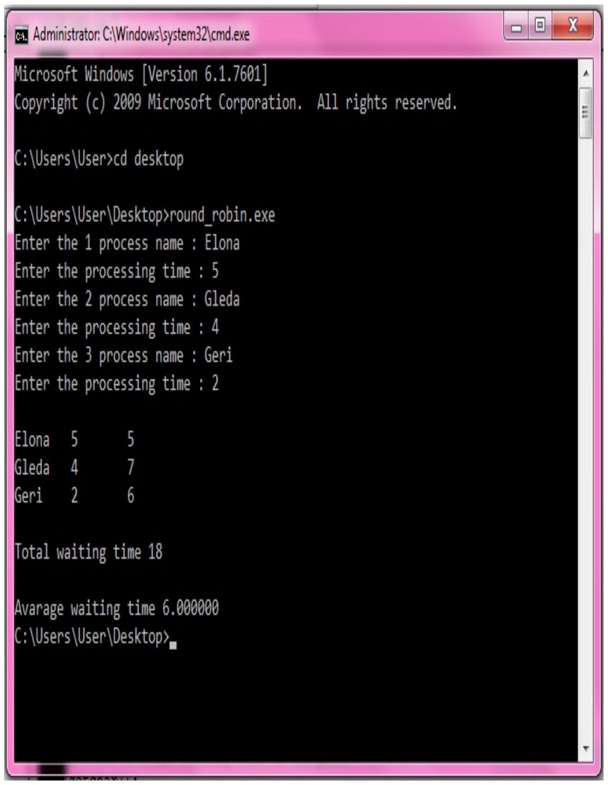


Figure 3: Round Robin Algorithm scheme



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\User>cd desktop

C:\Users\User\Desktop>round_robin.exe
Enter the 1 process name : Elona
Enter the processing time : 5
Enter the 2 process name : Gleda
Enter the processing time : 4
Enter the 3 process name : Geri
Enter the processing time : 2

Elona   5      5
Gleda   4      7
Geri    2      6

Total waiting time 18

Avarage waiting time 6.000000
C:\Users\User\Desktop>_
```

Figure 4: Screenshot of Round Robin Algorithm

4.3 Comparing the algorithms

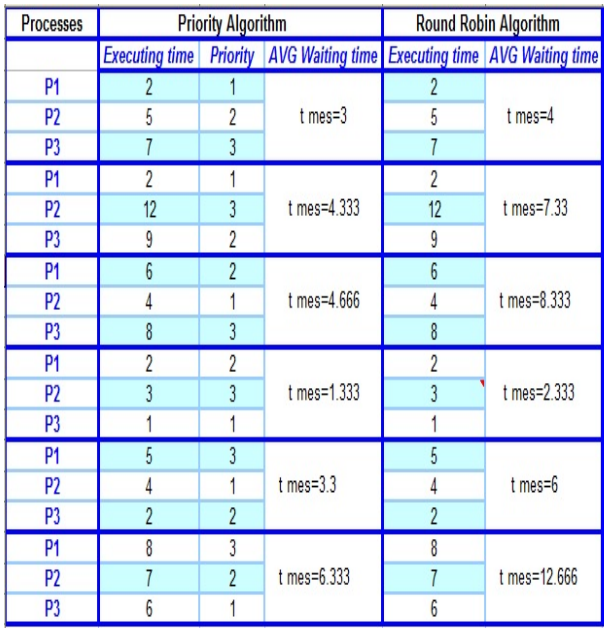
In order to make real comparisons between algorithms I have made several experiments with each of them. To be more precise I have put the results in a table so that they can be understood better.

Since these two algorithms are very different from each other the only way I have thought is to give processes of both algorithms the same execution time. But considering that one has to do with priority and the other no, I did some calculations of my own. So I chose the

executing time and the priorities so that the turn processes execute one after the other is nearly the same for both algorithms. I want to make clear once more that the numbers I have chosen and the process priorities are totally random numbers. I have chosen to experiment with tree processes in order not to complicate the things.

All this are shown in the table above.

Table1: Algorithm Results



Processes	Priority Algorithm			Round Robin Algorithm	
	Executing time	Priority	AVG Waiting time	Executing time	AVG Waiting time
P1	2	1	t mes=3	2	t mes=4
P2	5	2		5	
P3	7	3		7	
P1	2	1	t mes=4.333	2	t mes=7.33
P2	12	3		12	
P3	9	2		9	
P1	6	2	t mes=4.666	6	t mes=8.333
P2	4	1		4	
P3	8	3		8	
P1	2	2	t mes=1.333	2	t mes=2.333
P2	3	3		3	
P3	1	1		1	
P1	5	3	t mes=3.3	5	t mes=6
P2	4	1		4	
P3	2	2		2	
P1	8	3	t mes=6.333	8	t mes=12.666
P2	7	2		7	
P3	6	1		6	

5. Conclusions

As you can see from this table, the average waiting time for Round Robin Algorithm is considerably larger than for Priority Algorithm. This means that if we use Round Robin, the processes will need more time to finish executing compared with Priority. For the examples I have chosen, can be noticed that the average time for the first algorithm is nearly twice the time of the other algorithm. It seems like not a big deal, but in fact it is since there are only three processes. Imagine what would happen if there were much more.

So all in all I can say fully convinced that if I hade to choose between these algorithms, **based on the experiments I did**, I would definitely use the Priority Algorithm with its pros and cons.

6. Future work

I came to like this work, so I might think not to let everything like this.

I might do a comparison of other scheduling algorithms, and maybe, who knows in the distant future I might think of a brand new algorithm with no drawbacks for scheduling or an algorithm where processes are treated like equal to each other.

6. References

[1]http://en.wikipedia.org/wiki/Scheduling_(computing)
[2]http://web.cs.wpi.edu/~cs3013/c07/lectures/Section05-Scheduling.pdf
[3]http://www.moreprocess.com/process-2/priority-lottery-fair-share-scheduling-algorithm
[4]http://lass.cs.umass.edu/~shenoy/courses/fall10/lectures/Lec06.pdf
[5]http://inst.eecs.berkeley.edu/~cs162/sp11/sections/cs162-sp11-section5-answers.pdf
[6]http://siber.cankaya.edu.tr/OperatingSystems/ceng328/node125.html
[7]http://ijiet.com/wp-content/uploads/2012/11/1.pdf
[8]http://siber.cankaya.edu.tr/OperatingSystems/ceng328/node124.html
[9]http://www.personal.kent.edu/~rmuhamma/OpSystems/Myos/prioritySchedule.htm
[10]http://compfuter.wordpress.com/category/process-scheduling-c-programs/
[11]http://electrofriends.com/source-codes/software-programs/c/c-program-for-priority-cpu-scheduling-algorithm/
[12]http://pages.cs.wisc.edu/~markus/750/smp_scheduling.pdf

7. Appendix

7.1 Priority Algorithm [11]

```
#include<stdio.h>
int main()
  {
     int process[30],priority[30],btime[30],temp,max,waitime[30],
     totalavarage[30],sum=0,sum2=0,i,j,n;
     float avgwait,avgturn;
     waitime[0]=0;
     printf("Enter the number of processes to execute=");
     scanf("%d",&n);
     printf("Enter the processing time for each process");
     for(i=0;i<n;i++)
  {
  process[i]=i+1;
  printf("\np%d=",i+1);
  scanf("%d",&btime[i]);
  printf("\nPriority of p%d=",i+1);
  scanf("%d",&priority[i]);
}

     for(i=0;i<n;i++)
    {   max=i;
     for(j=i+1;j<n;j++)
    {
     if(priority[j]<priority[max])
     max=j;
    }
     temp=priority[max];
     priority[max]=priority[i];
     priority[i]=temp;
     temp=btime[max];
     btime[max]=btime[i];
     btime[i]=temp;
     temp=process[max];
     process[max]=process[i];
     process[i]=temp;
    }

    for(i=0;i<n;i++)
     {      waitime[i+1]=btime[i]+waitime[i];
     totalavarage[i]=btime[i]+waitime[i];
     sum+=totalavarage[i];
     sum2+=waitime[i];
     }
     avgturn=(float)sum/n;
     avgwait=(float)sum2/n;

      for(i=0;i<n;i++)
  {
  printf("\n         Waiting         time         for         process
p[%d]=%d",process[i],waitime[i]);
  printf("\t       Turn        around        time        for        process
p[%d]=%d",process[i],totalavarage[i]);
  }
  printf("\n\n  Average  turn  around  time  for  all  processes=
%f",avgturn);
  printf("\n\n Average waiting time for all processes= %f",avgwait);
  return 1;
}
```

7.2 Round Robin Algorithm[10]

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
#include<string.h>
int main(){
  char process[10][10];
int exetime[10],waitime[10], timer=3,count,ptime[10], remainingtime
,i,j,totwaitime=0, t,n=3,found=0,m;
  float avgwaitime;
  for(i=0;i<n;i++){
        printf("Enter the %d process name : ",i+1);
        scanf("%s",&process[i]);
        printf("Enter the processing time : ");
        scanf("%d",&ptime[i]);
        }
    m=n;
    waitime[0]=0;
    i=0;
    do{
        if(ptime[i]>timer)
        {remainingtime=ptime[i]-timer;
        strcpy(process[n],process[i]);
        ptime[n]=remainingtime;
        exetime[i]=timer;
        n++;  }
        else{
        exetime[i]=ptime[i];
        }
    i++;
    waitime[i]=waitime[i-1]+exetime[i-1];
    }while(i<n);

count=0;
for(i=0;i<m;i++)
    {
    for(j=i+1;j<=n;j++)
    {
if(strcmp(process[i],process[j])==0)
    {
    count++;
    found=j;
    }
}
if(found!=0)
          {
waitime[i]=waitime[found]-(count*timer);
          count=0;
          found=0;
          }
}
for(i=0;i<m;i++)
    {
    totwaitime+=waitime[i];
    }
avgwaitime=(float)totwaitime/m;
for(i=0;i<m;i++)
    {
    printf("\n%s\t%d\t%d",process[i],ptime[i],waitime[i]);
    }
printf("\n\nTotal waiting time %d\n\n",totwaitime);
printf("Avarage waiting time %f",avgwaitime);
getchar();
}
```