THE UNIVERSITY OF NEWCASTLE AUSTRALIA

FACULTY OF ENGINEERING AND BUILT ENVIRONMENT

www.newcastle.edu.au

OPERATING SYSTEMS

Week 3

Much of the material on these slides comes from the recommended textbook by William Stallings

Detailed content

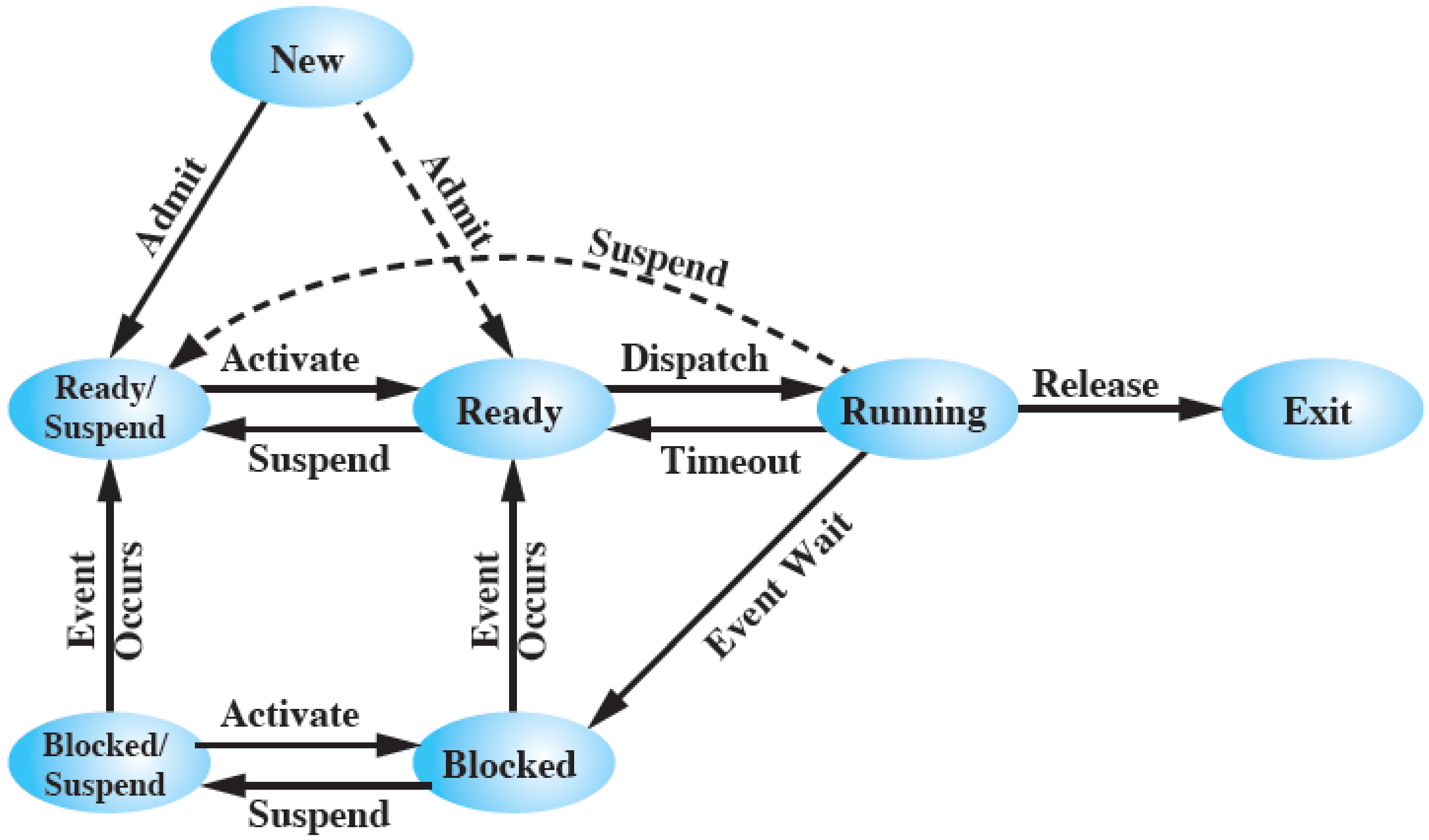
Weekly program

- ✓ Week 1 – Operating System Overview
- ✓ Week 2 – Processes and Threads
- ❑ **Week 3 – Scheduling**
- ❑ Week 4 – Real-time System Scheduling and Multiprocessor Scheduling
- ❑ Week 5 – Concurrency: Mutual Exclusion and Synchronization
- ❑ Week 6 – Concurrency: Deadlock and Starvation
- ❑ Week 7 – Memory Management
- ❑ Week 8 – Disk and I/O Scheduling
- ❑ Week 9 – File Management
- ❑ Week 10 – Real-world Operating Systems: Embedded and Security
- ❑ Week 11 – Real-world Operating Systems
- ❑ Week 12 – Revision of the course
- ❑ Week 13 – Extra revision (if needed)

Key Concepts From Last Week

- A process is an entity consisting of two essential elements: program code and a set of data
- Processes Control Block is a data structure that stores all relevant information related to a process
- A process passes through different states throughout its lifetime
  - Two-state process model
  - Five-state model
  - Seven-state model

Key Concepts From Last Week



Key Concepts from last week

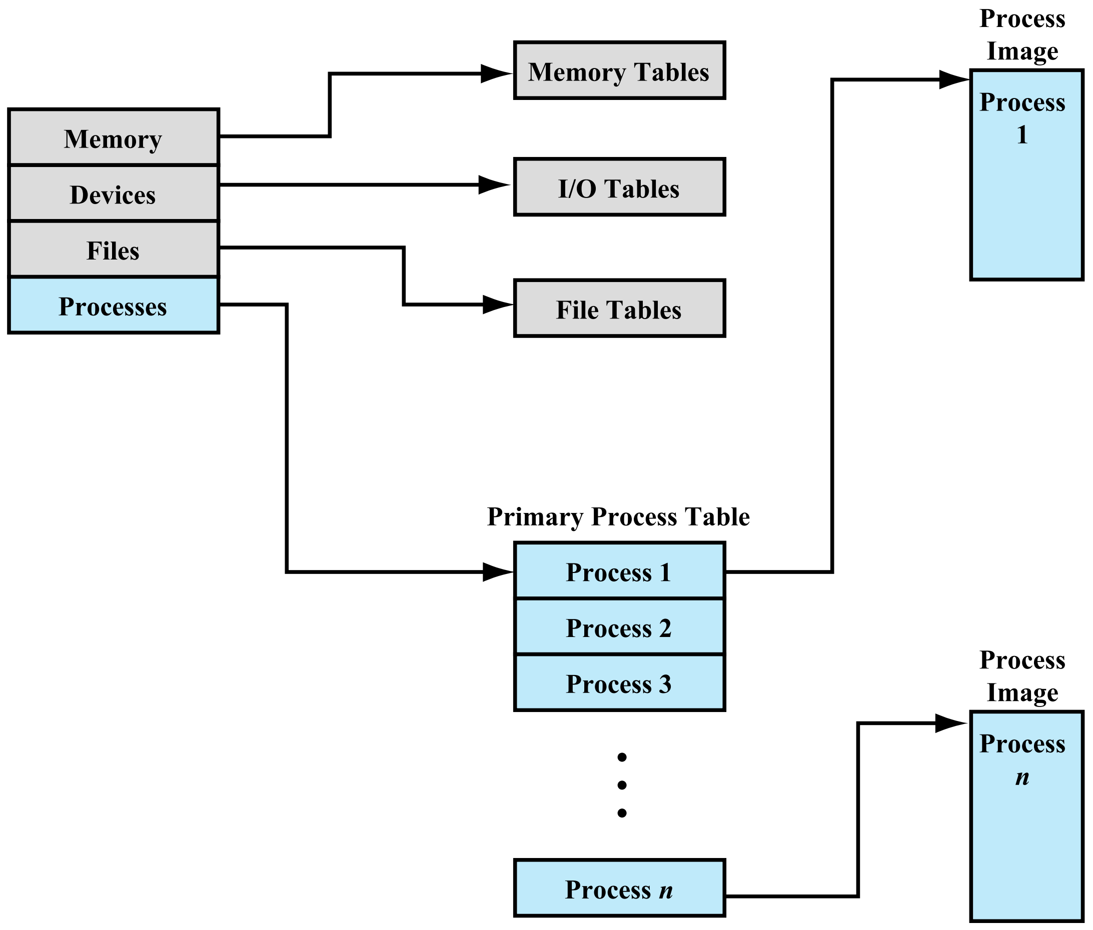


Figure 3.11 General Structure of Operating System Control Tables

Key Concepts from last week

- Significant amount of data is stored in PCB
  - Process ID, Process State Information, Process Control Information
- The difference between context switching and mode switching
- In multithreading environment: Process is related to resource ownership and Thread (light weight process) is related to program execution
- Types of threads: ULT, KLT, Combined

Week 03 Lecture Outline

Scheduling

- ❑ Scheduling – what, why and when?
- ❑ Types of Scheduling
- ❑ Short time Scheduling/CPU Scheduling
- ❑ Criteria for CPU Scheduling
- ❑ Use of priority
- ❑ Scheduling:
    - ❑ Selection function and
    - ❑ Decision mode
- ❑ Scheduling Algorithms
    - ❑ First Come First Served (FCFS)
    - ❑ Round Robin (RR)
    - ❑ Shortest Process Next (SPN)
    - ❑ Shortest Remaining Time (SRT)
    - ❑ Highest Response Ration Next (HRRN)
    - ❑ Feedback (FB)

Videos to watch before lecture

What is Scheduling?

- On a multi-programmed system
  - We may have more than one *Ready* process
- On a batch system
  - We may have many jobs waiting to be run
- On a multi-user system
  - We may have many users concurrently using the system
- The *scheduler* decides who to run next.
  - The process of choosing is called ***scheduling***.

Is scheduling important?

- It is not in certain scenarios
  - If you have no choice
- Early systems
  - Usually batching
  - Scheduling algorithm simple
    - Run next on tape or next on punch tape
  - Only one thing to run
- Simple PCs
  - Only ran a word processor, etc.…
- Simple Embedded Systems
  - TV remote control, washing machine, etc.…

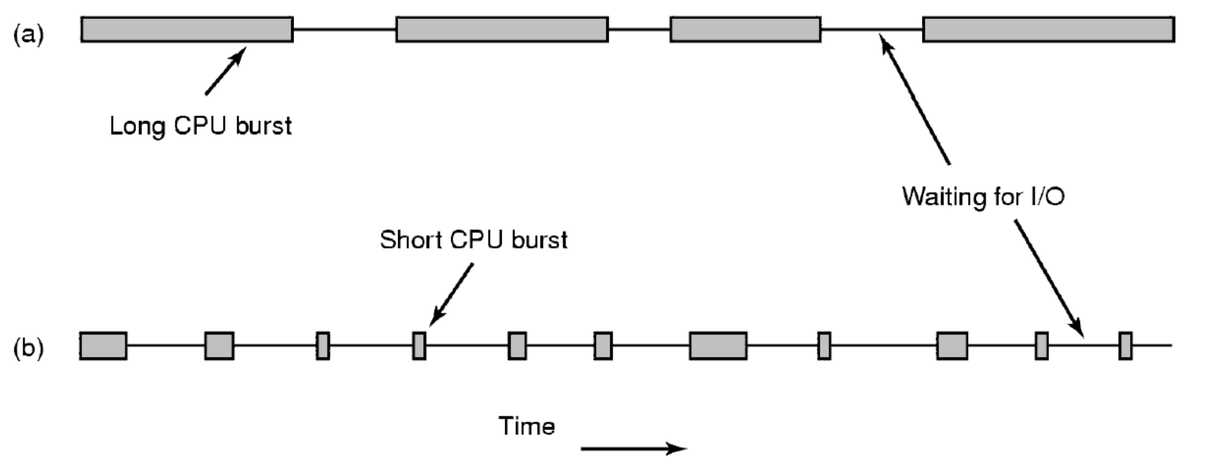
Is scheduling important?

- It is in more complex scenarios
  - Multitasking/Multi-user systems

- Example
  - Email daemon takes 2 seconds to process an email
  - User clicks button on application.
    - **Scenario 1**: Run daemon, then application
      - System appears really sluggish to the user
    - **Scenario 2**: Run application, then daemon
      - Application appears really responsive, small email delay is unnoticed

- Scheduling decisions can have a dramatic effect on the perceived performance of the system
  - Can also affect <u>correctness of a system</u> with deadlines

Terminology



a) CPU-Bound process

- Spends most of its computing
- Time to completion largely determined by received CPU time

b) I/O-Bound process

- Spend most of its time waiting for I/O to complete
  - Small bursts of CPU to process I/O and request next I/O
- Time to completion largely determined by I/O request time

Observations

1. Generally, technology is increasing CPU speed much faster than I/O speed
   - CPU bursts becoming shorter, I/O waiting is relatively constant
   - Processes are becoming more I/O bound

2. We need a mix of CPU-bound and I/O-bound processes to keep both CPU and I/O systems busy

3. Process can go from CPU- to I/O-bound (or vice versa) in different phases of execution

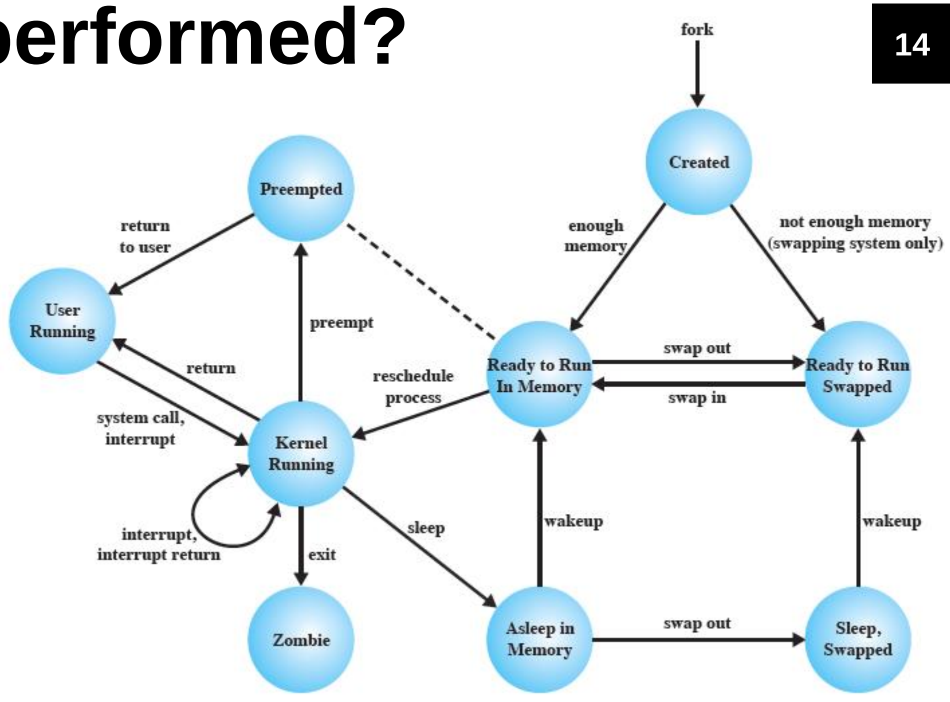
Observations

4. Choosing to run an I/O-bound process delays a CPU-bound process by very little

5. Choosing to run a CPU-bound process prior to an I/O-bound process delays the next I/O request significantly
   - No overlap of I/O waiting with computation
   - Results in device (disk) not as busy as possible

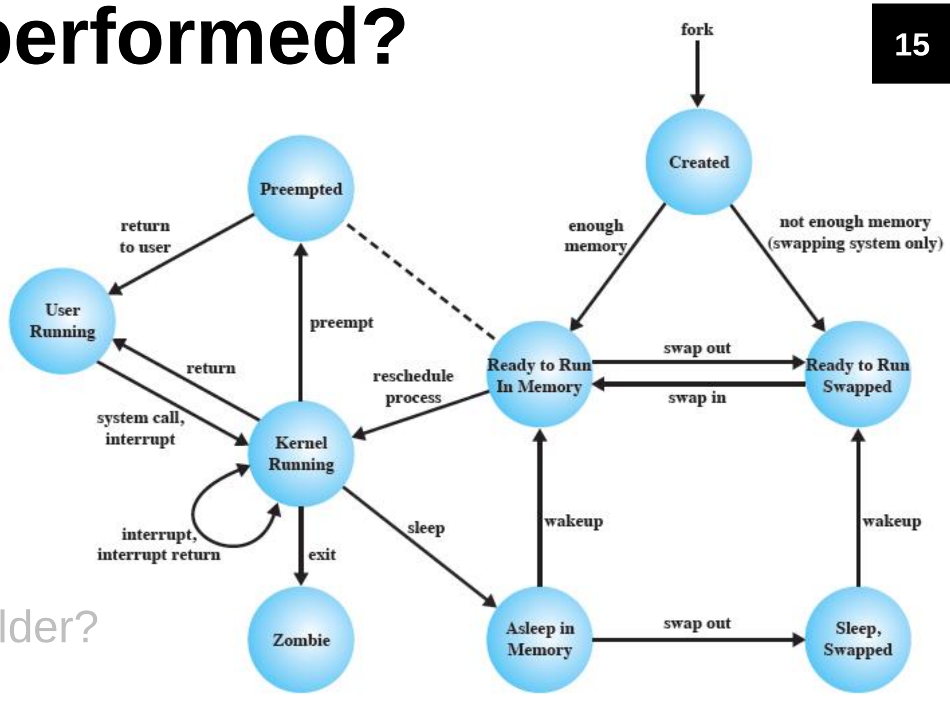
6. Generally, favour I/O-bound processes over CPU-bound processes

When is scheduling performed?



- A new process
  Run the parent or the child?
- A process exits
  - Who runs next?
- A process waits for I/O
  - Who runs next?
- A process blocks on a lock
  - Who runs next? The lock holder?
- An I/O interrupt occurs
  - Who do we resume, the interrupted process or the process that was waiting?
- On a timer interrupt? (Preemptive versus Non-preemptive Scheduling)

When is scheduling performed?



- A new process
  Run the parent or the child?
- A process exits
  - Who runs next?
- A process waits for I/O
  - Who runs next?
- A process blocks on a lock
  - Who runs next? The lock holder?
- An I/O interrupt occurs
  - Who do we resume, the interrupted process or the process that was waiting?
- On a timer interrupt? (Preemptive versus Non-preemptive Scheduling)

- Generally, a scheduling decision is required when a process (or thread) can no longer continue, or when an activity results in more than one ready process.

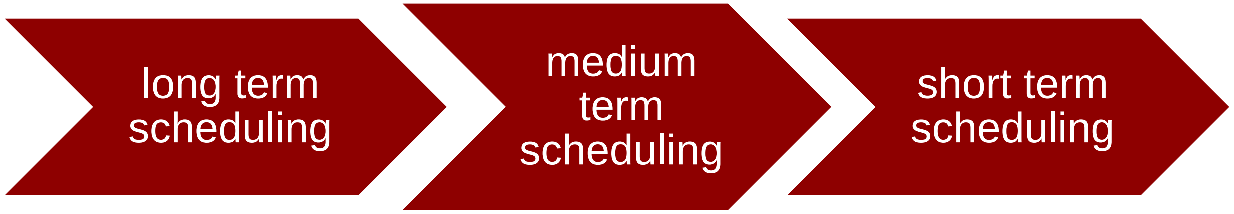
Types of scheduling

There are four kinds of scheduling in an OS:

1. **Long term scheduling**
   – The decision to add to the pool of processes to be executed
2. **Medium term scheduling**
   – The decision to add to the number of processes that are partially or fully in main memory
3. **Short term scheduling**
   – The decision as to which available process will be executed by the processor
4. **I/O scheduling**
   – The decision as to which process’s pending I/O request shall be handled by an available I/O device.

Processor Scheduling

- Aim is to assign processes to be executed by the processor in a way that meets system objectives, such as response time, throughput, and processor efficiency
- Broken down into three separate functions:



- The names suggest the relative time scales with which these functions are performed

Processor Scheduling

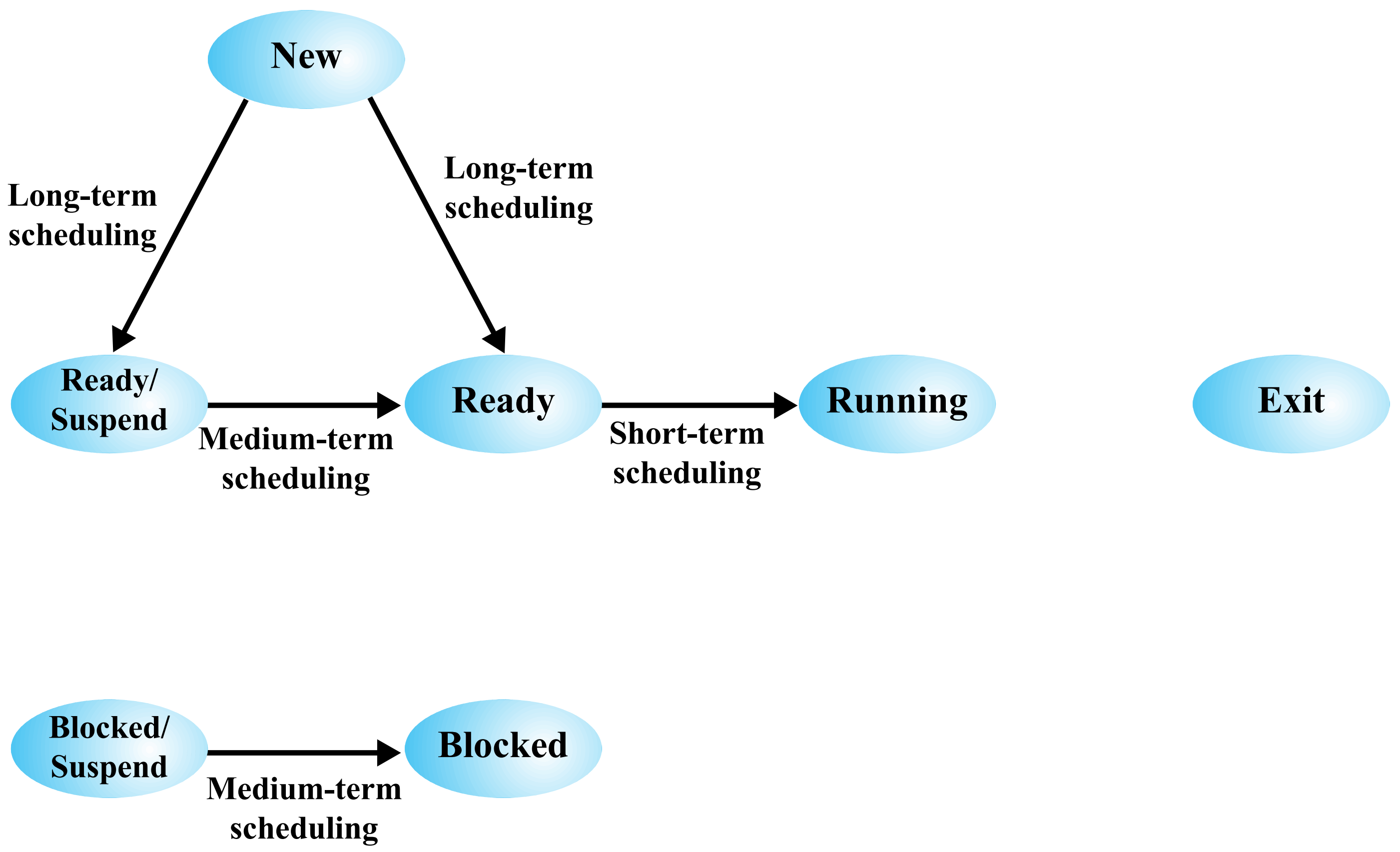
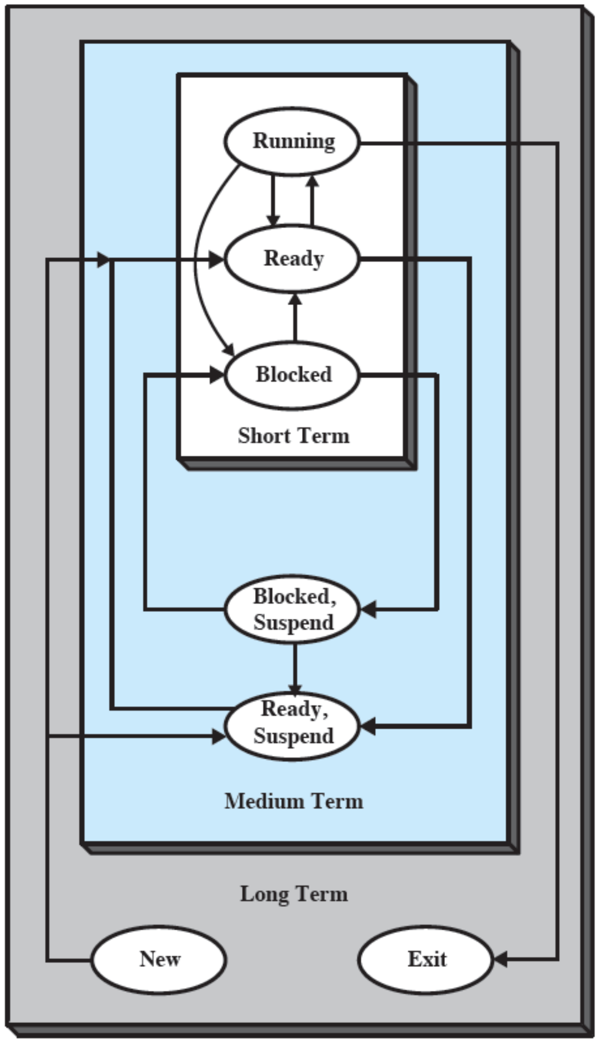


Figure 9.1 Scheduling and Process State Transitions

Processor Scheduling



Processor Scheduling

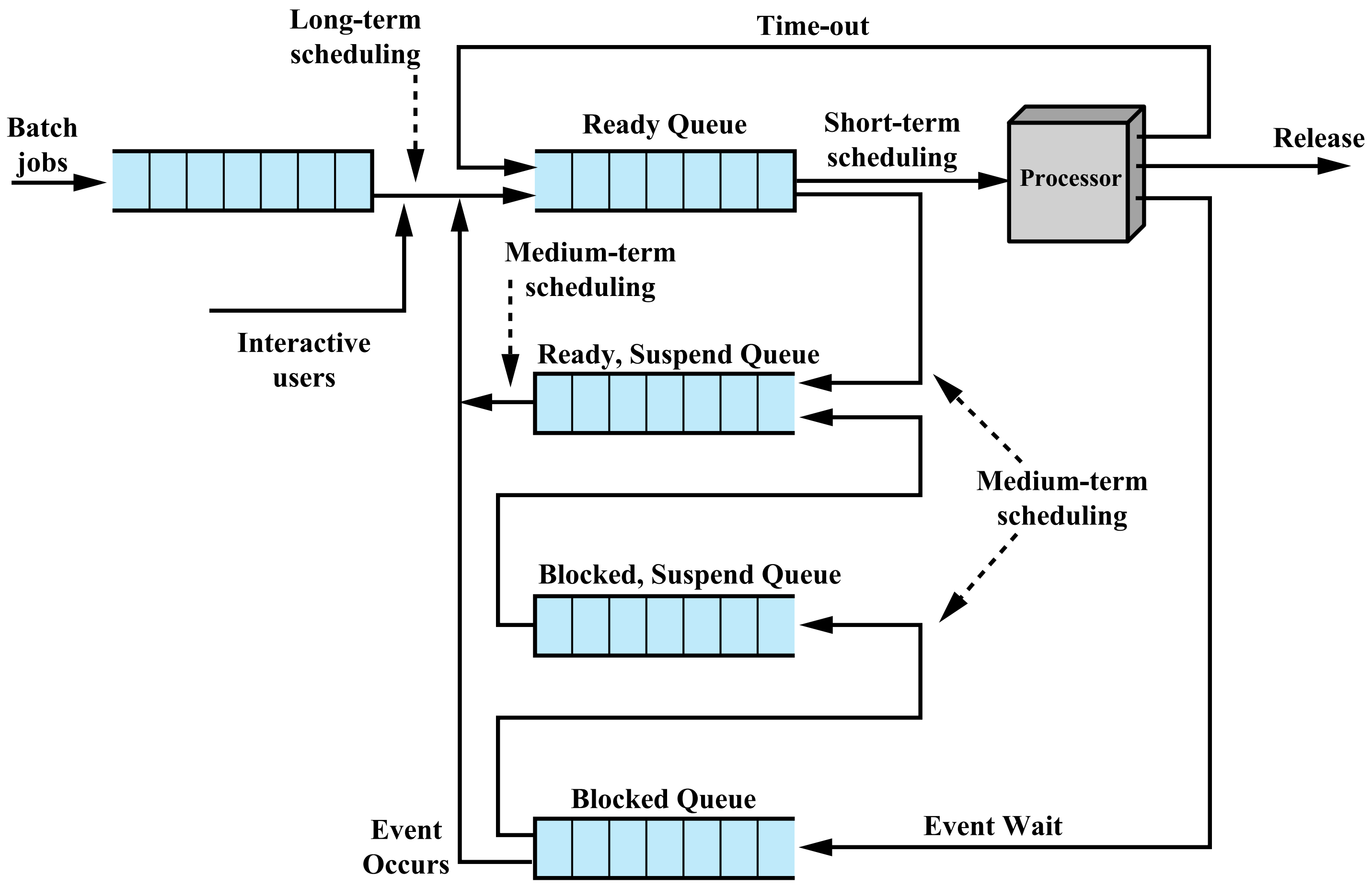
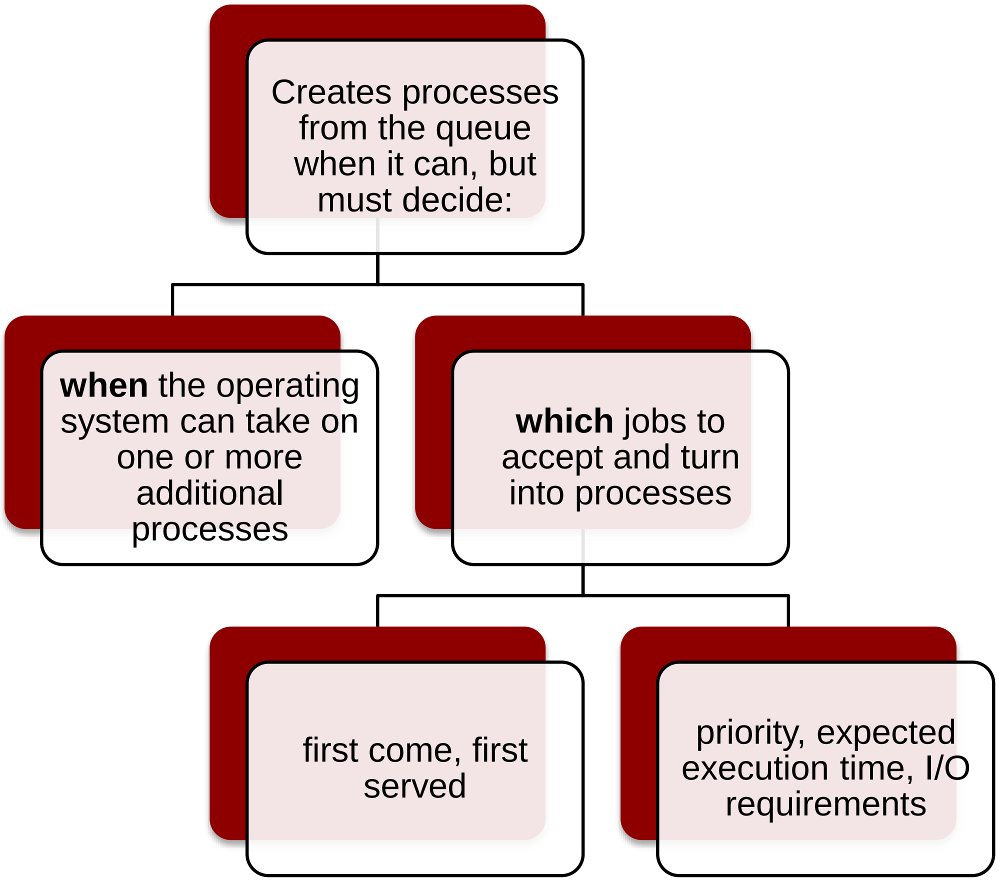


Figure 9.3 Queuing Diagram for Scheduling

Scheduling: Long-Term

- Determines which programs are admitted to the system for processing
- Controls the degree of multiprogramming
    - the more processes that are created, the smaller the percentage of time that each process can be executed
    - may limit to provide satisfactory service to the current set of processes


Creates processes from the queue when it can, but must decide:
when the operating system can take on one or more additional processes
which jobs to accept and turn into processes
first come, first served
priority, expected execution time, I/O requirements

Scheduling: Long-Term

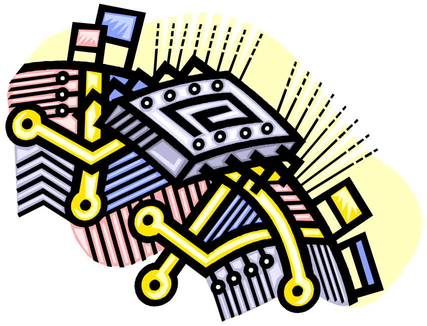
- For interactive-programs in a time-sharing system
  - Process creation request generated when user attempt to connect to a system
  - Are not queued up and kept waiting until the system is saturated
    - According to criteria
  - If saturated then refused with an error message

Scheduling: Medium-Term

- Part of the swapping function
- Swapping-in decisions are based on the need to manage the degree of multiprogramming
    - considers the memory requirements of the swapped-out processes

- Executes more frequently than long term scheduler.

- It is part of the **memory management** system.
    - We will look at this topic later.



Scheduling: Short-Term

- Known as the dispatcher
- Executes most frequently
- Makes the fine-grained decision of which process to execute next
- Invoked when an event occurs that may lead to the blocking of the current process or that may provide an opportunity to preempt a currently running process in favor of another
  - Examples:
    - Clock interrupts
    - I/O interrupts
    - Operating system calls
    - Signals (e.g., semaphores)

Short Term Scheduling Criteria

- Main objective is to allocate processor time to optimize certain aspects of system behavior
- A set of criteria is needed to evaluate the scheduling policy

- **User-oriented criteria**
  - relate to the behaviour of the system as perceived by the individual user or process (such as response time in an interactive system)
  - important on virtually all systems
  - **Focus:** Service provided to the user
- **System-oriented criteria**
  - focus in on effective and efficient utilization of the processor (rate at which processes are completed)
  - generally of minor importance on single-user systems
  - **Focus:** System performance

Scheduling Policies

There are many kinds of short term scheduling algorithms.

Schedulers have two main parameters:

- **Selection function**:
- **Decision mode**:

Selection function

- Determines which process, among ready processes, is selected next for execution
- May be based on **priority**, **resource requirements**, or the **execution characteristics** of the process
- If based on execution characteristics then important quantities are:
  - ***w*** = time spent in system so far, waiting
  - ***e*** = time spent in execution so far
  - ***s*** = total service time required by the process, including *e*; generally, this quantity must be estimated or supplied by the user

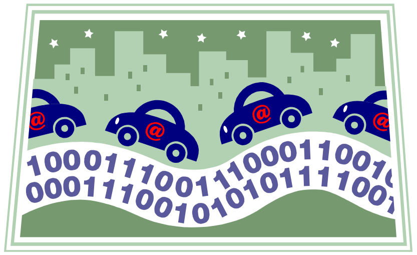
Decision mode

- Specifies the instants in time at which the selection function is exercised

- Two categories:
  - Nonpreemptive
  - Preemptive

Nonpreemptive vs. Preemptive

- **Nonpreemptive**
  - Once a process is in the running state, it will continue until it terminates or blocks itself (e.g. for I/O)
- **Preemptive**
  - Currently running process may be interrupted and moved to ready state by the OS
  - Preemption may occur when new process arrives, on an interrupt, or periodically



Nonpreemptive vs. Preemptive

- **Nonpreemptive**
  - Pros:
    - Less overhead (less switching)
  - Cons:
    - May monopolize
- **Preemptive**
  - Pros:
    - Does not monopolize
    - Better overall service to all processes
  - Cons:
    - More overhead (more switching)
      - Overhead minimization using efficient switching

Scheduling Algorithms

- First Come First Served (FCFS)
- Round Robin (RR)
- Shortest Process Next (SPN)
- Shortest Remaining Time (SRT)
- Highest Response Ration Next (HRRN)
- Priority Scheduling (PS)
- Feedback (FB)
- Fair Share Scheduling (FSS)

Performance Indices

- **Arrival Time (T_a)**: Time when process enter the system (short time scheduling)
- **Service Time (T_s)**: Total execution time required by the process
- **Turnaround Time (T_r)**: Total time that the items spends in the system
- **Normalized Turnaround Time**: T_r/T_s: Relative delay experienced by a process. Minimum value is 1.0; Increasing values correspond to decreasing level of service.

Running Example

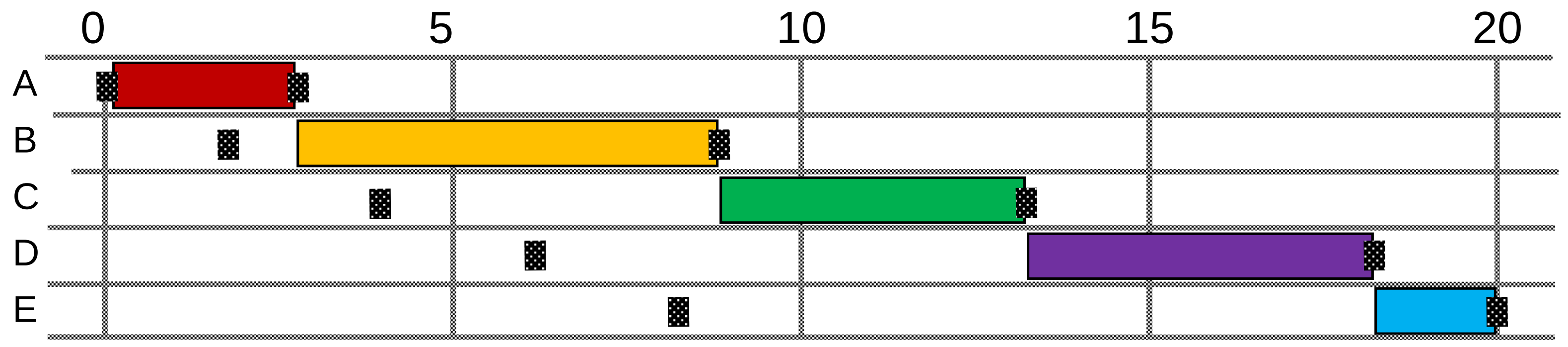
Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

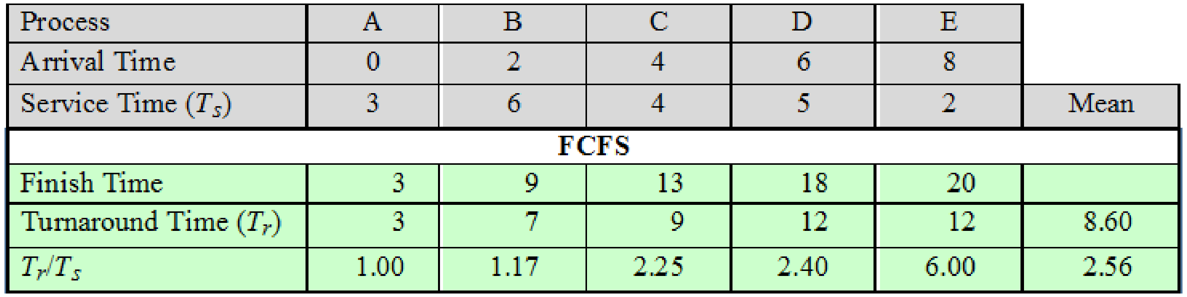
First-Come-First-Served (FCFS)

- Simplest scheduling policy
- Also known as first-in-first-out (FIFO) or a strict queuing scheme
- When the current process ceases to execute, the process that has been in the Ready queue the longest is selected
- Non-preemptive policy
- It may be implemented by using the ready queue as a FIFO queue.
- Performs much better for long processes than short ones
- Tends to favor processor-bound processes over I/O-bound processes

FCFS - Example

FCFS on this set of jobs:



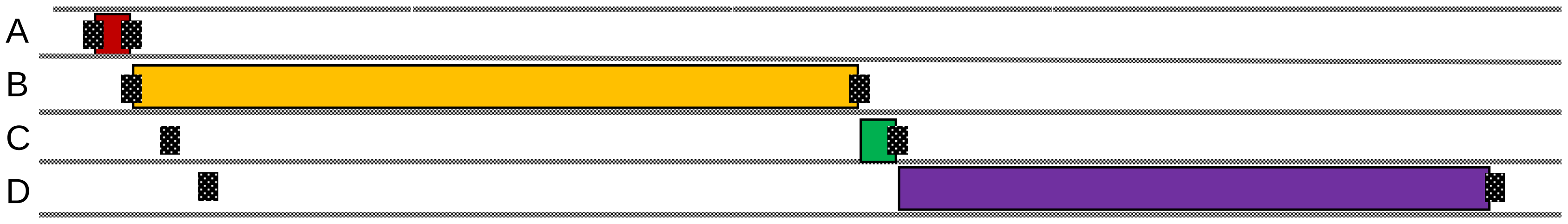


Process	A	B	C	D	E	
Arrival Time	0	2	4	6	8	
Service Time (T_s)	3	6	4	5	2	Mean
FCFS						
Finish Time	3	9	13	18	20	
Turnaround Time (T_r)	3	7	9	12	12	8.60
T_r/T_s	1.00	1.17	2.25	2.40	6.00	2.56

FCFS Performance

- FCFS is simple but performs badly if the job mix contains jobs of widely different characteristics, for example:

process	A	B	C	D
arrival time	0	1	2	3
service time	1	100	1	100
start time	0	1	101	102
finish time	1	101	102	202
turnaround time T_r	1	100	100	199
T_r/T_s	1.0	1.0	100	2.0



- Here the long jobs (B, and D) get a reasonable turnaround time, but the short job (C) has an unreasonable turnaround time.

FCFS Performance

- FCFS also has the **disadvantage** that a CPU bound job can monopolise the processor, leaving I/O devices idle.
  - Suppose there is one processor bound job and many I/O bound jobs
    - When CPU bound job runs all I/O bound jobs wait and I/O devices are idle
    - When the CPU bound job finishes, the I/O bound jobs quickly become blocked waiting and the CPU becomes idle.
    - May result in inefficient use of both the processor and the I/O devices
    - **Convoy effect**

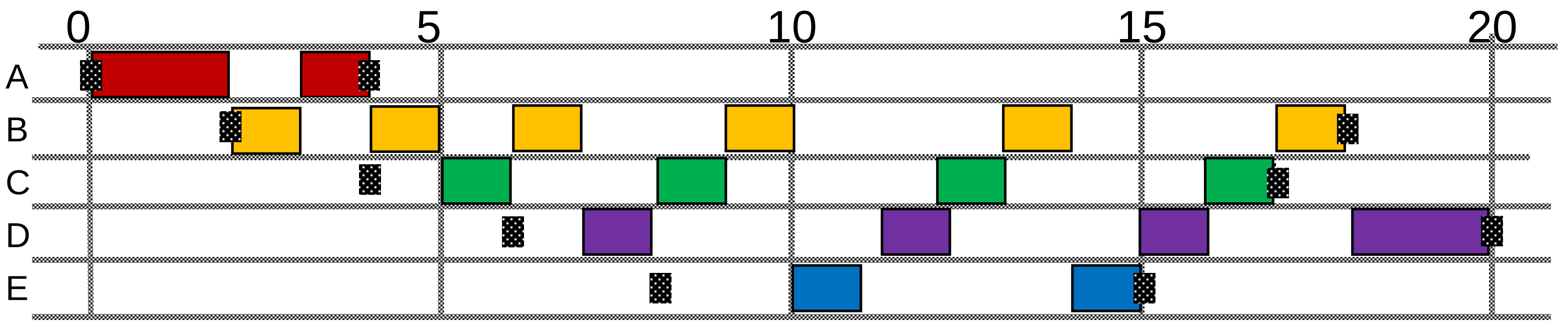
- FCFS is not an attractive scheduling policy on its own
- FCFS is best used in **combination** with another policy

Round Robin

- Stop monopolization of length job by preemption
- In **Round Robin scheduling**, (RR) or **time slicing**, a clock regularly interrupts and pre-empts the running process.
- The next process to run is chosen by an FCFS strategy.
- The main parameter is the **length of the time slice**. A short time slice ensures that brief processes move through the system rapidly, but means many interrupts.
- What happens when time slice becomes infinite?

RR - Example

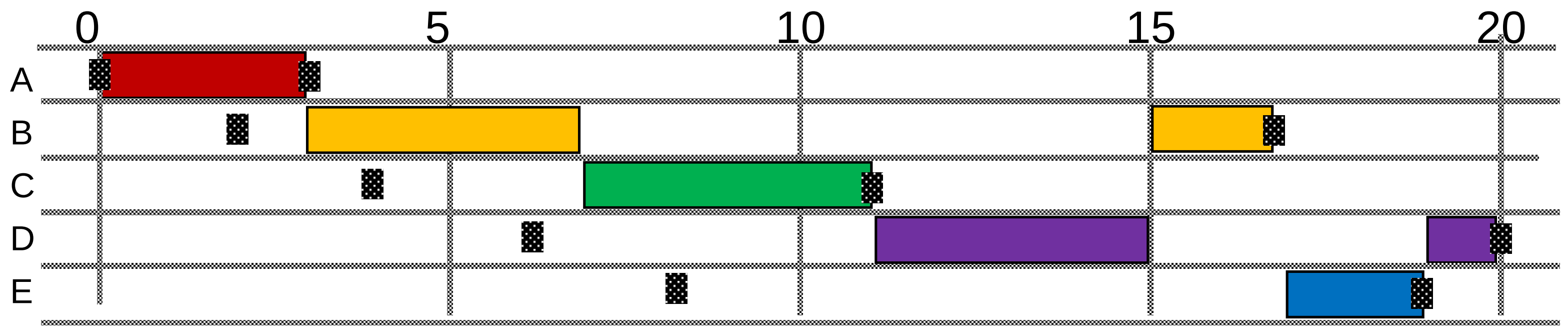
- Example for time slice of 1 unit:



Process	A	B	C	D	E	
Arrival Time	0	2	4	6	8	
Service Time (T_s)	3	6	4	5	2	Mean
RR q = 1						
Finish Time	4	18	17	20	15	
Turnaround Time (T_r)	4	16	13	14	7	10.80
T_r/T_s	1.33	2.67	3.25	2.80	3.50	2.71

RR - Example

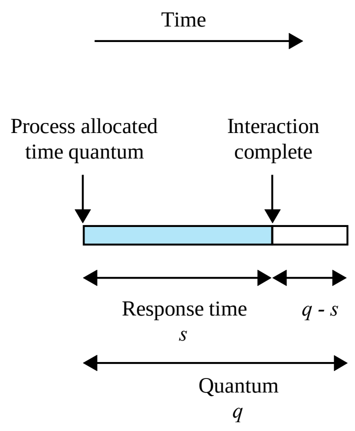
- Example with time slice of 4 units:



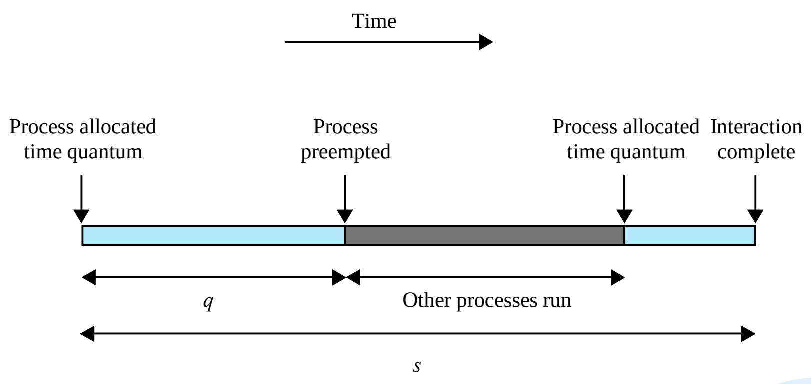
Process	A	B	C	D	E	
Arrival Time	0	2	4	6	8	
Service Time (T_s)	3	6	4	5	2	Mean
RR $q = 4$						
Finish Time	3	17	11	20	19	
Turnaround Time (T_r)	3	15	7	14	11	10.00
T_r/T_s	1.00	2.5	1.75	2.80	5.50	2.71

Effect of preemption time quantum

- The time slice may be chosen to be a little **larger than the time required for a typical interaction**; that is, most processes should require only one time slice.



(a) Time quantum greater than typical interaction



(b) Time quantum less than typical interaction

Round Robin Performance

- Round Robin is effective for general purpose time-sharing systems.
- Each context switch has the OS using the CPU instead of the user process
  - give up CPU, save all info, reload w/ status of incoming process
  - Say 20 ms quantum length, 5 ms context switch
  - Waste of resources
    - 20% of CPU time (5/25) for context switch
  - If 500 ms quantum, better use of resources
    - 1% of CPU time (5/505) for context switch
    - Bad if lots of users in system – interactive users waiting for CPU
  - Balance found depends on job mix
- Still favors CPU-bound processes
  - An I/O bound process uses the CPU for a time less than the time quantum and then is blocked waiting for I/O
  - A CPU-bound process runs for its whole time slice and goes back into the ready queue (in front of the blocked processes)

Virtual Round Robin (VRR)

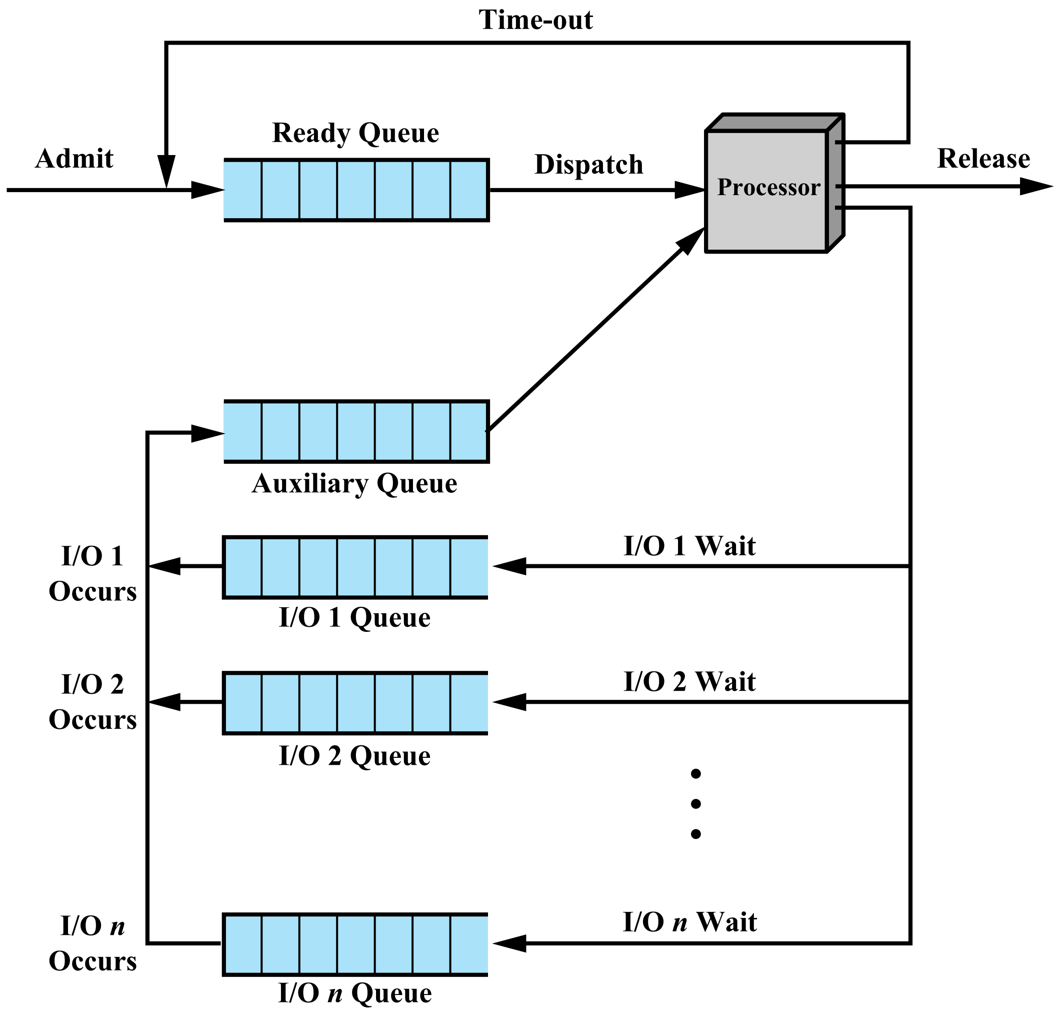


Figure 9.7 Queuing Diagram for Virtual Round-Robin Scheduler

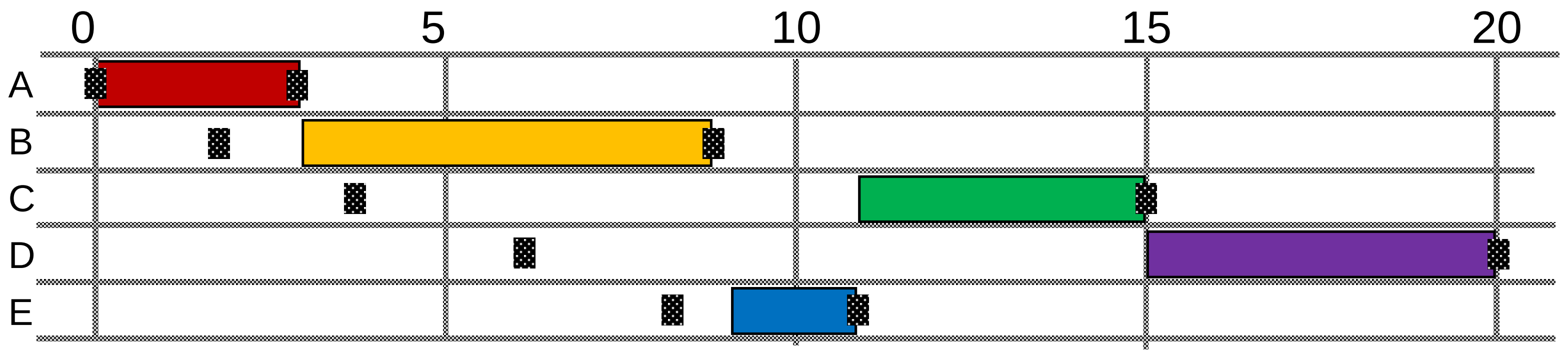
Virtual Round Robin (VRR)

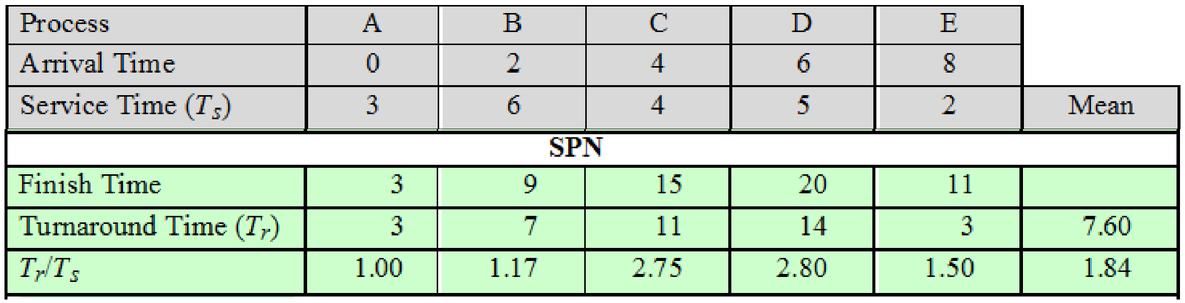
- **Virtual Round Robin** (VRR) avoids unfairness of RR.
- New processes arrive & join Ready queue - FCFS.
- Timed-out running processes rejoin Ready queue.
- I/O blocked processes join appropriate I/O queue.
  - (So far this is standard)
- NEW FEATURE: add FCFS *Auxiliary* queue.
- Processes released from I/O wait queue join *Auxiliary* queue.
- Scheduler dispatches processes from *Auxiliary* queue before processes from Ready queue. But these processes only run with up to the time remaining from their last basic time quantum (before they were blocked).

SPN

- The **Shortest Process Next** (SPN) policy always chooses the process which has the shortest expected running time.
- This policy is **non-pre-emptive**.
- The process with the shortest expected processing time is selected next (Shortest Job First – SJF)
- A short process will jump to the head of the queue

SPN - Example





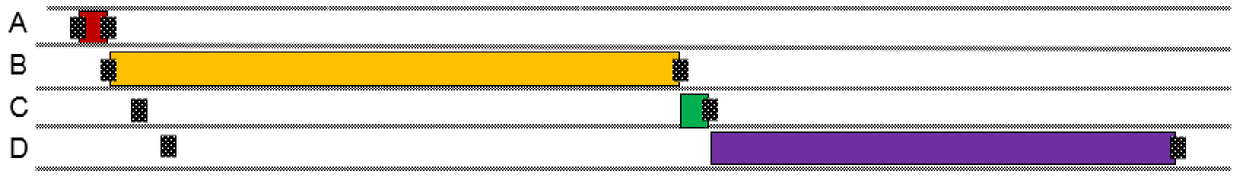
Process	A	B	C	D	E	
Arrival Time	0	2	4	6	8	
Service Time (T_s)	3	6	4	5	2	Mean
SPN						
Finish Time	3	9	15	20	11	
Turnaround Time (T_r)	3	7	11	14	3	7.60
T_r/T_s	1.00	1.17	2.75	2.80	1.50	1.84

- This example assumes that the processor knows the service time before it runs the processes.

SPN - Performance

- Gives minimum average waiting time for a given set of processes.
- Possibility of starvation for longer processes
- Not suitable for time-sharing
  - No premption
- The same bad example as for FCFS applies: with a mixture of very long and very short processes, SPN can be **unfair to short processes**.

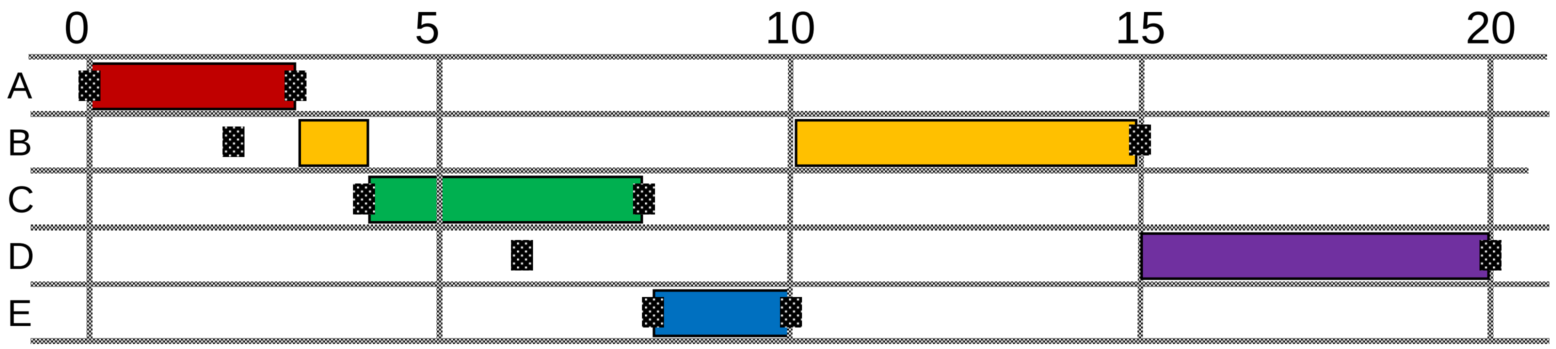
process	A	B	C	D
arrival time	0	1	2	3
service time	1	100	1	100
start time	0	1	101	102
finish time	1	101	102	202
turnaround time T_r	1	100	100	199
T_r/T_s	1.0	1.0	100	2.0



SRT

- The **Shortest Remaining Time** (SRT) scheduler chooses a process with the shortest expected remaining time.
- This is a kind of **preemptive version of SPN**.
- When a new process becomes available, the OS may **preempt** a running process if the expected time to completion for the currently running process is longer than the expected run time of the new process.
- As with SPN:
  - the OS must estimate runtimes
  - long processes may be starved

SRT Example



Process	A	B	C	D	E	
Arrival Time	0	2	4	6	8	
Service Time (T_s)	3	6	4	5	2	Mean
SRT						
Finish Time	3	15	8	20	10	
Turnaround Time (T_r)	3	13	4	14	2	7.20
T_r/T_s	1.00	2.17	1.00	2.80	1.00	1.59

Notice that the shortest 3 processes all receive immediate attention, and hence have a normalised turnaround time of 1.0 .

SRT - Performance

Pros:

- Does not have the bias in favour of long processes like in FCFS
- No additional interrupt are generated like in RR
- Turnaround time is better than SPN because short jobs get immediate attention.

Cons:

- Elapsed times must be recorded; this increases overhead.

HRRN

- The **Highest Response Ratio Next** (HRRN) policy chooses the next process to run to be the one with the highest value of RR, defined below.

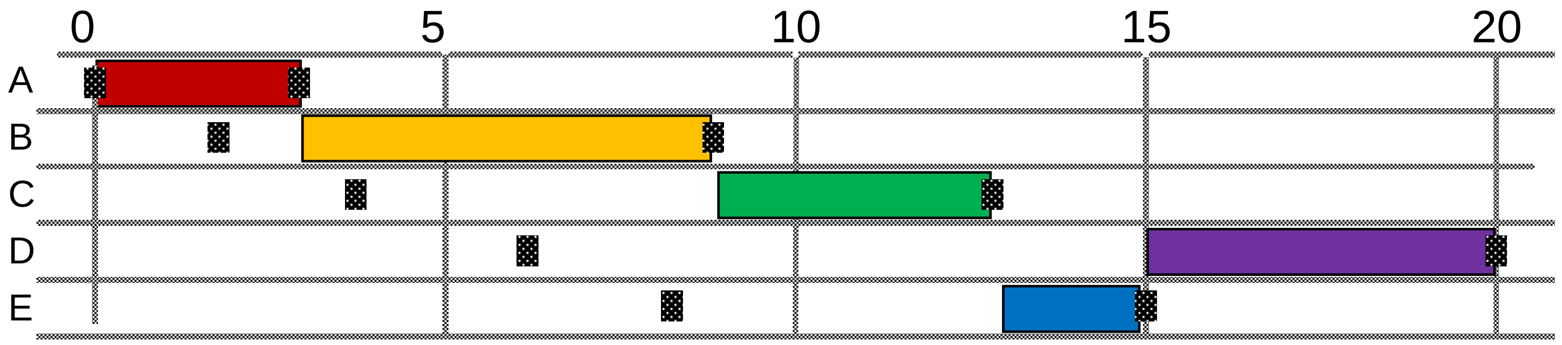
- The policy is **non-pre-emptive**.

- Define $RR = \frac{w+s}{s}$

  where
  w = time spent waiting, and
  s = expected service time.

- Thus, at the completion of a job, $RR = T_r/T_S$, which is the value we are trying to minimise.

HRRN - Example



Process	A	B	C	D	E	
Arrival Time	0	2	4	6	8	
Service Time (T_s)	3	6	4	5	2	Mean
HRRN						
Finish Time	3	9	13	20	15	
Turnaround Time (T_r)	3	7	9	14	7	8.00
T_r/T_s	1.00	1.17	2.25	2.80	3.5	2.14

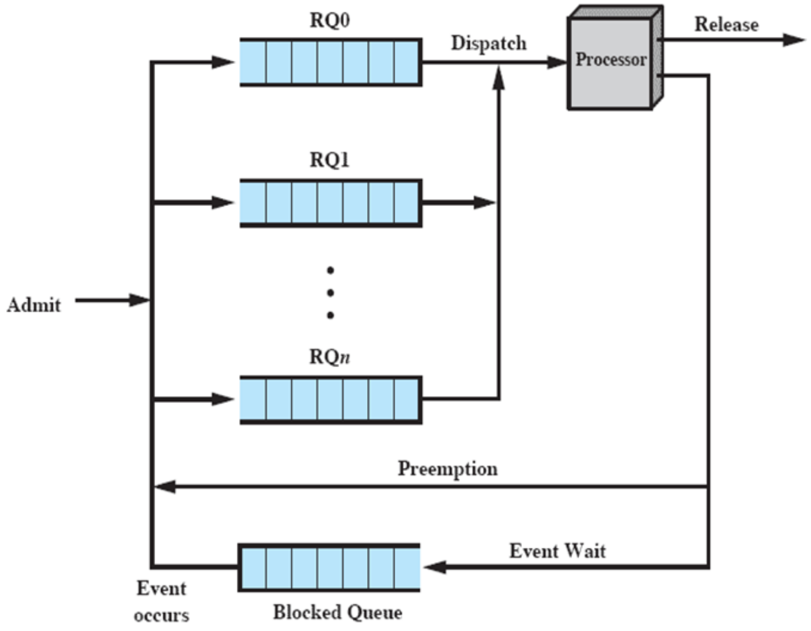
HRRN - Performance

- It considers the age of the process
- If a process is waiting and getting no service, then *w* increases while *s* remains constant, so $RR = \frac{w+s}{s}$ increases and the process is more likely to be chosen by the scheduler.
- Even though short processes are favoured with aging longer process will eventually get through

- Like SRT & SPN need to know expected service time.

Priority Scheduling

- Priority is associate with each process
- Priorities can be managed with a queue for each priority level
- Dispatcher selects process from the highest priority
- If queue of priority 0..m are empty and queue m+1 is non-empty, then a process is chosen from queue m+1.
- This procedure can be used in conjunction with a separate scheduling algorithm for each queue
- Preemptive or non-preemptive?
- SPN is a special case of priority scheduling.



Priority Boosting

- Major problem in priority scheduling is **starvation**
- A steady stream of higher-priority process can leave some low-priority process waiting indefinitely

RUMOR: When they shut down the IBM7094 at MIT in 1973, they found a low-priority process that had been submitted in 1967 and had not yet been run.

- A solution is **aging**
  - Gradually increase the priority of processes that wait in the system for a long time

Priority Levels

- Windows 2000 priority levels
  - 32 priority levels from 0 to 31
    - Priority 0 : system level
    - Priority 1 – 15 : variable levels
    - Priority 16 – 31 : real time levels

- Win32 API priority levels
  - 6 priority Class
    - Real-time, high, above normal, normal, below normal and idle
  - 7 relative thread priority levels in each process priority class
    - Time Critical, Highest, Above Normal, Normal, Below Normal, Lowest and Idle

Priority Boosting

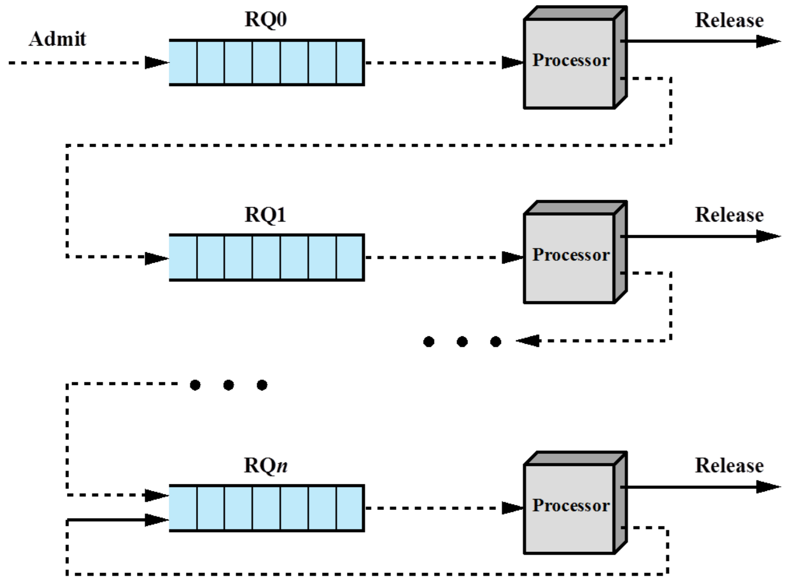
- On completion of I/O operations
- After waiting on executive events or *semaphores*
- After threads in the foreground process complete a wait operation
- When a GUI thread wakes up after windowing activity
- CPU starvation

FB

- SPN, SRT, HRRN can not be used if no indication of the relative length of the processes are available
- We can prefer shorter jobs by penalizing the jobs that have been running for long
- Here we are focusing
  - NOT on the time remaining
  - on the time already executed

Feedback

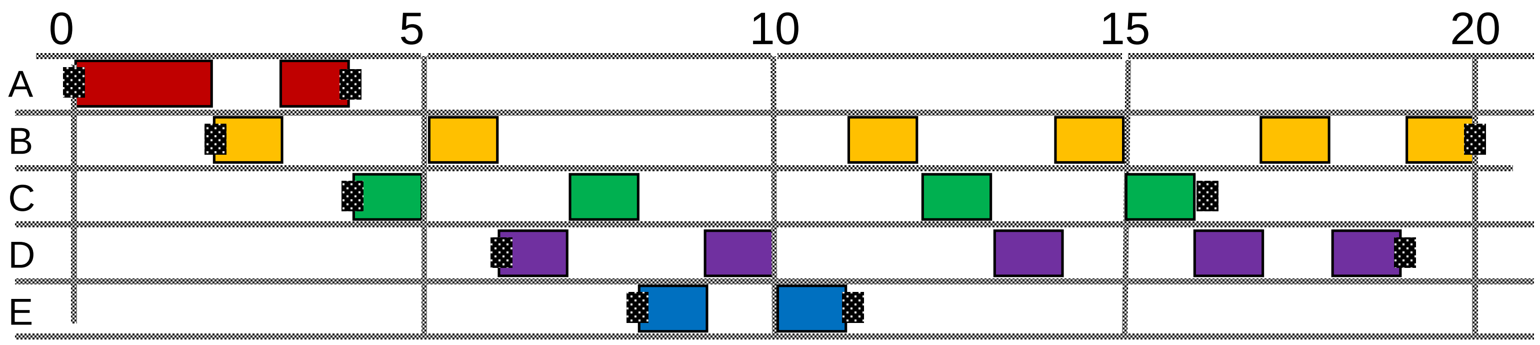
- Preemptive scheduling
- For the **FB** policy, we use a set of queues, one for each priority level.



- When a process enters the system
  - it enters the **high priority queue** (*priority i*, with *i=0*).
  - It receives a time slice of service
- When it is pre-empted or blocked,
  - it returns to a lower queue priority queue (*priority i+1*).
- Within each queue, an FCFS mechanism is used; except for the lowest priority queue, which is round-robin.

FB Example 1

With equal time slicing:



Process	A	B	C	D	E	
Arrival Time	0	2	4	6	8	
Service Time (T_s)	3	6	4	5	2	Mean
FB q = 1						
Finish Time	4	20	16	19	11	
Turnaround Time (T_r)	4	18	12	13	3	10.00
T_r/T_s	1.33	3.00	3.00	2.60	1.5	2.29

In this case, behaviour is similar to RR with time quantum of 1.

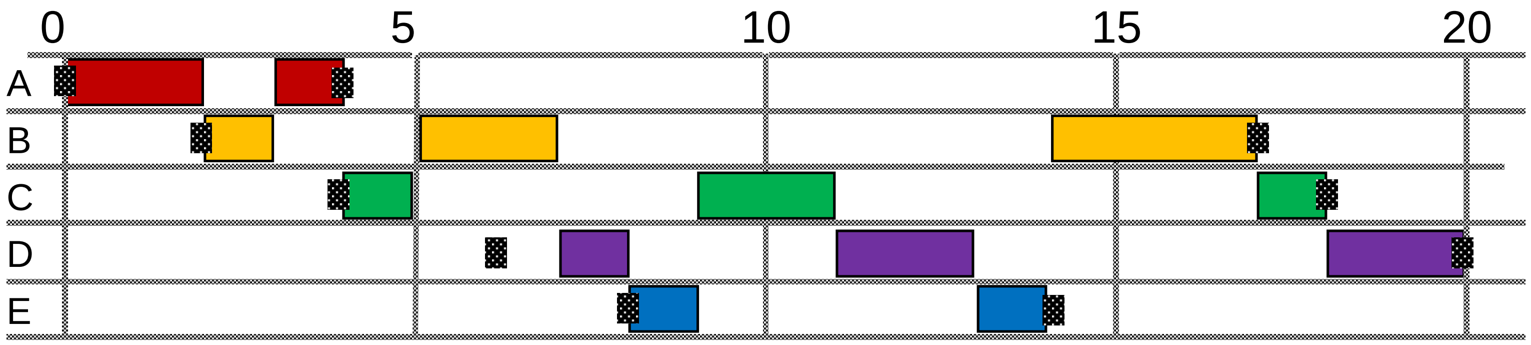
FB - Performance

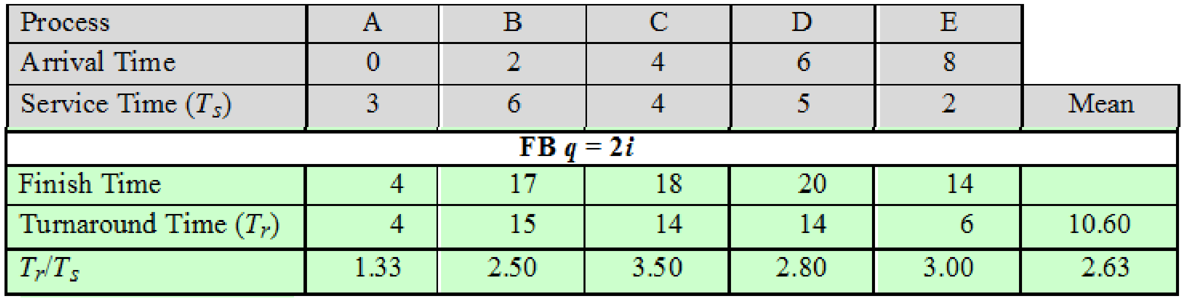
- Short process will complete quickly
- Longer process will gradually drift down
- Starvation can occur for long jobs if there are many new jobs (i.e. turnaround time s_t_r_e_c_h_e_s out).

- **Remedy**: In stead of constant time slices, lower priority queues can get larger time slices (for example, priority *k* can get a time slice of *2^k* units.

Feedback Example 2

with time slice of *2^k* for queue *k:*





Process	A	B	C	D	E	
Arrival Time	0	2	4	6	8	
Service Time (T_s)	3	6	4	5	2	Mean
FB $q = 2i$						
Finish Time	4	17	18	20	14	
Turnaround Time (T_r)	4	15	14	14	6	10.60
T_r/T_s	1.33	2.50	3.50	2.80	3.00	2.63

FB - Performance

With varying preemption time

- Long processes may still starve.
- **Remedy**: Possible to promote a process to a higher-priority queue after it has waited without service in its current queue.

Characteristics of scheduling policies

	FCFS	Round robin	SPN	SRT	HRRN	Feedback
Selection function	max[w]	constant	min[s]	min[s – e]	$\max\left(\frac{w+s}{s}\right)$	(see text)
Decision mode	Non-preemptive	Preemptive (at time quantum)	Non-preemptive	Preemptive (at arrival)	Non-preemptive	Preemptive (at time quantum)
Through-Put	Not emphasized	May be low if quantum is too small	High	High	High	Not emphasized
Response time	May be high, especially if there is a large variance in process execution times	Provides good response time for short processes	Provides good response time for short processes	Provides good response time	Provides good response time	Not emphasized
Overhead	Minimum	Minimum	Can be high	Can be high	Can be high	Can be high
Effect on processes	Penalizes short processes; penalizes I/O bound processes	Fair treatment	Penalizes long processes	Penalizes long processes	Good balance	May favor I/O bound processes
Starvation	No	No	Possible	Possible	No	Possible

Fair-Share Scheduling

- Scheduling algorithms usually individual processes
- Multiple processes might belong to the same application
  - Multiple thread might belong to the same process
- User Point of View:
  - NOT how individual process (thread) is doing
  - How the application (process) is doing
- The idea can even can be extended for a group of independent processes
  - For example processes run by a group

Summary

- OS must make three types of scheduling decisions
  - Long-term: when new processes are admitted to the system
  - Medium-term: when a process is loaded into main memory
  - Short-term: when a ready process will be executed by the processor
- A variety of criteria is used in designing CPU scheudler
  - User oriented VS System Oriented
  - Quantitative VS Qualitative
- From user point of view response time is usually most important characteristic of a system
- From system point of view throughput or processor utilization is usually more important

Summary

- Two decision modes for scheduling: Preemptive and Non-preemptive
- Selection may be based on priority, resource requirements, or the execution characteristics of the process
- A variety of algorithms have been developed for short-term scheduling
  - FCFS, RR, SPN, SRT, HRRN, FB, FSS
  - Each has some relative advantage and disadvantage
- The choice of algorithm depends on the expected performance and implementation complexity

References

- **Operating Systems – Internal and Design Principles**
  – By William Stallings
- Chapter 9

THE UNIVERSITY OF NEWCASTLE AUSTRALIA