Text, logo

Description automatically generated **San Francisco Bay University**

**CE521 - Real-time Systems and Programming**

**Homework Assignment #5**

**Due day: 4/13/2022**

**Instructions:**

1. **Push the answer sheet to Github**
2. **Overdue homework submission could not be accepted.**
3. **Takes academic honesty and integrity seriously (Zero Tolerance of Cheating & Plagiarism)**
4. **Describe the popular cpu scheduling algorithms in the uniprocessor system by giving detailed examples**

CPU Scheduling algorithm is an algorithm which is used to assign system resources to processes in a computing system. Consider the case where you are using two apps namely a game like Fortnite and a desktop application like Evernote. Both with require the use of a graphics processor and but only one can use it at a time. It is the CPU scheduling algorithms which manages which process will use a given resource at a time. The focus of such algorithms is to maximize CPU resources usage and minimize waiting time for each process.

**Types of CPU Scheduling:**

**Preemptive Scheduling:**

In Preemptive Scheduling, the tasks are mostly assigned with their priorities. Sometimes it is important to run a task with a higher priority before another lower priority task, even if the lower priority task is still running. The lower priority task holds for some time and resumes when the higher priority task finishes its execution.

**Non-Preemptive Scheduling:**

In this type of scheduling method, the CPU has been allocated to a specific process. The process that keeps the CPU busy will release the CPU either by switching context or terminating. It is the only method that can be used for various hardware platforms. That’s because it doesn’t need special hardware (for example, a timer) like preemptive scheduling.

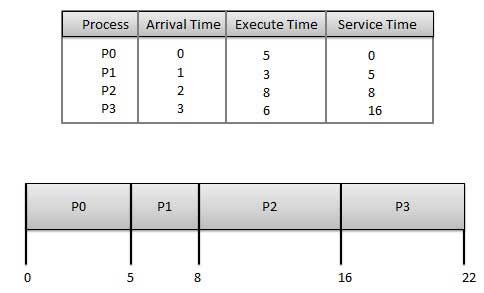
**Types of CPU scheduling Algorithm:**

There are mainly six types of process scheduling algorithms

**First Come First Serve (FCFS)**

First Come First Serve is the full form of FCFS. It is the easiest and most simple CPU scheduling algorithm. In this type of algorithm, the process which requests the CPU gets the CPU allocation first. This scheduling method can be managed with a FIFO queue.

As the process enters the ready queue, its PCB (Process Control Block) is linked with the tail of the queue. So, when CPU becomes free, it should be assigned to the process at the beginning of the queue.



e blanks that appear after the process arrives are I/O bursts determined by the left table.

**Wait time** of each process:

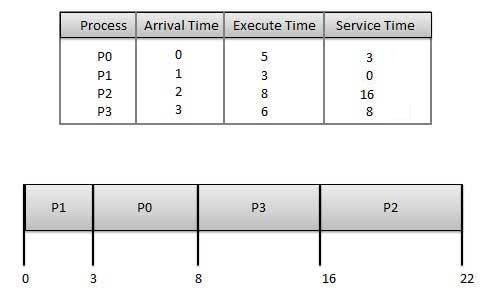
|  |  |
| --- | --- |
| **Process** | **Wait Time: Service Time - Arrival Time** |
| P0 | 0 - 0 = 0 |
| P1 | 5 - 1 = 4 |
| P2 | 8 - 2 = 6 |
| P3 | 16 - 3 = 13 |

Average Wait Time: (0+4+6+13) / 4 = 5.75

**Shortest-Job-First (SJF) Scheduling**

SJF is a full form of (Shortest job first) is a scheduling algorithm in which the process with the shortest execution time should be selected for execution next. This scheduling method can be preemptive or non-preemptive. It significantly reduces the average waiting time for other processes awaiting execution.

|  |  |  |  |
| --- | --- | --- | --- |
| **Process** | **Arrival Time** | **Execution Time** | **Service Time** |
| P0 | 0 | 5 | 0 |
| P1 | 1 | 3 | 5 |
| P2 | 2 | 8 | 14 |
| P3 | 3 | 6 | 8 |



**Waiting time** of each process:

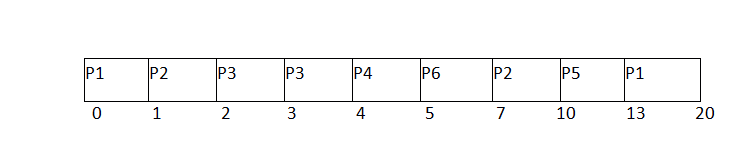
|  |  |
| --- | --- |
| **Process** | **Waiting Time** |
| P0 | 0 - 0 = 0 |
| P1 | 5 - 1 = 4 |
| P2 | 14 - 2 = 12 |
| P3 | 8 - 3 = 5 |

Average Wait Time: (0 + 4 + 12 + 5)/4 = 21 / 4 = 5.25

**Shortest Remaining Time**

The full form of SRT is Shortest remaining time. It is also known as SJF preemptive scheduling. In this method, the process will be allocated to the task, which is closest to its completion. This method prevents a newer ready state process from holding the completion of an older process.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Process ID** | **Arrival Time** | **Burst Time** | **Completion Time** | **Turn Around Time** | **Waiting Time** | **Response Time** |
| 1 | 0 | 8 | 20 | 20 | 12 | 0 |
| 2 | 1 | 4 | 10 | 9 | 5 | 1 |
| 3 | 2 | 2 | 4 | 2 | 0 | 2 |
| 4 | 3 | 1 | 5 | 2 | 1 | 4 |
| 5 | 4 | 3 | 13 | 9 | 6 | 10 |
| 6 | 5 | 2 | 7 | 2 | 0 | 5 |

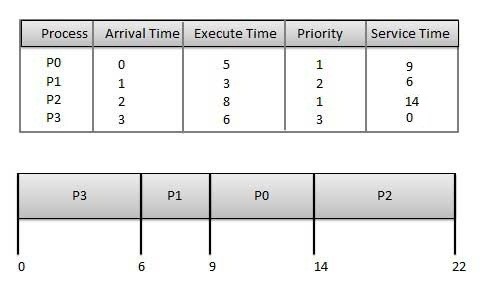
Avg Waiting Time = 24/6

**Priority Scheduling**

Priority scheduling is a method of scheduling processes based on priority. In this method, the scheduler selects the tasks to work as per the priority.

Priority scheduling also helps OS to involve priority assignments. The processes with higher priority should be carried out first, whereas jobs with equal priorities are carried out on a round-robin or FCFS basis. Priority can be decided based on memory requirements, time requirements, etc.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Process** | **Arrival Time** | **Execution Time** | **Priority** | **Service Time** |
| P0 | 0 | 5 | 1 | 0 |
| P1 | 1 | 3 | 2 | 11 |
| P2 | 2 | 8 | 1 | 14 |
| P3 | 3 | 6 | 3 | 5 |



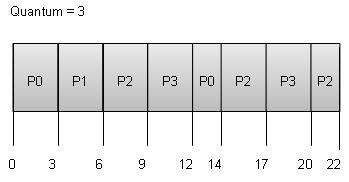
**Waiting time** of each process:

|  |  |
| --- | --- |
| **Process** | **Waiting Time** |
| P0 | 0 - 0 = 0 |
| P1 | 11 - 1 = 10 |
| P2 | 14 - 2 = 12 |
| P3 | 5 - 3 = 2 |

Average Wait Time: (0 + 10 + 12 + 2)/4 = 24 / 4 = 6

**Round Robin Scheduling**

Round robin is the oldest, simplest scheduling algorithm. The name of this algorithm comes from the round-robin principle, where each person gets an equal share of something in turn. It is mostly used for scheduling algorithms in multitasking. This algorithm method helps for starvation free execution of processes.



**Wait time** of each process:

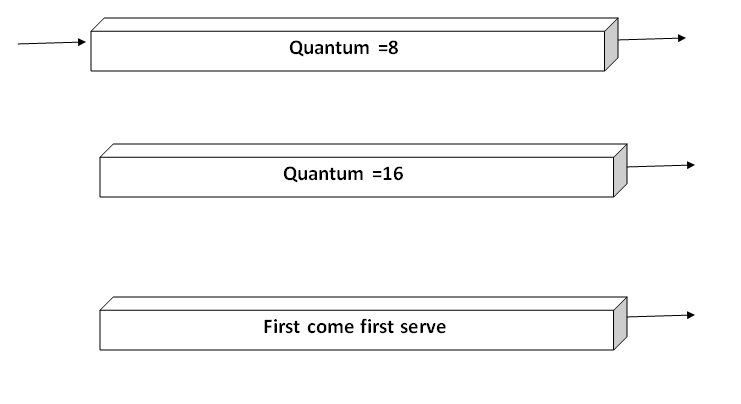
|  |  |
| --- | --- |
| **Process** | **Wait Time: Service Time - Arrival Time** |
| P0 | (0 - 0) + (12 - 3) = 9 |
| P1 | (3 - 1) = 2 |
| P2 | (6 - 2) + (14 - 9) + (20 - 17) = 12 |
| P3 | (9 - 3) + (17 - 12) = 11 |

Average Wait Time: (9+2+12+11) / 4 = 8.5

**Multilevel Queue Scheduling**

This algorithm separates the ready queue into various separate queues. In this method, processes are assigned to a queue based on a specific property of the process, like the process priority, size of the memory, etc.

However, this is not an independent scheduling OS algorithm as it needs to use other types of algorithms to schedule the jobs.



Table

Description automatically generated

Priority of queue 1 is greater than queue 2. queue 1 uses Round Robin (Time Quantum = 2) and queue 2 uses FCFS.

**Gantt chart**:

Application

Description automatically generated with low confidence

At starting both queues have process so process in queue 1 (P1, P2) runs first (because of higher priority) in the round robin fashion and completes after 7 units then process in queue 2 (P3) starts running (as there is no process in queue 1) but while it is running P4 comes in queue 1 and interrupts P3 and start running for 5 second and after its completion P3 takes the CPU and completes its execution.

1. **In multiprocessor system, cache coherence is the uniformity of shared resource data that ends up stored in multiple local caches. Serval protocols are commonly used as the selections of the architecture designs. Please explain how MESI works with the state diagram**

The letters in the acronym MESI represent four exclusive states that a cache line can be marked with (encoded using two additional bits):

**Modified (M)**

The cache line is present only in the current cache and is dirty - it has been modified (M state) from the value in main memory. The cache is required to write the data back to main memory at some time in the future, before permitting any other read of the (no longer valid) main memory state. The write-back changes the line to the Shared state(S).

**Exclusive (E)**

The cache line is present only in the current cache but is clean - it matches main memory. It may be changed to the Shared state at any time, in response to a read request. Alternatively, it may be changed to the Modified state when writing to it.

**Shared (S)**

Indicates that this cache line may be stored in other caches of the machine and is clean - it matches the main memory. The line may be discarded (changed to the Invalid state) at any time.

**Invalid (I)**

Indicates that this cache line is invalid (unused).

Diagram, venn diagram

Description automatically generated

**Operation:**

The state of the FSM transitions from one state to another based on 2 stimuli. The first stimulus is the processor specific Read and Write request. For example: A processor P1 has a Block X in its Cache, and there is a request from the processor to read or write from that block. The second stimulus comes from another processor, which doesn't have the Cache block or the updated data in its Cache, through the bus connecting the processors. The bus requests are monitored with the help of Snoopers,[4] which monitor all the bus transactions.

Following are the different type of Processor requests and Bus side requests:

Processor Requests to Cache include the following operations:

PrRd: The processor requests to read a Cache block.

PrWr: The processor requests to write a Cache block

Bus side requests are the following:

BusRd: Snooped request that indicates there is a read request to a Cache block requested by another processor

BusRdX: Snooped request that indicates there is a write request to a Cache block requested by another processor that doesn't already have the block.

BusUpgr: Snooped request that indicates that there is a write request to a Cache block requested by another processor, but that processor already has that Cache block residing in its own Cache.

Flush: Snooped request that indicates that an entire cache block is written back to the main memory by another processor.

FlushOpt: Snooped request that indicates that an entire cache block is posted on the bus to supply it to another processor (Cache to Cache transfers).

1. **Explain why interrupt and dispatch latency times must be bounded in a hard real-time system.**

Scheduling behavior for real-time applications is the most significant element of real-time scheduling class. The standard time-sharing scheduling class is not suitable for real-time applications because this scheduling class treats every process equally and has a limited notion of priority. Real-time applications need a scheduling class in which process priorities are taken as absolute and are changed only by explicit application operations. The term dispatch latency describes the amount of time it takes for a system to respond to a request for a process to begin operation. With a scheduler written specifically to honor application priorities, real-time applications can be developed with a bounded dispatch latency.

**Application Response Time:**

Timeline

Description automatically generated

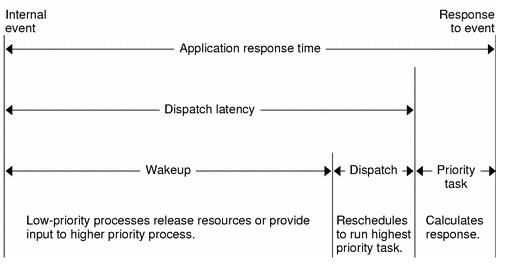
The overall application response time is composed of the interrupt response time, the dispatch latency, and the time it takes the application itself to determine its response. The interrupt response time for an application includes both the interrupt latency of the system and the device driver's own interrupt processing time. The interrupt latency is determined by the longest interval that the system must run with interrupts disabled; this is minimized in SunOS 5.0 through 5.8 using synchronization primitives that do not commonly require a raised processor interrupt level.

During interrupt processing, the driver's interrupt routine wakes up the high priority process and returns when finished. The system detects that a process with higher priority than the interrupted process in now dispatchable and arranges to dispatch that process. The time to switch context from a lower priority process to a higher priority process is included in the dispatch latency time.

Figure below illustrates the internal dispatch latency/application response time of a system, defined in terms of the amount of time it takes for a system to respond to an internal event. The dispatch latency of an internal event represents the amount of time required for one process to wake up another higher priority process, and for the system to dispatch the higher priority process.

The application response time is the amount of time it takes for a driver to wake up a higher priority process, have a low priority process release resource, reschedule the higher priority task, calculate the response, and dispatch the task.

**Internal Dispatch Latency:**

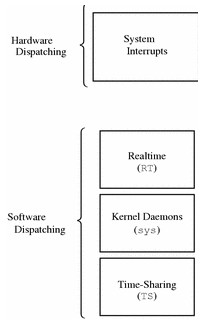


**Scheduling Classes**

The SunOS 5.0 through 5.8 kernel dispatches processes by priority. The scheduler or dispatcher supports the concept of scheduling classes. Classes are defined as Real-time(RT), System(sys), and Time Sharing(TS).Each class has an unique scheduling policy for dispatching processes within its class.

The kernel dispatches the highest priority processes first. By default, real-time processes have precedence over **sys** and **TS** processes, but administrators can configure systems so that **TS** and **RT** processes have overlapping priorities.

**Dispatch Priorities for Scheduling Classes**

****

Consider that a program is in execution process, Do the following tasks:

First, save the currently executing instruction, and then determine the type of interrupt, next save the current process state, and then invoke the appropriate interrupt service routine. Dispatch latency is the cost associated with stopping one process and starting another. Both interrupt and dispatch latency need to be minimized to ensure that real-time tasks receive immediate attention. Furthermore, sometimes interrupts are disabled when kernel data structures are being modified, so the interrupt does not get serviced immediately. For hard real-time systems, the time-period for which interrupts are disabled must be bounded to guarantee the desired quality of service.