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

**Answer: CPU Scheduling** is a process of determining which process will own CPU for execution while another process is on hold. The main task of CPU scheduling is to make sure that whenever the CPU remains idle, the OS at least select one of the processes available in the ready queue for execution. The selection process will be carried out by the CPU scheduler. It selects one of the processes in memory that are ready for execution.

CPU scheduling decisions may take place when a process:   
1. Switches from running to waiting state   
2. Switches from running to ready state   
3. Switches from waiting to ready   
4. Terminates

There are two kinds of Scheduling methods:

### 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.

There are mainly six types of process scheduling algorithms

1. First Come First Serve (FCFS)
2. Shortest-Job-First (SJF) Scheduling
3. Shortest Remaining Time
4. Priority Scheduling
5. Round Robin Scheduling
6. Multilevel Queue Scheduling

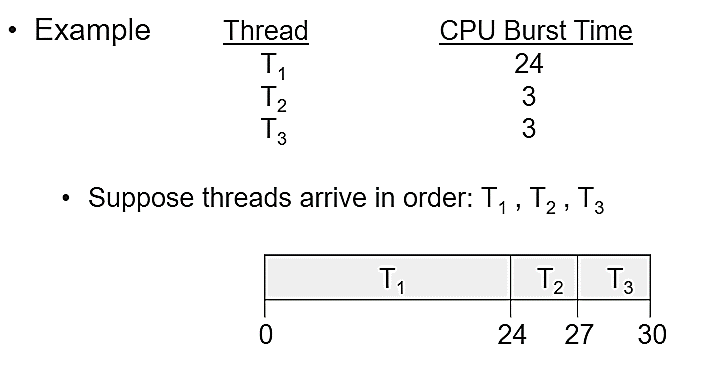
## First Come First Serve

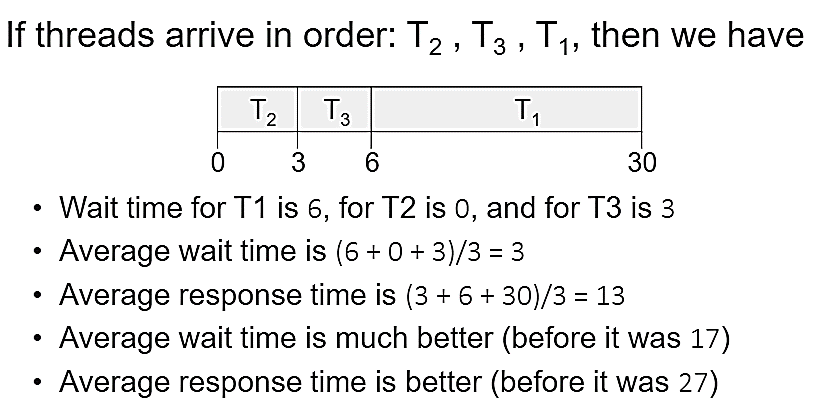
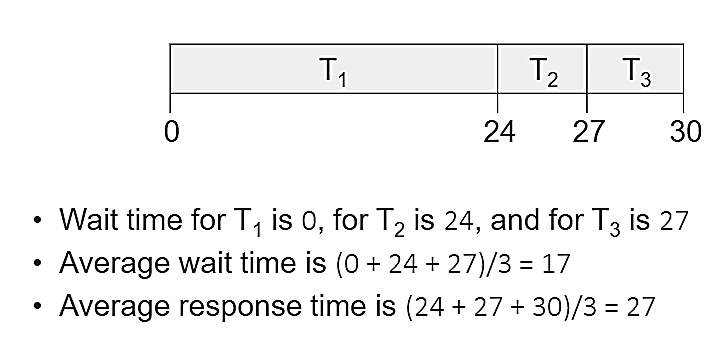
First Come First Serve is the full form of FCFS. It is the easiest and most simple CPU scheduling algorithm. In this type of algorithm, the process which requests the CPU gets the CPU allocation first. This scheduling method can be managed with a FIFO queue.

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.

### Characteristics of FCFS method:

* It offers non-preemptive and pre-emptive scheduling algorithm.
* Jobs are always executed on a first-come, first-serve basis
* It is easy to implement and use.
* However, this method is poor in performance, and the general wait time is quite high.





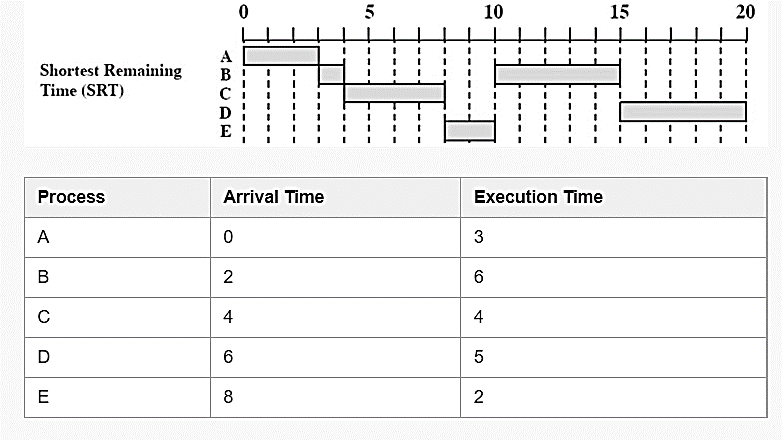
## Shortest Remaining Time

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.

### Characteristics of SRT scheduling method:

* This method is mostly applied in batch environments where short jobs are required to be given preference.
* This is not an ideal method to implement it in a shared system where the required CPU time is unknown.
* Associate with each process as the length of its next CPU burst. So that operating system uses these lengths, which helps to schedule the process with the shortest possible time.

Example :

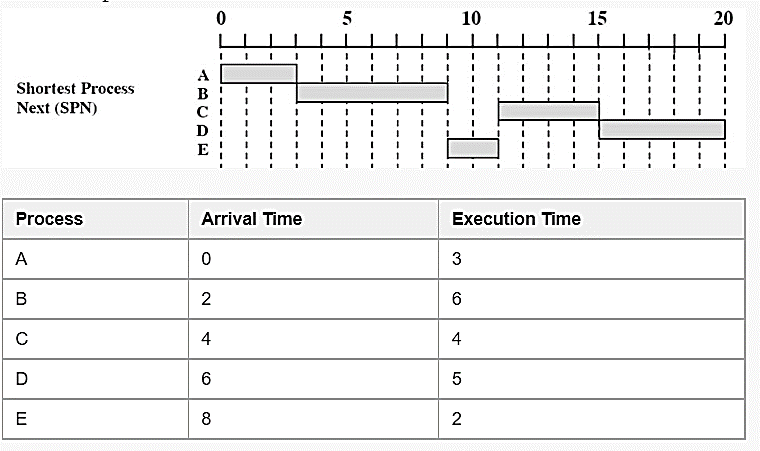


## Shortest Job First

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.

### Characteristics of SJF Scheduling

* It is associated with each job as a unit of time to complete.
* In this method, when the CPU is available, the next process or job with the shortest completion time will be executed first.
* It is Implemented with non-preemptive policy.
* This algorithm method is useful for batch-type processing, where waiting for jobs to complete is not critical.
* It improves job output by offering shorter jobs, which should be executed first, which mostly have a shorter turnaround time.

Example:

Priority Based 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.

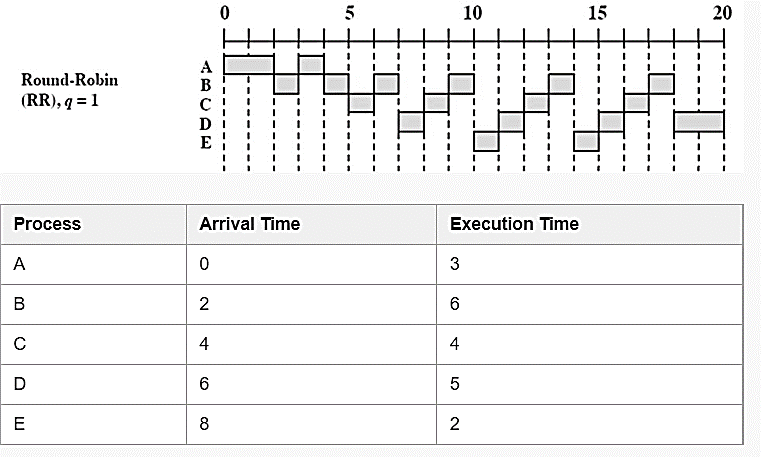
## Round-Robin Scheduling

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.

### Characteristics of Round-Robin Scheduling

* Round robin is a hybrid model which is clock-driven
* Time slice should be minimum, which is assigned for a specific task to be processed. However, it may vary for different processes.
* It is a real time system which responds to the event within a specific time limit.
  + Each process gets a small unit of CPU time (time quantum q), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue. If there are n processes in the ready queue and the time quantum is q, then each process gets 1/n of the CPU time in chunks of at most q time units at once. No process waits more than (n-1)q time units.   
    - Timer interrupts every quantum to schedule next process   
    - Performance   
    - q large ⇒ FIFO   
    - q small ⇒ q must be large with respect to context switch, otherwise overhead is too high

Example:



## Multiple-Level Queues Scheduling

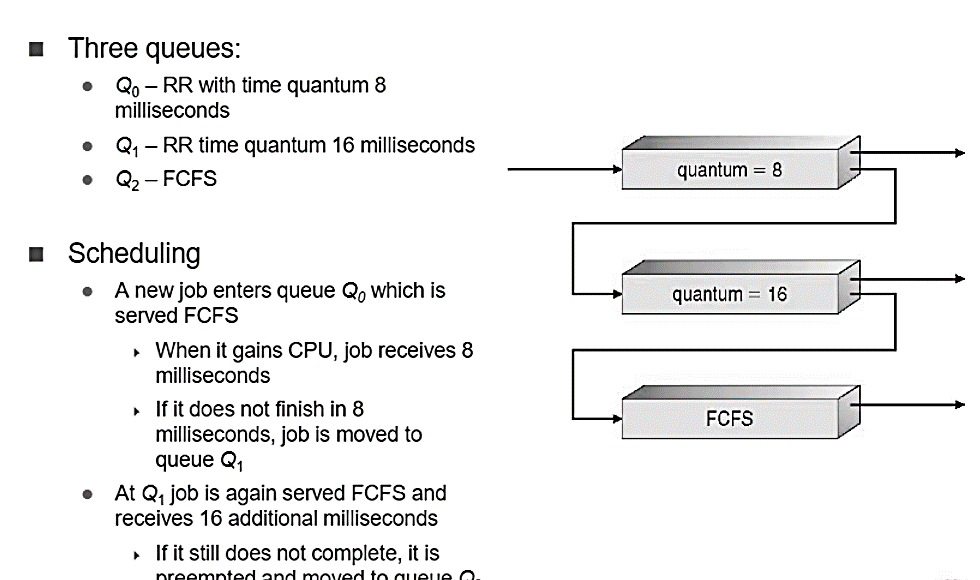
This algorithm separates the ready queue into various separate queues. In this method, processes are assigned to a queue based on a specific property of the process, like the process priority, size of the memory, etc.

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

### Characteristic of Multiple-Level Queues Scheduling:

* Multiple queues should be maintained for processes with some characteristics.
* Every queue may have its separate scheduling algorithms.
* Priorities are given for each queue.

Example:



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

Answer:

1. As described earlier, in MSI, a cache block can be in one of three states -

• Invalid (uncached): not in the cache (not valid in any cache)

• Shared/clean: cached in one or more processors and memory is up-to-date

• Modified/dirty/exclusive: one processor (owner) has data; memory out-of-date

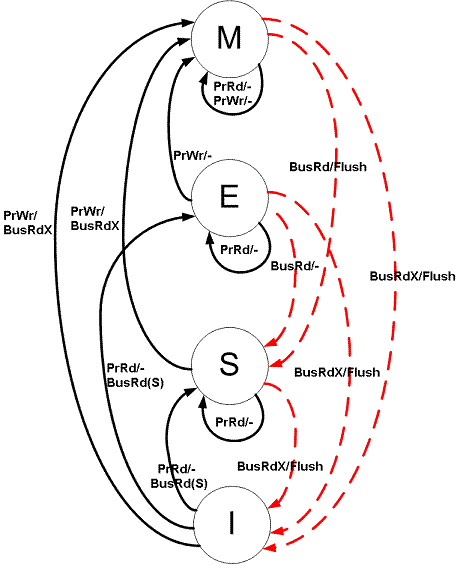
• The MESI protocol regroup the Shared and Modified states into three states:

• Invalid (uncached): same as in MSI

• Shared: cached in more than one prosessors and memory is up-to-date

• Exclusive: one processor (owner) has data and it is clean (clean but not shared)

• Modified: one processor (owner) has data, but it is dirty



3. 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](https://en.wikipedia.org/wiki/Bus_snooping),[[4]](https://en.wikipedia.org/wiki/MESI_protocol#cite_note-4) which monitor all the bus transactions.

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

a).Processor Requests to Cache include the following operations:

1. PrRd: The processor requests to read a Cache block.
2. PrWr: The processor requests to write a Cache block

b).Bus side requests are the following:

1. BusRd: Snooped request that indicates there is a read request to a Cache block requested by another processor
2. 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.
3. 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.
4. Flush: Snooped request that indicates that an entire cache block is written back to the main memory by another processor.
5. FlushOpt: Snooped request that indicates that an entire cache block is posted on the bus in order to supply it to another processor(Cache to Cache transfers).

**Snooping Operation**: In a snooping system, all caches on the bus monitor (or snoop) all the bus transactions. Every cache has a copy of the sharing status of every block of physical memory it has stored. The state of the block is changed according to the State Diagram of the protocol used. (Refer image above for MESI state diagram). The bus has snoopers on both sides:

1. Snooper towards the Processor/Cache side.
2. The snooping function on the memory side is done by the Memory controller.

5.If MESI is implemented using a directory, then the information kept for each block in the directory is the same as the three- state protocol:

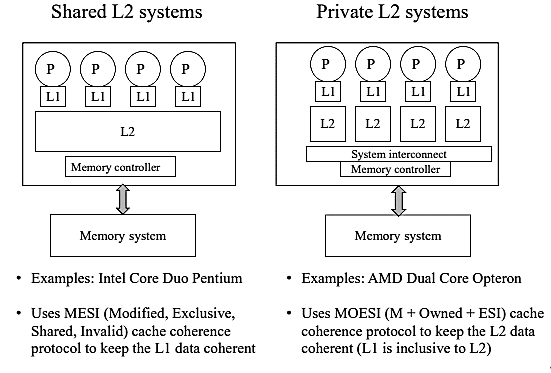
• Shared in MESI = shared/clean but more than one sharer

• Exclusive in MESI = shared/clean but only one sharer

• Modified in MESI = Exclusive/Modified/dirty

• However, at each cached copy, a distinction is made between shared, exclusive and modified (rather than only shared and modified)

**Cache organization in multicore systems**



6.On a read miss (local block is invalid), load the block and change its state to

• “exclusive” if it was uncached in memory

• “shared” if it was already shared, modified or exclusive - if it was modified, the owner will send you a clean copy - if was modified or exclusive, the previous owner will change the state of the block to “shared” in its cache.

7. On a write miss: same as read miss, except set the state to “modified” ¬ copies in other caches (if any) are invalidated

• On a write hit to a “modified” block, do nothing

• On a write hit to an “exclusive” block change the block to “modified” ¬ no need for invalidation

• On a write hit to a “shared” block change the block to “modified” and invalidate the other cached copies.

8.When a modified block is evicted, write it back. this is the main advantage of MESI over MSI

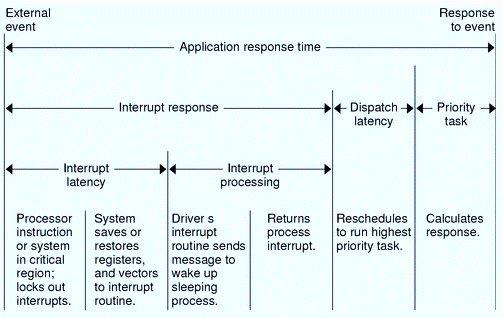
9. In snooping bus implementations of MESI, on a read miss, we need to set its state correctly to “shared” (if the block is in some other cache(s)) or “exclusive” (if not).

10.To take full advantage of MESI, should know when a block is to be changed from shared to exclusive.

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

Answer: Scheduling behaviour 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 needs 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.

**Fig.1 Application Response Time**

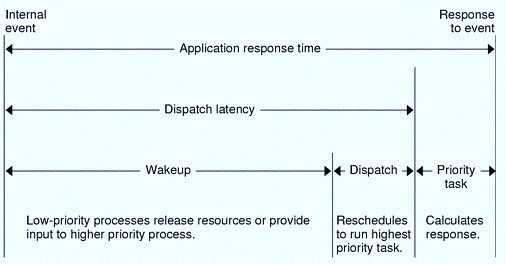


  The overall application response time is composed of the interrupt response time, the dispatch latency, and the time it takes the application itself to determine its response.The interrupt response time for an application includes both the interrupt latency of the system and the device driver's own interrupt 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.

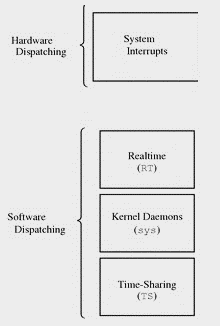
          During interrupt processing, the driver's interrupt routine wakes up the high priority process and returns when finished. The system detects that a process with higher priority than the interrupted process in now dispatchable and arranges to dispatch that process. The time to switch context from a lower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

Fig.2 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 resources, reschedule the higher priority task, calculate the response, and dispatch the task.

**Fig.2 Internal Dispatch Latency:**



     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 dispatch 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. First of all 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 needs to be minimized in order 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 in order to guarantee the desired quality of service.



**Fig.3** **Dispatch Priorities for Scheduling Classes**

First of all 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 needs to be minimized in order 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 in order to guarantee the desired quality of service.