

Operating System: Process Management and Scheduling

CIT 595
Spring 2008

What is System Software?

- **System Software** is a computer software which
 - Manages and Controls the hardware so that application software can perform a task
- Operating System Software
 - Manages computer hardware
 - Help various applications interact with computer hardware
- Programming Tools
 - Help programmers in application development
 - Tools such as assemblers, compilers, linkers, loaders etc.
- Middleware (not our focus)
 - Above the OS but below the application program layer
 - To support applications distributed over networks

CIT 595

2

Services of Operating System

- Interface manager
 - Human interaction made easy for average user
 - Abstraction, control and sharing for programmers
- **Resource manager** : efficient use of resources i.e. CPU, Memory, and I/O among processes
 - Process Management and Scheduling
 - Provide Synchronization and Deadlock Prevention/Detection mechanism wrt Memory and I/O
 - File Management
- Security: Data protection and isolation
 - Some inherent due to other services
 - File Permissions
 - Passwords etc.

CIT 595

3

OS Service: Resource Management

- Multiple processes share the same *physical memory*
 - Using Disk as a backing store
 - Isolation and Protection: processes limited to execution within own memory space
- Multiple processes share *CPU*
 - Access to CPU is managed by *OS scheduler*
- Multiple process share same *I/O resources*
 - Allow generic interface for I/O through various *system calls*
 - Protection via user and supervisor mode
 - Also manages file system and their mapping on the secondary storage devices such as disks

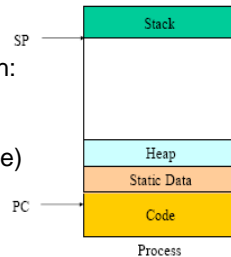
CIT 595

4

Recap: Process

- A process is a name given to a program instance that has been loaded into memory and managed by the operating system
- Process address space is generally organized into *code*, *data (static/global)*, *heap*, and *stack* segments

- Every process in execution works with:
 - Registers: PC, Working Registers
 - Call stack (Stack Pointer Reference)



CIT 595

5

Process Activity

- As process executes over time it can be doing either of the activities or is in following states :

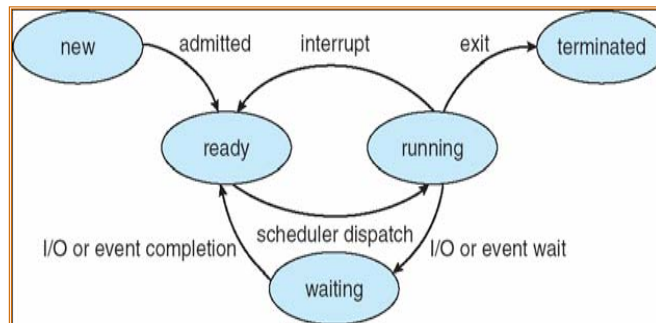
State/Activity	Description
new	Process is being created
running	Instructions are being executed on the processor
waiting/blocked	Process is waiting for some event to occur
ready	Process is waiting to use the processor
terminated	Process has finished execution

Note: state names vary across different OS

CIT 595

6

State Diagram of a Process Activity/State



Note: In uniprocessor system, only one process can be in running state while many processes can be in ready and waiting states

CIT 595

7

Process Management

- OS maintains a data structure for each process called *Process Control Block (PCB)*
- Information associated with each PCB:
 - Process state: e.g. ready, or waiting etc.
 - Program counter: address of next instruction
 - CPU registers: PC, working registers, stack pointer, condition code
 - CPU scheduling information: scheduling order and priority
 - Memory-management information: page table/segment table pointers
 - Accounting information: book keeping info e.g. amt CPU used
 - I/O status information: list of I/O devices allocated to this process and files (open)

CIT 595

8

Process Scheduling

- Idea of multi-tasking
- Realize that process has cpu-burst and I/O burst cycle
 - When I/O burst, CPU idle
 - Exploit the idleness to better achieve parallel tasking
 - On Uniprocessor system switch between processes so fast to give an illusion of parallelism
- Determine which process should be next in line for the CPU
 - Selects from among the processes that are *ready* to execute
 - Done by short term scheduler

CIT 595

9

Issues to Consider

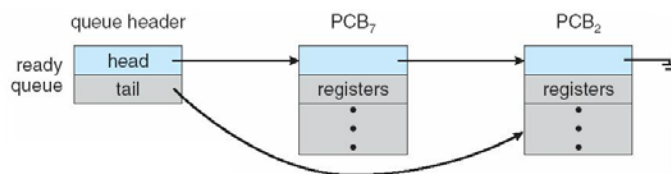
- Efficiency
 - If scheduler is invoked every 100 ms & it takes 10 ms to decide the order of execution of the processes then $10/(100 + 10) = 9\%$ of the CPU is being used simply for scheduling the work
- Fairness
 - Give each process a fair share
- Priority
 - Allow more important processes
- Context switch overhead
 - Saving PCB on process, loading PCB of other

CIT 595

10

Ready Queue

- Processes resident in main memory and that in *ready* state are kept in a *ready queue*
 - Process waits in the ready queue until selected
- Unless a process terminates, it will eventually be put back into a ready queue



Similarly OS keeps device queues for processes waiting for I/O

CIT 595

11

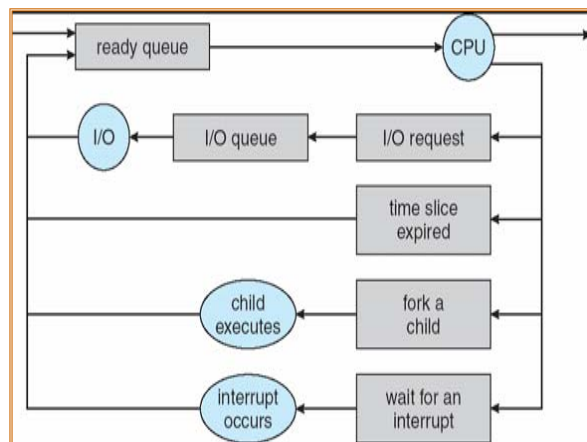
Non-Preemptive vs. Preemptive

- A process can give up CPU in two ways
- **Non-preemptive:** A process voluntarily gives up CPU
 - I/O request
 - Process is blocked, then when request ready it is put back into ready queue
 - A process creates a new child/sub process (more later)
 - Finished Instructions to execute (Process termination)
 - PCB and resources assigned are de-allocated
- **Preemptive:** A process is forced to give up the CPU
 - Interrupted due to higher priority process
 - Each process has fixed time-slice to use CPU

CIT 595

12

Ready Queue Representation

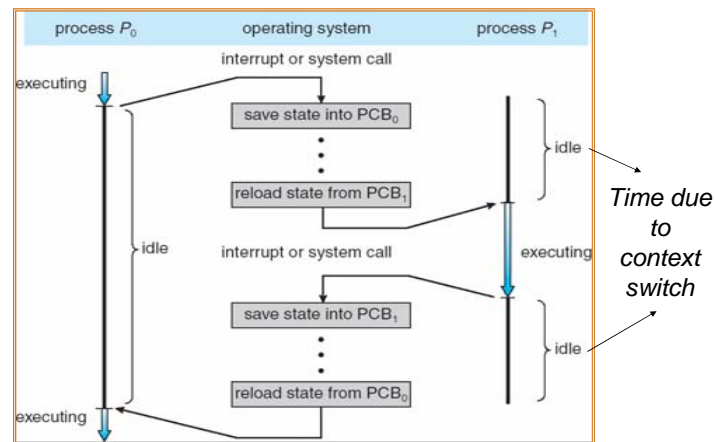


CIT 595

Source: Operating System Concept (7 Ed) – Silberschatz, Gavin and Gagne

13

Context Switch



CIT 595

Source: Operating System Concept (7 Ed) – Silberschatz, Gavin and Gagne

14

Scheduling Algorithm Metrics

- CPU utilization: Fraction of time CPU is utilized
- Throughput: # of processes that completed per time unit
- Turnaround time: amount of time to execute a particular process
 - Waiting in Ready Queue + Executing on CPU + doing I/O
- Response time: amount of time it takes from when a request was submitted until the first response is produced
 - Ideal for interactive systems
- Note:
 - For simplicity illustration and discussion only one CPU burst per process is used in examples
 - Measure of comparison is done with Average Waiting Time
 - Waiting Time is the sum of the periods spent waiting in ready queue
 - Context switch time is negligible

CIT 595

15

Scheduling Scheme: FCFS

- First Come First Served (FCFS)
 - The process that requests the CPU first is allocated the CPU
- Implemented using a FIFO queue
 - When the process enters the read queue, its PCB is linked to the tail
 - The process at the head of the queue is given to the CPU
- FCFS is non-preemptive and hence easy to implement
- Wait time varies substantially if the processes that comes first are CPU intensive

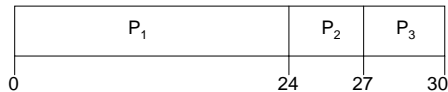
CIT 595

16

FCFS Example (Execution Time)

Process	Execution Time
P_1	24
P_2	3
P_3	3

- Suppose that the processes arrive in the order: P_1, P_2, P_3
The Gantt/Time Chart for the schedule is:



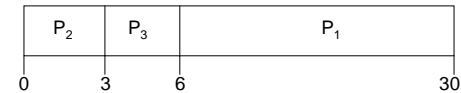
- Waiting time for $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- Average waiting time: $(0 + 24 + 27)/3 = 17$

CIT 595

17

FCFS Example (contd..)

- Suppose that the processes arrive in the order:
 P_2, P_3, P_1
- The time chart for the schedule is:



- Waiting time for $P_1 = 6$; $P_2 = 0$; $P_3 = 3$
- Average waiting time: $(6 + 0 + 3)/3 = 3$
 - Much better than previous case
- Convoy effect: all processes wait for the one big process to get off the CPU

CIT 595

18

Scheduling Algorithm: Shortest Job First (SJF)

- The process with the shortest execution time takes priority over others
- Associate with each process the length of its next CPU execution time
 - Achieved by **guessing** the run time of process for its next execution
- Non-preemptive**
 - Once CPU given to the process it cannot be preempted until completes its CPU execution time is complete.
 - If 2 short jobs with same execution time then use their time of arrival to break the tie
- Preemptive**
 - If a new process arrives with CPU execution time less than remaining time of current executing process, preempt

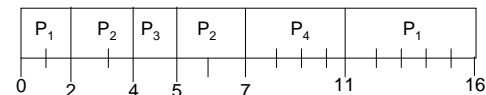
CIT 595

19

SJF with Preemption Example

Process	Arrival Time	Next CPU Ex Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (preemptive)



- Average waiting time = $(9 + 1 + 0 + 2)/4 = 3$
- Also known as Shortest Remaining Time First

CIT 595

20

SJF Advantages and Disadvantages

Advantage

- SJF is optimal – gives minimum average waiting time for a given set of processes

Disadvantage

- Need to have a good heuristic to guess the next CPU execution time
- Short duration processes will starve longer ones

Estimating the Next CPU Execution Time

- Use previous value of the execution time
- Predict new time using Exponential Averaging
- Method:
 1. t_n = actual length of n^{th} CPU Execution Time
 2. τ_{n+1} = predicted value for the next CPU Execution Time
 3. $\alpha, 0 \leq \alpha \leq 1$
 4. Formula: $\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n$.
- Since both α and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor

Scheduling Scheme: Priority

- A priority number (integer) is associated with each process
- CPU is allocated to the process with the highest priority
 - If processes have same priority then schedule according to FCFS
- SJF is an example of a priority scheduling
- Problem: Starvation – low priority processes may never execute
- Solution: Aging – as time progresses increase the priority of the process

Scheduling Scheme: Round Robin (RR)

- Each process gets a small unit of CPU time called *time quantum or slice*
 - 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
- Performance
 - If q is large then RR becomes like FCFS
 - q should not be so small such that it requires too many context switches

What do real systems use?

- Multi-level Feedback queues
 - N priority levels
 - Priority scheduling between levels
 - RR within a level
 - Quantum size decreases as priority level increase
 - Process in a given not scheduled until all higher priority queues are empty
 - If process is does complete in a given quantum at a priority level it is moved to next lower priority level
 - Aging

Next Time

- How can we create processes ?
- What is the difference between process and threads?
- Examples of process and thread creation in Unix
- Some issue with concurrency