

**Electronics and Computer Engineering (ECE)**

**"Final Project “**

|  |  |  |
| --- | --- | --- |
| No | Name | ID |
| 1 | Adel Saeed Mohamed Saad | 19105594 |
| 2 | Hesham Mamdouh | 19106431 |

**Fall 22**

**TA**: Silvana Atef

**List of Content**

[Introduction 3](#_Toc123559755)

[Scheduling Algorithms 3](#_Toc123559756)

[1) First Come First Serve: 3](#_Toc123559757)

[2) Shortest Job First 5](#_Toc123559758)

[3) Longest Job First: 6](#_Toc123559759)

[4) Priority Algorithm: 7](#_Toc123559760)

[5) Round-Robin 8](#_Toc123559761)

[Comparison 9](#_Toc123559762)

[Conclusion 11](#_Toc123559763)

**List of Figures**

[Figure 1 : Code of ( FCFS ) 4](#_Toc123559597)

[Figure 2 Rest of code for the ( FCFS )) 4](#_Toc123559598)

[Figure 3 part of code for (SJF) 5](#_Toc123559599)

[Figure 4 Code for the preemptive ( SJF ) 6](#_Toc123559600)

[Figure 5 Part of code for ( LJF ) 6](#_Toc123559601)

[Figure 6 code for priority algorithm 7](#_Toc123559602)

[Figure 7 part of code for the Round robin 8](#_Toc123559603)

**List of tables**

[Table 1 Cons vs pros for (FCFS) 9](#_Toc123559604)

[Table 2 Cons and Pros for Preemptive and the Non-Preemptive (SJF) 9](#_Toc123559605)

[Table 3 Cons Vs Pros for (LJF) 10](#_Toc123559606)

[Table 4 Priority Vs Round Robin 10](#_Toc123559607)

# **Introduction**

CPU scheduling is the process of deciding which of the processes in the ready queue will be executed by the CPU next. There are many CPU scheduling algorithms that have been developed to optimize the use of the CPU and improve the overall performance of the system. Some of the most common CPU scheduling algorithms are:

* First Come First Serve
* Shortest Job First
* Longest Job First
* Priority
* Round Robin

There are many other CPU scheduling algorithms that have been developed, and the choice of algorithm depends on the requirements of the system and the workload.

# **Scheduling Algorithms**

1. **First Come First Serve:** (FCFS) scheduling is the simplest and most basic CPU scheduling algorithm. It works by executing the processes in the order that they arrive in the ready queue.

* The operating system maintains a queue of processes that are ready to be executed.
* When a new process arrives in the ready queue, it is added to the end of the queue.
* The CPU executes the process at the front of the queue.
* Once the process has completed execution, it is removed from the queue.
* The next process in the queue is then executed by the CPU.

Text

Description automatically generated

Figure 1 : Code of ( FCFS )

The following code ensure that the all lists are sorted by the arrival time:

This code imports the NumPy library and defines three variables: arrival, burst, and priority\_list. It also creates a list processes\_count that contains the numbers 0 through 4 (the number of processes). The code then checks if the priority\_list variable is None. If it is not None, the code sorts the arrival, processes\_count, priority\_list, and burst variables by the arrival time and stacks them in a NumPy array. If the priority\_list variable is None, the code sorts the arrival, processes\_count, and burst variables by the arrival time and stacks them in a NumPy array.

Text

Description automatically generated

Figure 2 Rest of code for the ( FCFS ))

* The code defines a function called first\_come\_first\_serve that takes three required arguments: arrival\_time, burst time, and processes count. The \*\*kwargs syntax allows the function to accept any additional keyword arguments that may be passed to it.
* The function creates an empty dictionary called gantt and a variable called current\_time initialized to 0.
* The code enters a while loop that continues as long as there are any non-zero values in the burst\_time array.
* Inside the while loop, the code checks if there are any processes that have arrived and have not yet completed execution.
* If there are, it filters the burst\_time array to only include the arrived processes and gets the first non-zero value in the filtered array.
* It then adds a tuple representing the start and end time of the process execution to the Gantt dictionary. The value of the tuple is the process number.
* The code then sets the burst time of the completed process to 0 and updates the current time variable by adding the burst time of the completed process.
* If there are no arrived processes that have not yet completed execution, the code gets the arrival time of the next process, adds a tuple representing the start and end time of the idle period to the Gantt dictionary, and updates the current\_time variable to the arrival time of the next process.
* The function returns the Gantt dictionary, which contains the execution schedule for the processes.

1. **Shortest Job First** (SJF) : have Two Types :

* *Non-preemptive:* In non-preemptive SJF, the CPU executes the process with the shortest burst time to completion before moving on to the next process.
* *preemptive:* In preemptive SJF, the CPU can be interrupted to execute a process with a shorter burst time, even if the current process has not completed execution. This is achieved by allowing the scheduler to preempt the execution of a process if a process with a shorter burst time arrives



Figure 3 part of code for (SJF)

In the **Non-preemptive SJF** we implement the same function in **FCFS**, but we return the min burst time instead of the first job burst time .

Text

Description automatically generated

Figure 4 Code for the preemptive ( SJF )

In the **Preemptive SJF** we only add the previous part to out function, this part is executed when the **preemptive** arguments is set to **True**.

1. **Longest Job First:** (LJF) algorithm is a CPU scheduling algorithm that aims to minimize the average waiting time of processes by prioritizing the execution of processes with the longest burst time. In non-preemptive mode, once a process starts executing, it is not interrupted until it completes.



Figure 5 Part of code for ( LJF )

In the **Non-preemptive LJF** we implement the same function as **Non-preemptive SJF**, but we return the max burst time instead of the min burst time

1. **Priority Algorithm:** is a CPU scheduling algorithm that assigns priority levels to processes and executes the process with the highest priority first. In non-preemptive mode, once a process starts executing, it is not interrupted until it completes

* Sort the processes in the order of the highest priority first.
* Initialize a variable called current\_time to 0. This variable keeps track of the current time of the CPU.
* Start a while loop that continues as long as there are any non-zero values in the burst\_time array.
* Check if there are any processes that have arrived and have not yet completed execution. If there are, filter the burst\_time array to only include the arrived processes.
* Select the process with the highest priority from the filtered array and append a block to the gantt chart representing the execution of this process.
* Set the burst time of the selected process to 0 to indicate that it has completed execution.
* Update the current\_time variable by adding the burst time of the selected process.
* If there are no arrived processes or all processes have completed execution, add a block to the gantt chart representing the idle time of the CPU until the next process arrives.
* Update the current\_time variable to the arrival time of the next process.
* Repeat the process until all processes have completed execution.

Text

Description automatically generated

Figure 6 code for priority algorithm

In the **Non-preemptive Priority**,we implement the same function as previous. But we return the higher priority (from arrived processes) burst time.

1. **Round-Robin** Robin is a scheduling algorithm used in computer operating systems to distribute processing time fairly among all processes. It is designed to serve each process for a small unit of time called a time quantum, before moving on to the next process. The idea behind Round Robin is to ensure that no process is left waiting too long for its turn to be executed, while also ensuring that each process gets a fair share of the available processing time.

Here's how the Round Robin algorithm works:

* The operating system maintains a queue of processes that are ready to be executed.
* The scheduler selects the first process in the queue and executes it for a fixed time quantum.
* If the process finishes within the time quantum, it is removed from the queue and the scheduler moves on to the next process.
* If the process has not finished within the time quantum, it is placed at the end of the queue and the scheduler moves on to the next process.
* This process continues until all processes have been completed.

Text

Description automatically generated

Figure 7 part of code for the Round robin

In the Preemptive Round Robin, we implement the same function as FCFS with the quantum in consideration. Also, we implement this algorithm in a different way which the executed process is shifted lift using the circular\_shift function.

# **Comparison**

**Table 1 Cons vs pros for (FCFS)**

|  |  |
| --- | --- |
| Cons | Pros |
| Can lead to long wait times for processes that arrive later, as they must wait for all earlier processes to be completed | Simple to implement and understand |

**Table 2 Cons and Pros for Preemptive and the Non-Preemptive (SJF)**

|  |  |  |
| --- | --- | --- |
|  | **Cons** | **Pros** |
| Preemptive | Can be difficult to predict the length of a process, and may lead to longer wait times for processes with longer processing needs | Can lead to faster overall completion times, as shorter processes are executed first |
| Non-Preemptive | Can be difficult to predict the length of a process, and may lead to longer wait times for processes with longer processing needs | Can lead to faster overall completion times, as shorter processes are executed first |

**Table 3 Cons Vs Pros for (LJF)**

|  |  |
| --- | --- |
| Cons | Pros |
| Can lead to longer overall completion times, as longer processes are executed first | Can lead to shorter wait times for processes with longer processing needs |

**Table 4 Priority Vs Round Robin**

|  |  |  |
| --- | --- | --- |
|  | **Cons** | **Pros** |
| Priority | Can lead to lower priority processes being starved of processing time if higher priority processes are constantly being added to the queue | Allows the operating system to prioritize certain processes over others, leading to faster completion of important tasks |
| Round-Robin | Ensures that each process gets a fair share of the available processing time, and can be used with a variable time quantum | Ensures that each process gets a fair share of the available processing time, and can be used with a variable time quantum |

# **Conclusion**

**CPU scheduling algorithms** are a crucial component of operating systems that determine how efficiently the CPU is utilized. They allow the system to prioritize different processes and allocate resources, accordingly, ensuring that the CPU is used in the most efficient and fair way possible. There are a variety of different CPU scheduling algorithms that have been developed over the years, each with their own strengths and weaknesses. Some of the most common include first-come, first-served (FCFS), shortest job first (SJF), and round robin (RR). Ultimately, the choice of which CPU scheduling algorithm to use depends on the specific needs and goals of the system in question.