Comparison of process planning algorithms in order to determine their effectiveness in an operating system through a web simulation

Nancy Yaneth Gelvez García #1, Danilo López Sarmiento #2, Nelson Vera Parra #3

#1 Full Time Professor at Universidad Distrital Francisco José de Caldas, Faculty of Engineering, Bogotá (Colombia-South America)
nygelvezg@udistrital.edu.co

#2 Full Time Professor at Universidad Distrital Francisco José de Caldas, Faculty of Engineering, Bogotá (Colombia-South America)
dalopecs@udistrital.edu.co

#3 Full Time Professor at Universidad Distrital Francisco José de Caldas, Faculty of Engineering, Bogotá (Colombia-South America)
everap@udistrital.edu.co

Abstract — This article seeks to compare process planning algorithms in operating systems; for this purpose test cases are defined for each of them, setting different processes which have a lifetime, resource and priority in order to perform an analysis and comparison. Similarly, graphical tools are used to contrast performance and lifetime. Advantages and disadvantages of each algorithm are evidenced based on the metrics set for processing algorithms within a web simulation.

Keywords — Planning algorithm, Operating Systems, processes, performance, metrics, resource, priority, appropriative/preemptive, non-appropriative/non-preemptive, queues, gantt.

I. INTRODUCTION

Most computer systems require an operating system as interface to display the processes running and the hardware that they will use for their implementation during the set lifetime. Thus it can be stated that an operating system is a set of programs or software that manages system resources and serves as the interface for the user to perform any actions on the system. In short, the operating system together with the hardware form a computer system. [1, 2]

Today modern operating systems make use of different methods of planning processes in order to perform essential tasks for them, thus we can assume they are becoming increasingly complex, due to the ease of switching from a single-task environment to a multitask one [3]. That is, through its initial configuration the operating system proceeds to execute processes and assign them to the CPU, ensuring fairness and avoiding inanition at all costs, in order to allow a series of processes at the same time, maximize CPU utilization of the CPU and switch between different processes with such a high frequency that users can interact with each program while it executes [4].

With this background, and understanding that operating systems are a vital part of any computer system, it opens up to the large and complex world that encompasses everything related to planning processes along with planning algorithms. It can be said that what this process planner seeks is to meet these objectives, selecting a process available (within a set of multiple processes) for it to be executed in the CPU. It is important to note that in single processor systems, there will never be more than one process running: if more, it will have to wait for the CPU to become available and another process may be assigned to it [5].

Knowing that the operation of each of these algorithms is based on the concept of queues that host all processes of the computer system, we proceed to develop a study showing and clarifying the notions of the most used planning algorithms, in addition to their efficiency and effectiveness when addressing various processes in the system. This simulation aims to make a comparison through their implementation in a web environment.
II. THEORETICAL FRAMEWORK

A. Processes

It could be argued that the processes are the vital unit of the whole operating system, therefore they are the most important concept that is immersed within an operating system, because they are a small abstraction of a running program that has the ability to operate concurrently, even when there is only one CPU available [1].

Also noteworthy is that every process executed changes status as time passes and this is defined as the current activity of that process. Several types of statuses (Fig. 1) which are defined as:

- **Execution:** The process currently running.
- **Ready:** processes waiting to be executed.
- **Blocked:** processes that can not be run until a certain event ends, an I/O operation.
- **New:** a process that was just created.
- **Completed:** a process that has completed its function, was executed and completed.

![Fig. 1. Five-status model. [6]](image)

Some examples of processes stand out, such as word processors, a web browser and an email program, seeking to achieve a result defined in terms of concrete actions to be performed within a given time [7]. Such processes are often associated with certain properties such as name, resources needed, runtimes or probable completion, priority, etc. [8].

B. Process planning

Each time the operating system is executed, the user may or may not make the decision to execute one or more processes simultaneously, then each process is assigned a ready state and arranged to be serviced by the CPU. This decision which is a part of the operating system is known as process planner and the algorithm used to service the processes that are in the ready queue is known as planning algorithm [1].

Not surprisingly, depending on the environment different planning algorithms are required based on the objectives that it wants to meet. Consequently, optimization within the planner is a subjective concept depending on the system that uses it [1].

Then, throughout the centuries a series of algorithms have been developed that according to their nature serve different purposes. They are governed by the precept of what each of them should do and what a good planning algorithm should have, according to the goal that must be accomplished they will be desirable or not, within the application environment [1, 5], below some of the basic criteria will be explained:

- **Utilization of the CPU:** desire to keep the CPU as busy as is possible.
- **Processing rate:** the number of processes that pass over to the completed queue per unit of time.
- **Runtime:** how long it takes to execute a process. Defined as the interval from the instant the process execution is ordered to the time it is completed.
- **Wait time:** the sum of the cost in time of some processes while waiting in the ready queue. Note that the CPU planning algorithm only affects the time that a process invests waiting in the ready queue.
- **Response time:** time elapsed since the request is submitted until the first response occurs.

Having clarified these concepts, it is said that the desired objective is to maximize CPU utilization and processing rate, minimizing the wait time, the execution time and response time. But we should also consider a very important feature in planning algorithms and that is the type of discipline applied, which are divided into [9]:
• Appropriative: One the CPU has been assigned a process, it cannot be withdrawn, i.e., it is non-ejecting.
• Non-appropriative: One the CPU has been assigned a process, it can be withdrawn, i.e., it ejects the process.

III. SIMULATION DEVELOPMENT

Having introduced the basic concepts of operating systems, we proceed to do a web simulation where planning algorithms are evaluated: Round Robin, Shortest Job First (SJF), Short Time Remaining First (SRTF), planning by appropriative and non-appropriative priority, multiple queues and multiple queues with feedback.

In order to evaluate the individual behavior of each of the planning algorithms, a metric is set based on the one proposed by "Tanenbaum" [1] with certain modifications. In Equation 1, we can see clearly how each of the criteria is defined for this particular exercise as it is important to note that the five-status model will not be used entirely, therefore it does not apply fully. However the same concepts previously explained for planning algorithms are used.

\[ \text{Response time (t)} = \text{time elapsed since the creation of the process until it is served on the CPU} \]
\[ \text{Process time} = \text{lifetime} \]
\[ \text{Wait time (TE)} = \text{Response time} - \text{Process time} \]
\[ \text{TimeP} = \text{TE} + t, \text{where t equals performance} \]
\[ \text{Penalty (P)} = \frac{\text{Response time}}{\text{Process time}} \]
\[ \text{Response portion} = \frac{1}{\text{Penalty}} \]
\[ \text{Total time} = \text{sum of total time of running processes} \]
\[ \text{CPU utilization} = \frac{(\text{total time} \times 100)}{\text{execution time of simulation}} \]

Equation 1

For clarity, the statuses that are handled in this simulation are as follows:

- **Completed:** the process successfully completed its lifetime or quantum, as appropriate.
- **Ready:** the process is waiting to be executed.
- **Blocked:** the process cannot run and will remain in this status until the resource it needs to fulfill its function is released.
- **On hold:** the process was interrupted by an action that occurred within the system and will have to wait until the penalty time stipulated for this queue is met.

Additionally, to measure and calculate the effectiveness of each of the planning algorithms which is one of the pillars of this study, the following Equation is proposed:

\[ \text{Effectiveness} = \frac{(\text{Wait time} - \text{Time in CPU})}{\text{Total processes}} \]

Equation 2

Applying these criteria will allow not only to give qualitative values on the performance of each of these planning algorithms, but also assign a quantitative value which will allow to conclude which one is the best time to avoid process inanition. Consequently, for the development of this simulation it is crucial that the same processes are used for all and each one of them, as shown in Table 1, in order for the simulations to be under the same conditions ensuring the accuracy and validity of the results.
### TABLA I. Process description by processor

| NAME | LIFETIME | PRIORITY | RESOURCE |
|------|----------|----------|----------|
| P0   | 34       | 1        | Printer  |
| P1   | 54       | 3        | Speakers |
| P2   | 23       | 1        | Keyboard |
| P3   | 65       | 2        | Printer  |
| P4   | 32       | 3        | Printer  |
| P5   | 76       | 1        | Mouse    |
| P6   | 54       | 2        | Keyboard |
| P7   | 12       | 3        | Mouse    |
| P8   | 35       | 3        | Speakers |
| P9   | 83       | 2        | Keyboard |

### Processor 2

| NAME | LIFETIME | PRIORITY | RESOURCE |
|------|----------|----------|----------|
| P0   | 90       | 1        | Speakers |
| P1   | 76       | 2        | Printer  |
| P2   | 35       | 1        | Speakers |
| P3   | 89       | 2        | Mouse    |
| P4   | 70       | 3        | Mouse    |
| P5   | 47       | 2        | Speakers |
| P6   | 85       | 2        | Keyboard |
| P7   | 13       | 3        | Screen   |

### Processor 3

| NAME | LIFETIME | PRIORITY | RESOURCE |
|------|----------|----------|----------|
| P0   | 78       | 1        | Speakers |
| P1   | 42       | 3        | Keyboard |
| P2   | 67       | 2        | Keyboard |
| P3   | 45       | 3        | Mouse    |
| P4   | 86       | 1        | Printer  |
| P5   | 94       | 2        | Printer  |
| P6   | 23       | 1        | Printer  |
| P7   | 13       | 2        | Screen   |
| P8   | 54       | 2        | Screen   |
| P9   | 87       | 3        | Mouse    |
| P10  | 32       | 2        | Printer  |
| P11  | 72       | 1        | Keyboard |
| P12  | 49       | 3        | Speakers |

#### A. Round Robin

One of the oldest, most simple, impartial and widely used algorithms is Round Robin or circular shift. In which each process is assigned a small unit of time, known as quantum, which ensures that the CPU is shared equally among all processes and there is not a process that monopolizes. The Quatum decreases only when the process is in the CPU or critical zone, once this portion of time, if the process still has lifetime, this becomes suspended state and the quantum will be recalculated to go again to the ready queue. Otherwise, the process state passes over [10]. Also noteworthy is that the processes in the ready queue are governed by the concept of FIFO (First Input, First Output), first in, first out.
One of the most interesting questions of this algorithm is the length of time quantum and appropriation of the CPU until the quantum of time has expired. Despite being these innovative features in its infancy, this creates a major performance problem, because of the dependence you have a good estimate of the quantum; so, if it is small, all processes will be interrupted several times, affecting processing speed [11].

Then in Fig. 2, the results obtained from simulation and values that throw its metric is.

B. Shortest Job First (SJF)

This algorithm gives priority to processes that have a shorter lifetime within the ready queue. Every time SJF is executed, the queue is order by lifetime for the one with the shortest time to be the first to run. As this is a non-ejecting or appropriative algorithm, the process that is in critical zone (CPU) will not be interrupted until completion or until an interruption occurs in the system [11]. One feature to highlight is that shall there be a tie, this is settled by FIFO.

It is important to understand that in the absence of a quantum, the process will not enter the on hold status unless an interruption occurs in the system. Excluding the above it can be said that the behavior of the blocked and completed queue is the same as R.R.

Because of its nature, it is possible to say that SJF is optimal in the sense that the wait time for processes that have less time is minimal, but processes with too large times will present a very high wait time, therefore this mechanism is said to be useful for long-term planning.

Fig. 2. Round Robin planning algorithm

Fig. 3 shows the results obtained after applying the SJF algorithm to a set of previously established processes.
C. Short Remaining Time First (SRTF)

It is an improved version of SJF and therefore, is very similar. The difference is that the design is based on a non-appropriative algorithm, i.e. it ejects the processes found in the critical zone depending on the condition to be established [11].

In this case the condition of removal is based on whether in the ready queue there is a process that has a shorter lifetime compared to that found in the CPU, then it will be interrupted and sent to the on hold queue until the penalty time finishes and it returns to the ready status with the remaining lifetime.

Fig. 4 shows the application of this algorithm in terms of the Gantt chart and its outcome in terms of the metric proposed.
D. Appropriative priority algorithm

This algorithm establishes an initial priority to all and each of the processes to be executed within the system, from a series of well-defined criteria:

- **High** – system process, also known as priority 1.
- **Medium** – processes performed by the user, also known as priority 2.
- **Low** – input and output processes, also known as priority 3.

Once this priority is established, processes will be served according to their importance in the processor, for this exercise, the top priority will be the system processes. Should there be a process running and in the ready queue there is a higher priority process, it cannot be served until the process in the critical area ends its lifetime [11]. Therefore, it can be said that this type of algorithm is similar to SJF, the only difference is their ordering criteria. A major problem with priority planning algorithms is the indefinite block or inanition, because the low-priority processes can stay waiting indefinitely to be served while there are still processes with higher priority [11]. In Fig. 5 the graphical and numerical implementation of this algorithm results are shown.

E. Non-appropriative algorithm with Priority

The non-appropriative algorithm with priority is a variation of the algorithm described in Section D, except this one has the ability to eject the process that is running in critical zone. If upon comparison with those found in the ready queue, there is a process with higher priority, this process will go into on hold status and upon ending its penalty time it will go back to the ready queue. In case there are multiple processes with the same priority, they will be served by FIFO. After running the algorithm, the results from the metric used were obtained, where inanition is less evident in the processes regarding the appropriative priority algorithm. For more clarity, see Fig. 6.

F. Multiple queues

This kind of planning algorithm is a solution developed to the problems that arise with the above algorithms. It is vital for operating systems where processes coexist with different needs. This algorithm is based on a predetermined scheme which gives special treatment to the processes according to the queue in which they are [11]. Here priorities are also handled, but this time the priority is in the queue, which means that there is a queue (ready, blocked and on hold) that has a higher priority over the others.

It is also worth emphasizing the appropriation of processes over the CPU and a defining characteristic of such algorithms, which specifies that until they find a queue (ready, blocked and on hold) that is completely free, they cannot move on to serve the following [11]. And this condition generates inanition processes in lower priority queues until they are fully served.

![Fig. 5. Appropriative priority planning](image-url)
G. Multiple queues with feedback

This algorithm is governed under the same concept of multiple queues, which allows the movement of processes from one queue to another when a process takes too long in the CPU or when its aging time is over, but the difference is that the CPU is not being appropriated, if for example, there is a process in the critical area that has priority 3 and one with lower priority reaches the system it will be rejected and put on hold. This means that this algorithm performs a comparison between the CPU and the ready queues to determine which process should be executed first by priority [11].

Multiple queues with feedback as a solution to the major problems of inanition and indefinite blocking apply the aging mechanism, a technique to gradually increase the priority of the processes that have been waiting a long time in the system [11]. That is, a process that has been waiting too long in line 3, upon ending its aging time will jump to line 2 and so on.
After execution of the planning algorithms, the simulation produced the following results that contrast performance versus runtime. It can be seen that time varies depending on the algorithm and in general processor 3 behavior tends toward better performance.

The Round Robin planning algorithm ran all its processes in 751 seconds and its performance on most processors tends to decrease over time (Fig. 9). The troughs show some kind of prolonged inactivity, therefore we can say that the algorithm is acceptable.

Fig. 9. Performance of Round Robin Planning Algorithm
On the other hand, the SJF algorithm and appropriative priority algorithm show better performance in CPU, since the peaks and troughs are not as pronounced and inactivity of the critical zone starts later with respect to the previous algorithm. Fig. 10 shows that the runtime for the proposed processes was 721.

Fig. 10. Performance of SJF and appropriative priority planning algorithms

The SRTF and non-appropriative algorithm with priority, as shown in Fig. 11, show greater variety in performance of the critical zone of each of the processors, as process ejection generates downtimes and strong activity peaks.

Fig. 11. Performance of SRTF and non-appropriative algorithm with priority
By nature, multiple queues have a greater runtime for process completion. As displayed in Fig. 12, this algorithm shows with more urgency a downtime in critical zones, but performance remains constant over time.

![Fig. 12. Performance of multiple queues algorithm](https://via.placeholder.com/150)

Finally, multiple queues with Feedback (Fig. 13) present a more uniform and consistent performance over time since having an aging time makes possible combating inanition processes easier.

![Fig. 13. Performance of multiple queues with feedback algorithm](https://via.placeholder.com/150)

By averaging the criteria that are involved in the metrics, it can be seen that some algorithms respond better due their nature as evidenced in Table 2, comparing their response time, wait time, penalty time and response rate. It shows that the SRTF algorithm has a better response in all items.

Given that the response time refers to the time it takes for the process to be served, this is one of the most important and conclusive metrics to determine the superiority of an algorithm. Having said that, SRTF and the non-appropriative with priority algorithms manage to optimize management of processor resources, minimizing in each of its processors the indefinite blocking and process inanition.

Another metric that can support the above statement is the "penalty" in which a more lenient process behavior is observed, so they do not waste time in queues designed to penalize but to concentrate more on ready queues.

Also in Table 2, it is clearly seen that there is no coincidence given by a context pattern, the similarity in the results obtained by the metric, but instead is a behavior that persists between processors 1, 2 and 3, where despite the change of initial conditions, number of process, lifetime and priorities, they remain around the arithmetic mean.
| Processor 1 | Name                  | Response time | Wait time | Penalty | Responses rate |
|-----------|-----------------------|---------------|-----------|---------|----------------|
|           | Round Robin           | 354.4         | 307.6     | 9.3     | 0.2            |
|           | SJF                   | 199.5         | 152.7     | 3.6     | 0.4            |
|           | SRTF                  | 153.7         | 107.4     | 3.0     | 0.5            |
|           | Appropriate with priority | 199.5     | 152.7     | 3.6     | 0.4            |
|           | Non-Appropriate with priority | 153.7   | 107.4     | 3.0     | 0.5            |
|           | Multiple queues       | 334.6         | 287.8     | 10.8    | 0.3            |
|           | Multiple queues with feedback   | 336.2      | 289.4     | 7.8     | 0.2            |
| Processor 2 | Name                  | Response time | Wait time | Penalty | Responses rate |
|           | Round Robin           | 405.4         | 342.3     | 7.7     | 0.2            |
|           | SJF                   | 241.8         | 178.6     | 3.4     | 0.4            |
|           | SRTF                  | 164.1         | 100.0     | 2.2     | 0.6            |
|           | Appropriate with priority | 241.8     | 178.6     | 3.4     | 0.4            |
|           | Non-Appropriate with priority | 164.1   | 100.0     | 2.2     | 0.6            |
|           | Multiple queues       | 310.3         | 247.1     | 8.5     | 0.3            |
|           | Multiple queues with feedback   | 269.0      | 205.9     | 5.0     | 0.3            |
| Processor 3 | Name                  | Response time | Wait time | Penalty | Responses rate |
|           | Round Robin           | 513.7         | 456.6     | 10.4    | 0.2            |
|           | SJF                   | 320.0         | 262.9     | 5.1     | 0.3            |
|           | SRTF                  | 195.9         | 137.6     | 3.0     | 0.5            |
|           | Appropriate with priority | 320.0     | 262.9     | 5.1     | 0.3            |
|           | Non-Appropriate with priority | 195.9   | 137.6     | 3.0     | 0.5            |
|           | Multiple queues       | 447.4         | 390.3     | 9.7     | 0.2            |
|           | Multiple queues with feedback   | 601.4      | 544.3     | 14.0    | 0.1            |
V. CONCLUSIONS

Appropriate planning algorithms such as Round Robin, significantly help improve processor performance, thanks to the fulfillment of the objectives of efficient planning, when partitioning process runtime. This addresses its shortcomings by merging with priority algorithms and aging strategies to reach the expected performance.

It is difficult to make a definitive comparison between all planning algorithms, because performance depends on several factors, including the probability distribution of process service times, planning efficiency and mechanisms of change of context.

It can be stated in regard to the simulation presented, that the calculation of quantum length directly affects the performance of the planning algorithm, for example, a short quantum with respect to process lifetime reduces CPU performance, and a long quantum negatively affects response times.

Processes that are assigned a low priority or a very long lifetime, show inanition in planning algorithms that are based on priorities and those sorted by lifetime.

REFERENCES

[1] A.S. Tanenbaum, Sistemas Operativos Modernos. Pearson Educación, México. 3 – 7. 2009.
[2] H.M. Deitel Introducción a los Sistemas Operativos. Addison-Wesley Iberoamericana, México. 1987.
[3] A. Singh, P. Goyal, S. Batra. An Optimized Round Robin Scheduling Algorithm for CPU Scheduling. International Journal on Computer Science and Engineering Vol. 02. 2383-2385. 2010.
[4] A. Silberchatz, P. BGalvin, G.Gagne. Operating systems concepts. John Wiley & Sons, Inc, Estados Unidos. 2003.
[5] A. Silberchatz, P. BGalvin, G.Gagne. Fundamentos de los sistemas operativos. Mc Graw Hill, España. 2005.
[6] H. J. Ortiz, Sistemas operativos modernos. Sello editorial, Colombia. 2005.
[7] J. A. Jiménez, J. J. Medel, Planificación por prioridades para sistemas SISO a través de un controlador. Latin-American Journal of Physics Education. Vol. 5, Nº. 1. 2010.
[8] A. S. Tanenbaum, M. V Steen Sistemas Distribuidos. Pearson Educación, México. 2008.
[9] D. Martínez, Sistemas Operativos: diseño e implementación. Martin Iturbide, Argentina. 1998.
[10] A. McIlver, JFlynn, Sistemas Operativos. Cengage Learning Editores. México. 2010.
[11] P. Martínez, M. Cabello, J. C. Díaz, Sistemas operativos: teoría y práctica. Ediciones Díaz de Santos. España. 1996.