Journal Of Harmonized Research (JOHR)

Journal Of Harmonized Research in Engineering
1(2), 2013, 45-53

ISSN 2347 – 7393

Original Research Article

REDUCING TIME: SCHEDULING JOB

Nisha Yadav, Nikita Chhillar, Neha jaiswal

Department of Computer Science and Engineering,
Dronacharya College of Engineering, Khentawas, Farukhnagar,
Gurgaon, India

Abstract:
Scheduling is the method by which threads, processes or data flows are given access to system resources like processor or other resources required. This is basically done to load balance a system efficiently or accomplish a target quality of service. Scheduling algorithms are required by most contemporary systems to perform multitasking i.e. execution of more than one process at a time and multiplexing i.e. transmission of multiple flows simultaneously.
The scheduling is concerned chiefly with:

- Throughput - The total number of processes that complete their execution per time unit.
- Latency, specifically:
  - Turnaround time - total time between submission of a process and its completion.
  - Response time - amount of time it takes from when a request was submitted until the first response is produced.
- Fairness / Waiting Time - Equal CPU time to each process (or more generally appropriate times according to each process' priority). It is the time for which the process remains in the ready queue.

Keywords: Scheduling, turnaround time, response time, average waiting time, scheduler

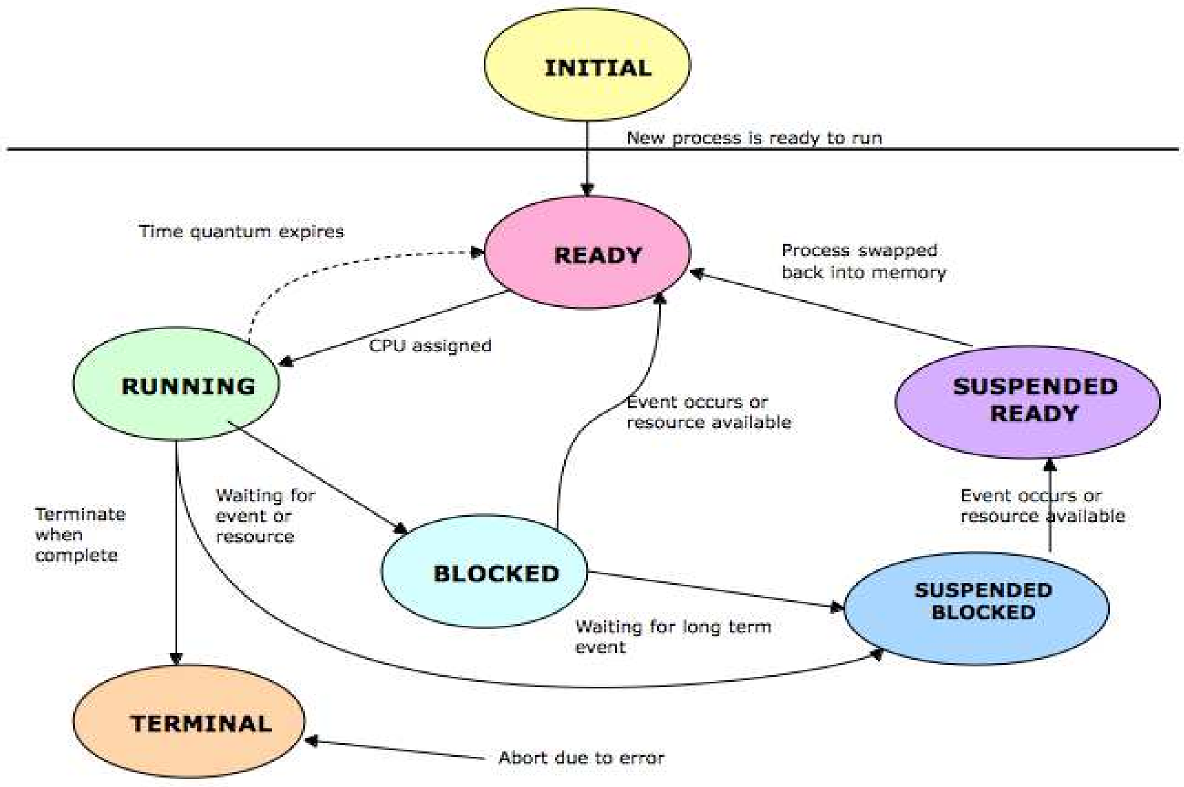
Introduction:

"Job scheduling" refers to a batch system that supervises and scrutinizes the background data and applications that are necessary for batch jobs to occur. **Job scheduler** is a computer application for managing unattended behind-the-scenes program execution. It is also known as **Distributed Resource Management System** and **Distributed Resource Manager**. Job Scheduling has been one of the major components of IT infrastructure since the early mainframe systems.

For Correspondence:
nikitachhillar@yahoo.com
Received on: September 2013
Accepted after revision: September 2013
Downloaded from: www.johronline.com

❖ **Long-term scheduling:**

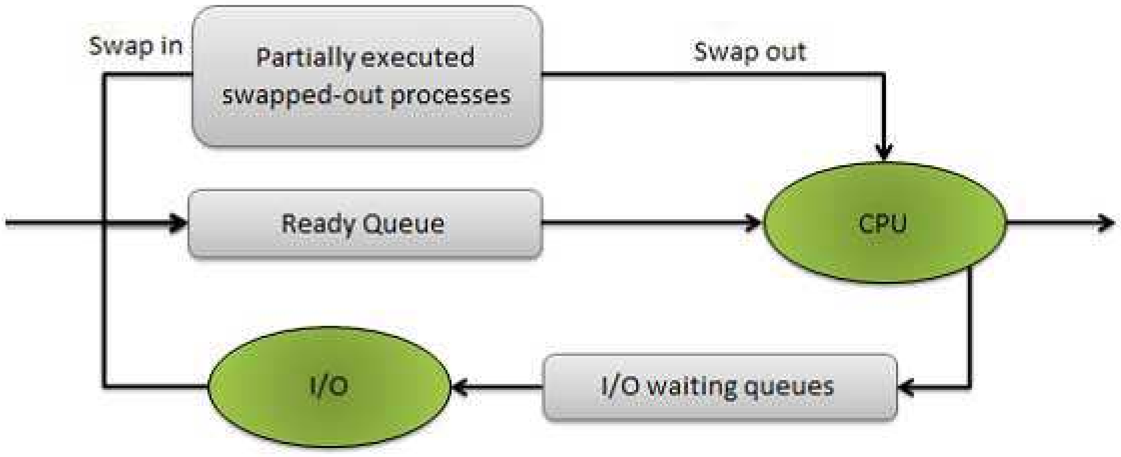
The long-term, or admission scheduler, decides which jobs or processes are to be acknowledged to enter the ready queue i.e. in the Main Memory which means, when an attempt is made to execute a program, its admission to the set of presently executing processes is either permitted or delayed by the long-term scheduler . Thus, this scheduler orders what processes are to run on a system, and the degree of concurrency to be sustained at any one time i.e. whether a high or low number of processes are to be executed concurrently, and how the split between input output intensive and CPU intensive processes is to be handled. So long term scheduler is responsible for controlling the degree of multiprogramming. In modern operating systems, this is used to make certain that real time processes get sufficient CPU time to finish their assignments. Without proper real time scheduling, modern Graphical user interfaces would seem slow. The long term queue exists in the Hard Disk or the "Virtual Memory". Long-term scheduling is also important in large-scale systems such as batch processing systems, computer clusters, supercomputers and render farms. In these cases, special purpose job scheduler software is naturally used to support these functions, in addition to any underlying admission scheduling support in the operating system.


INITIAL
New process is ready to run
Time quantum expires
READY
Process swapped back into memory
CPU assigned
RUNNING
SUSPENDED READY
Event occurs or resource available
Event occurs or resource available
Terminate when complete
Waiting for event or resource
BLOCKED
SUSPENDED BLOCKED
Waiting for long term event
TERMINAL
Abort due to error

❖ **Medium term scheduling:**

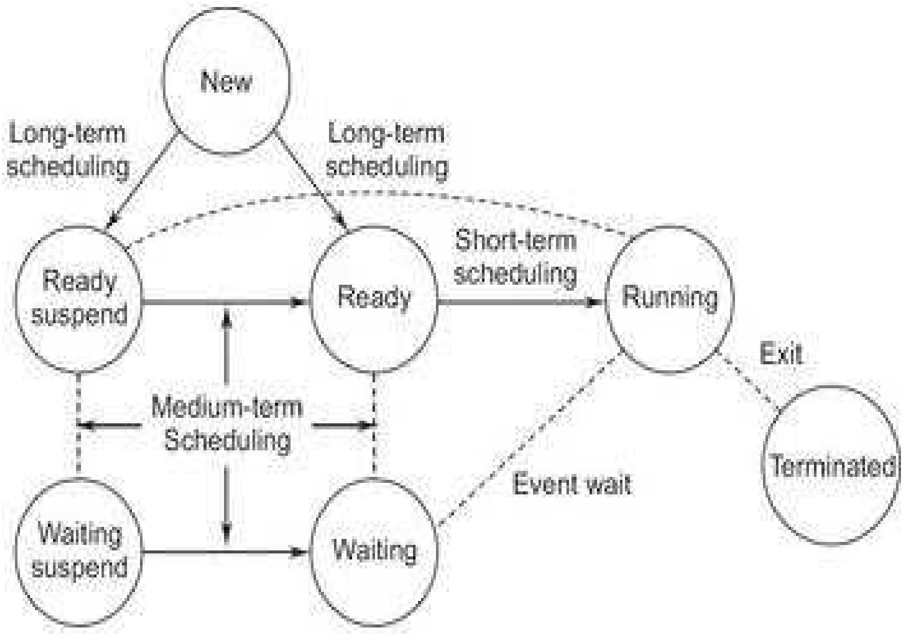
Scheduler temporarily eliminates processes from main memory and situates them on secondary memory (such as a disk drive) or vice versa. This is commonly referred to as "swapping out" or "swapping in" (also incorrectly as "paging out" or "paging in"). The medium-term scheduler possibly will decide to swap out a process which has not been active for a little time, or a process which has a low priority, or a process which is page faulting regularly, or a process which is taking up a large amount of memory in order to clear up main memory for other processes, swapping the process back in later when more memory is available, or when the process has been unblocked and is no longer waiting for a resource.

In many systems today which support mapping virtual address space to secondary storage other than the swap file, the medium-term scheduler may essentially perform the role of the long-term scheduler, by treating binaries as "swapped out processes" upon their execution. In this way, when a segment of the binary is required it can be swapped in when insisted, or "lazy loaded".



❖ **Short-term scheduling:**
The short-term scheduler also called the CPU scheduler is responsible for deciding which of the ready, in-memory processes are to be executed or allocated a processor after a clock interrupt, an I/O interrupt, an operating system call or another form of signal. Thus the short-term scheduler makes scheduling decisions much more frequently than the long-term or mid-term schedulers - a scheduling decision will at a minimum have to be made after every time slice, and these are very short. This scheduler can be preemptive, implying that it is capable of forcibly removing processes from a CPU when it decides to allocate that CPU to another process, or non-preemptive (also known as "voluntary" or "co-operative"), in which case the scheduler is unable to "force" processes off the CPU.
A preemptive scheduler relies upon a programmable interval timer which invokes an interrupt handler that runs in kernel mode and implements the scheduling function.

Comparison between Schedulers:



S.N.	Long Term Scheduler	Short Term Scheduler	Medium Term Scheduler
1	It is a job scheduler	It is a CPU scheduler	It is a process swapping scheduler.
2	Speed is least as compared to all.	Speed is fastest among the all.	Speed lies in between short and long term scheduler.
3	It controls the degree of multiprogramming	It provides less control over degree of multiprogramming	It decreases the degree of multiprogramming.
4	It is almost absent or minimal in time sharing system	It is also minimal in time sharing system	It is an element of Time sharing systems.
5	It selects processes from pool and loads them into memory for execution	It selects those processes which are ready to execute	It can re-introduce the process into memory and execution can be continued.

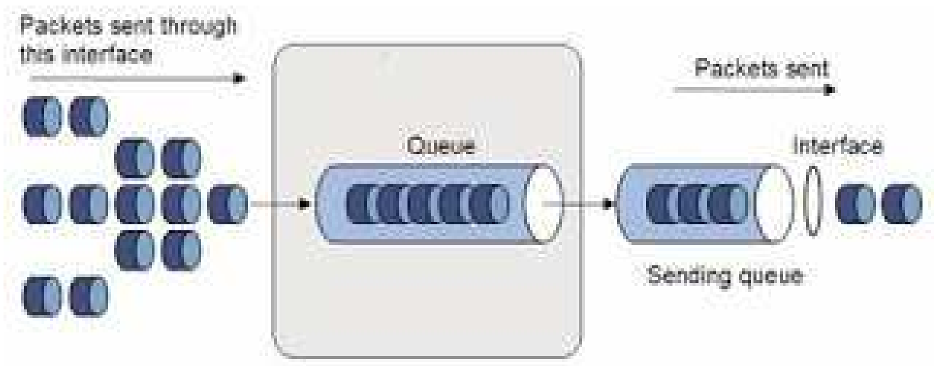
Scheduling disciplines:
Scheduling disciplines are algorithms used for allocating resources among parties which simultaneously and asynchronously demand them. Scheduling disciplines are used in routers to handle packet traffic as well as in operating systems to share processor time among both threads and processes, disk drives (I/O scheduling), printers (print spooler), most embedded systems, etc. The main purposes of scheduling algorithms are to reduce the resource starvation and to ensure a fair distribution amongst the parties consuming the resources. Scheduling deals with the problem of deciding which of the outstanding requests is to be allocated resources. There are many different scheduling algorithms. In this section, we introduce several of them.
In packet-switched computer networks and other statistical multiplexing, the notion of a **scheduling algorithm** is used as an alternative to first-come first-served queuing of data packets.
The easiest and best-effort scheduling algorithms are round-robin, fair queuing which is a max-min fair scheduling algorithm, proportionally fair scheduling and maximum throughput. If differentiated or promised quality of service is offered, as opposed to best-effort communication, weighted fair queuing may be utilized.
In advanced packet radio wireless networks such as HSDPA (High-Speed Downlink Packet Access) 3.5G cellular system, **channel-dependent scheduling** may be used to take advantage of channel state information. If the channel conditions are favorable, the throughput and system spectral efficiency may be increased. In even more advanced systems such as LTE, the scheduling is combined by channel-dependent packet-by-packet dynamic channel allocation, or by assigning OFDMA multi-carriers or other frequency-domain equalization components to the users that best can utilize them.

➢ **First in first out:**

Also known as ***First Come, First Served*** (FCFS), is the simplest scheduling algorithm, FIFO simply queues processes in the order that they arrive in the ready queue.

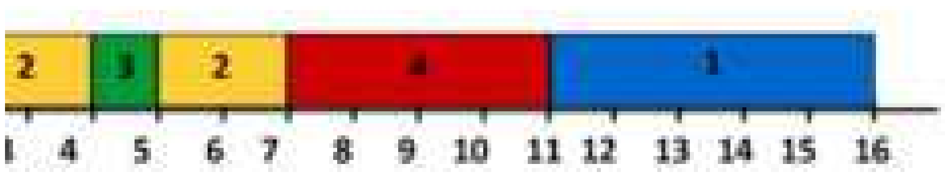
- Since context switches only occur upon process termination, and no reorganization of the process queue is required, scheduling overhead is minimal.
- Throughput can be low, since long processes can hold the CPU
- Turnaround time, waiting time and response time can be high for the same reasons above

- No prioritization occurs, thus this system has trouble meeting process deadlines.
- The lack of prioritization means that as long as every process eventually completes, there is no starvation. In an environment where some processes might not complete, there can be starvation.
- It is based on Queuing



➢ **Shortest remaining time:**
Similar to *Shortest **Job First*** (SJF). With this strategy the scheduler arranges processes with the least estimated processing time remaining to be next in the queue. This requires advanced knowledge or estimations about the time required for a process to complete.

- If a shorter process arrives during another process' execution, the currently running process may be interrupted (known as preemption), dividing that process into two separate computing blocks. This creates excess overhead through additional context switching. The scheduler must also place each incoming process into a specific place in the queue, creating additional overhead.
- This algorithm is designed for maximum throughput in most scenarios.
- Waiting time and response time increase as the process's computational requirements increase. Since turnaround time is based on waiting time plus processing time, longer processes are significantly affected by this. Overall waiting time is smaller than FIFO, however since no process has to wait for the termination of the longest process.
- No particular attention is given to deadlines, the programmer can only attempt to make processes with deadlines as short as possible.
- Starvation is possible, especially in a busy system with many small processes being run.
- This policy is no more in use.
- To use this policy we should have at least two processes of different priority

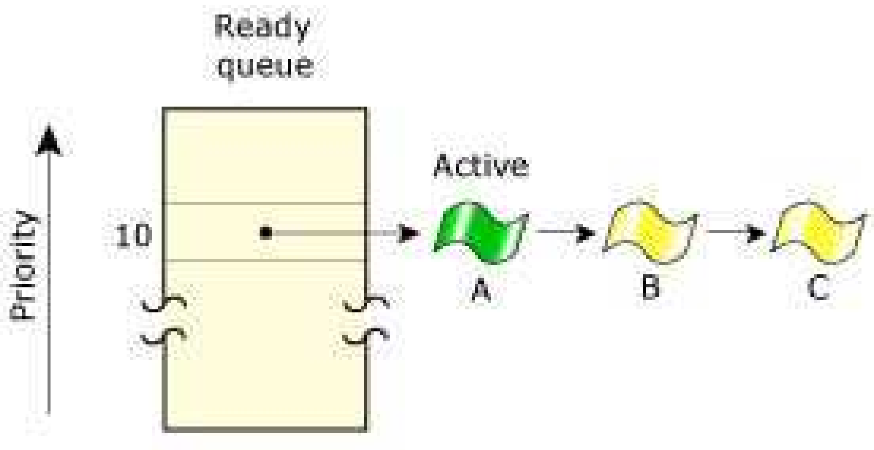


➢ **Fixed priority pre-emptive scheduling:**
The OS assigns a fixed priority rank to every process, and the scheduler arranges the processes in the ready queue in order of their priority. Lower priority processes get interrupted by incoming higher priority processes.

- Overhead is not minimal, nor is it significant.
- FPPS has no particular advantage in terms of throughput over FIFO scheduling.
- If the number of rankings is limited it can be characterized as a collection of FIFO queues, one for each priority ranking.

Processes in lower-priority queues are selected only when all of the higher-priority queues are empty.

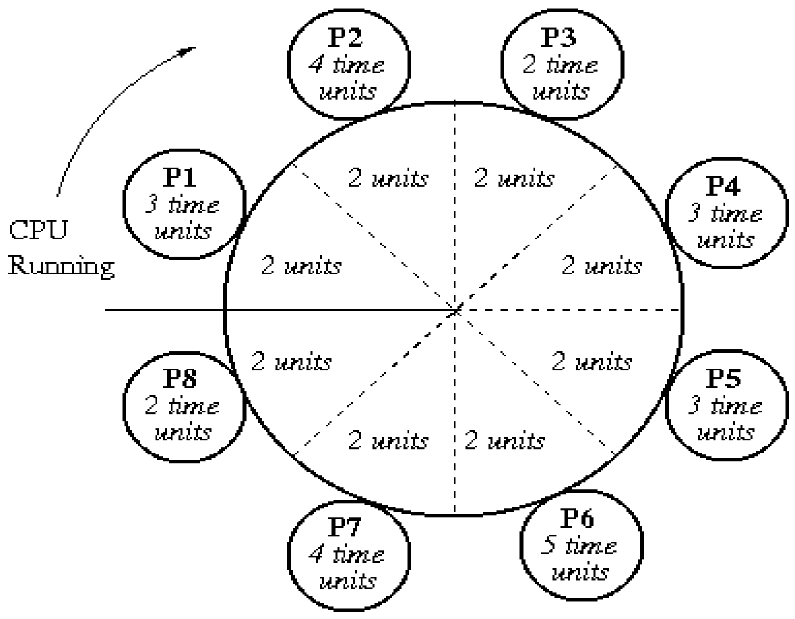
- Waiting time and response time depend on the priority of the process. Higher priority processes have smaller waiting and response times.
- Deadlines can be met by giving processes with deadlines a higher priority.
- Starvation of lower priority processes is possible with large amounts of high priority processes.



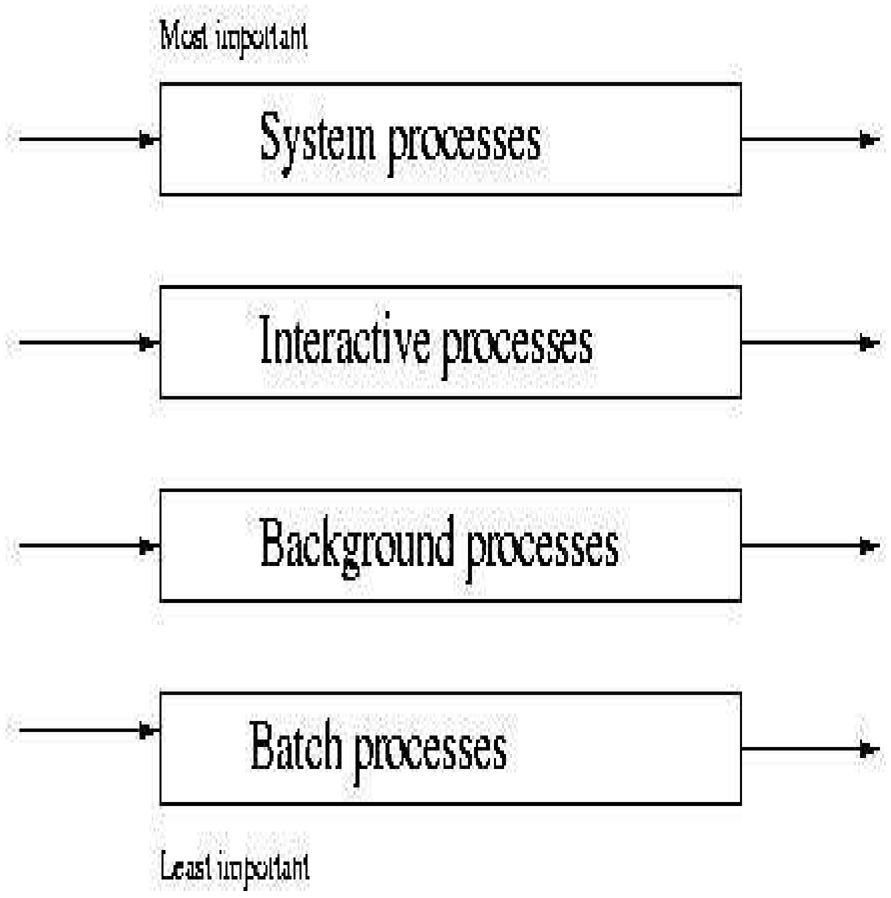
➢ **Round-robin scheduling:**

The scheduler assigns a fixed time unit per process, and cycles through them.

- RR scheduling involves extensive overhead, especially with a small time unit.
- Balanced throughput between FCFS and SJF, shorter jobs are completed faster than in FCFS and longer processes are completed faster than in SJF.
- Poor average response time, waiting time is dependent on number of processes, and not average process length.
- Because of high waiting times, deadlines are rarely met in a pure RR system.
- Starvation can never occur, since no priority is given. Order of time unit allocation is based upon process arrival time, similar to FCFS.

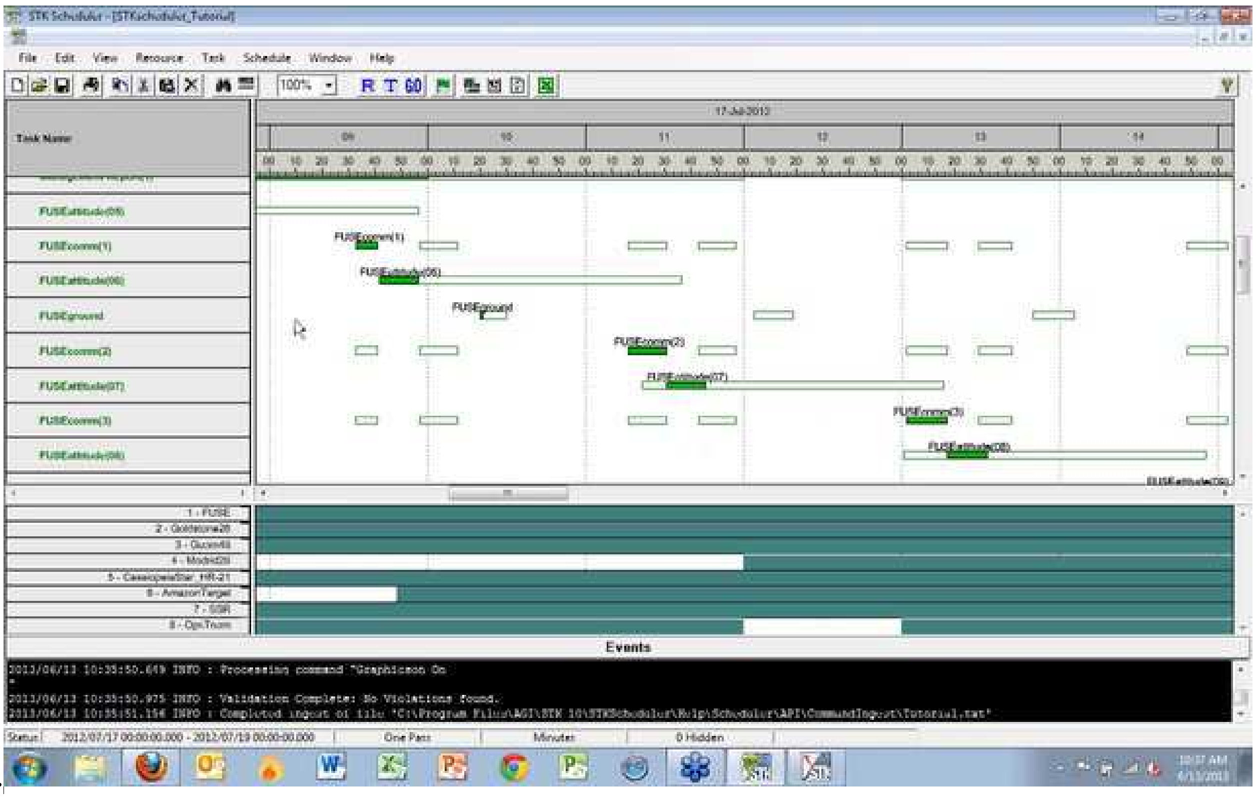


➢ **Multilevel queue scheduling:**
This is used for situations in which processes are easily divided into different groups. For example, a common division is made between foreground (interactive) processes and background (batch) processes. These two types of processes have different response-time requirements and so may have different scheduling needs. It is very useful for shared memory problems.
MQS is similar to PRI, except that the jobs arrive sorted by their priority. For example, all system jobs may have a higher priority than interactive jobs, which enjoy a higher priority than batch jobs. Jobs of different priorities are placed into different queues.
In some implementations, jobs in all higher priority queues must be executed before jobs in any lower priority queue. This absolute approach can lead to starvation in the same way as its simplier cousin, PRI. In some preemptive implementations, a lower-priority process will be returned to its ready queue, if a higher-priority process arrives.
Another approach is to time-slice among the queues. Higher priority queues can be given longer or more frequent time slices. This approach prevents absolute starvation.



➢ **Manual scheduling:**
A very common method in embedded systems is to manually schedule jobs. This can for example be done in a time-multiplexed fashion. Sometimes the kernel is divided in three or more parts: Manual scheduling, preemptive and interrupt level. Exact methods for scheduling jobs are often proprietary.

- No resource starvation problems.
- Very high predictability; allows implementation of hard real-time systems.
- Almost no overhead.
- May not be optimal for all applications.
- Effectiveness is completely dependent on the implementation.



```
2013/06/13 10:35:50.649 INFO : Processing command "GraphicsOn On
"
2013/06/13 10:35:50.975 INFO : Validation Complete: No Violations found.
2013/06/13 10:35:51.156 INFO : Completed ingest of file 'C:\Program Files\AGI\STK 10\STKScheduler\Help\Scheduler\API\CommandIngest\Tutorial.txt'
```

Status: 2012/07/17 00:00:00.000 - 2012/07/19 00:00:00.000 | One Pass | Minutes | 0 Hidden

Uses:
In real-time environments, such as embedded systems for automatic control in industry (for example robotics), the scheduler also must ensure that processes can meet deadlines; this is crucial for keeping the system stable. Scheduled tasks are sent to mobile devices and managed through an administrative back end.

Conclusion:
In practice, these objectives often conflict like throughput versus latency, thus a scheduler will implement a suitable negotiation. Preference is always given to any one of the above stated concerns depending upon the user's needs and purposes. While designing an operating system, a programmer must consider which scheduling algorithm will achieve the target in best way for the use which the system is going to see. There is no collective or we can say universal “best” scheduling algorithm, and many operating systems use extended or combinations of the scheduling algorithms stated above. For example, Windows NT/XP/Vista uses a multilevel feedback queue, a combination of fixed priority preemptive scheduling, round-robin, and first in first out. In this system, threads can dynamically increase or decrease in priority depending on if it has been serviced already, or if it has been waiting generally. Every priority stage is characterized by its own queue, with round-robin scheduling amongst the high priority threads and FIFO among the lower ones. In this common sense, response time is short for most threads, and short but critical system threads get completed very quickly. Since threads can only use one time unit of the round robin in the highest priority queue, starvation can be a problem for longer high priority threads

References:

1. Effect of Job Size Characteristics on Job Scheduling Performance
2. Brief discussion of Job Scheduling algorithms
3. Sriram Krishnan. "A Tale of Two Schedulers Windows NT and Windows CE"
4. Jump up to: [a] [b] "Technical Note TN2028 - Threading Architectures"
5. "Mach Scheduling and Thread Interfaces"
6. Molnár, Ingo (2007-04-13). "[patch] Modular Scheduler Core and Completely Fair Scheduler [CFS]"

7. Efficient and Scalable Multiprocessor Fair Scheduling Using Distributed Weighted Round-Robin

8. Błażewicz, Jacek; Ecker, K.H.; Pesch, E.; Schmidt, G.; Weglarz, J. (2001). Scheduling computer and manufacturing processes (2 ed.). Berlin [u.a.]: Springer. ISBN 3-540-41931-4.

9. Stallings, William (2004). Operating Systems Internals and Design Principles (fifth international edition). Prentice Hall