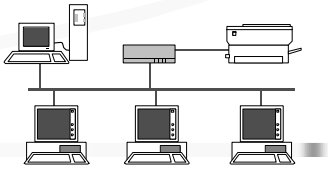


Operating Systems
600.418

Process Control & Scheduling

Department of Computer Science
The Johns Hopkins University

Process Control & Scheduling

Lecture 2

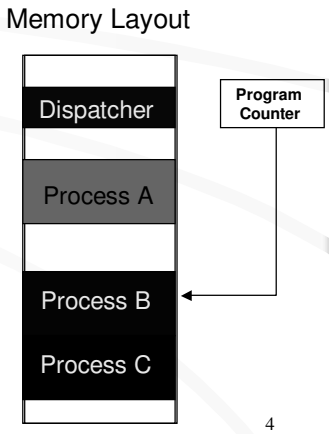
Reading:
Silberschatz & Galvin
chapters 4, 5
Additional Reading:
Stallings
chapters 3,4,9

Process

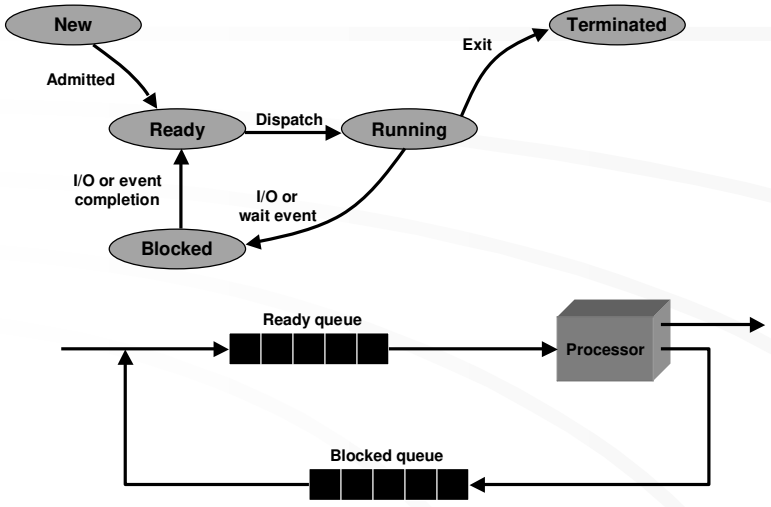
- Classical
  - A program in execution
  - A container for resources
    - File descriptors
    - Memory mappings
    - Access rights
  - Unit of Scheduling
  - A set of registers
- Current:
  - A container for resources
  - A container for *threads*
    - Unit of Scheduling
    - Set of registers

The Process

- Process - a program in execution.
- (Also called Task, Job).
- Several processes share the computer simultaneously.
- Reasons:
  - Better utilization.
  - Convenience.



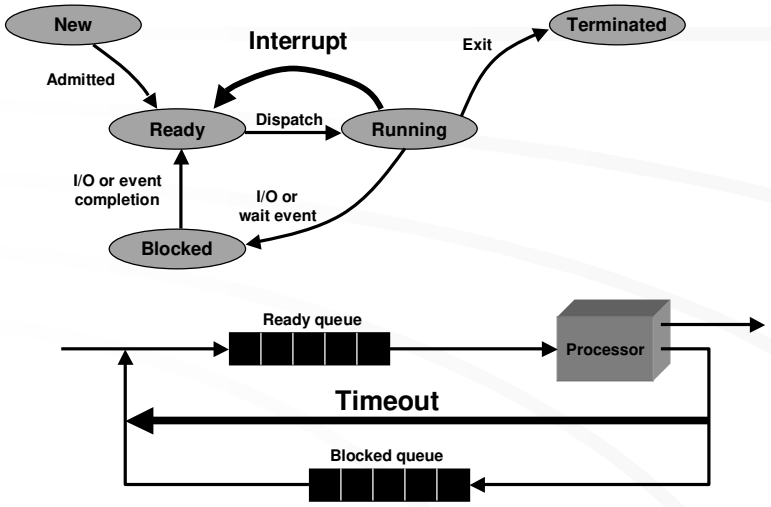
Process State Transition Diagram



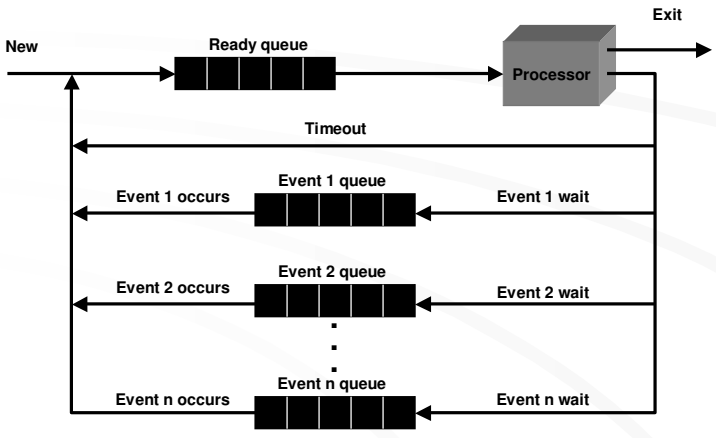
Scheduling Timeouts

- Goal: give each process a certain amount of time to run before running the next process.
- Solution: hardware timer and “timer interrupt”

Process State Transition Diagram



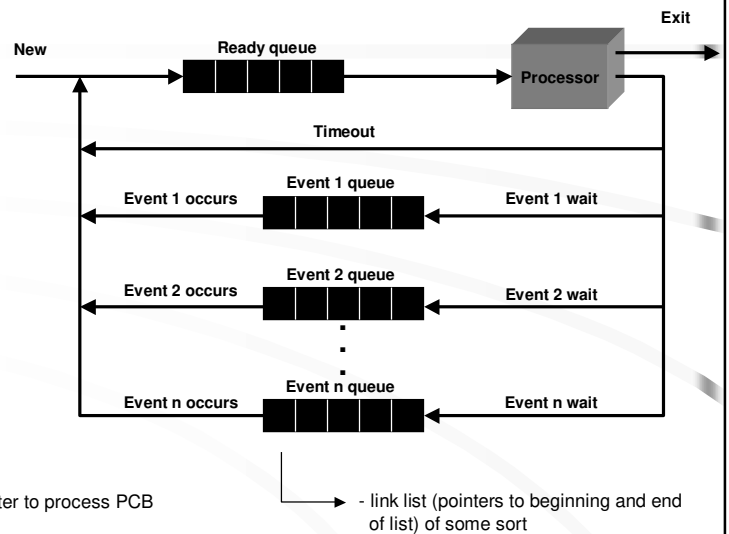
Multiple Blocked Queues



Process Control Block (PCB)

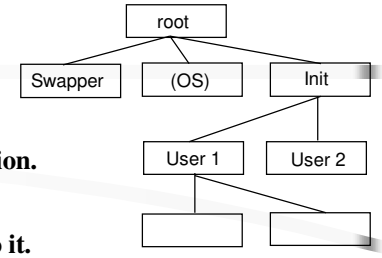
- A structure in the operating system representing the process. It includes:
  - **Process id.**
  - **Process State (state diagram)**
  - **Registers (and program counter).**
  - **Memory info.**
  - **List of open files, Inter process communication info.**
  - **Accounting info.**
  - **Pointers to other data structures in the OS.**
- Process table: logically contains a PCB for all of the current processes in the system.

Data Structures (again)...



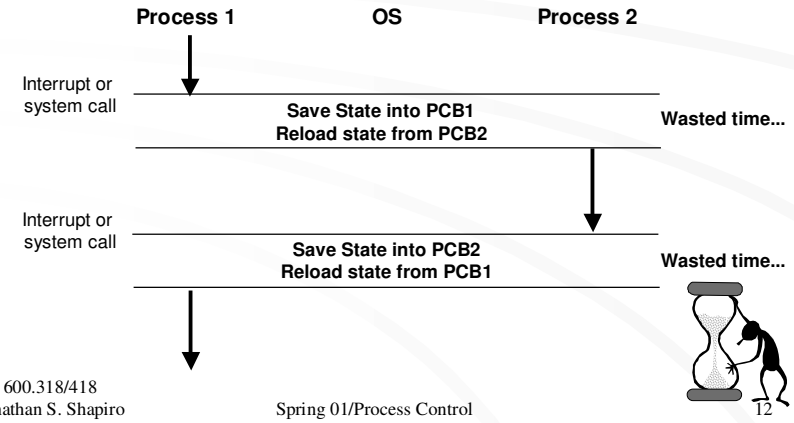
Operations of Processes

- Process creation
  - **Execution:**
    - **Parent executes concurrently.**
    - **Parent waits for child's completion.**
  - **Address space:**
    - **Child has a program loaded into it.**
    - **Child is a duplicate of the parent process.**
- Process termination
  - **The process completed its operation.**
  - **The process' parent is exiting and OS does not allow child processes to continue in that case**
  - **The process exceeded its limit of some resource.**



Context Switch

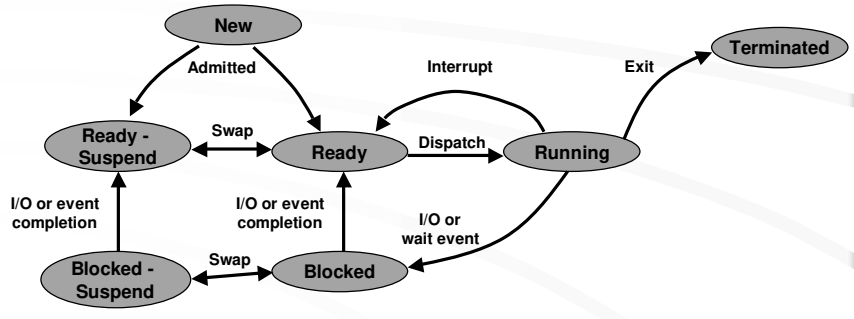
Switching the CPU to another process requires saving the state of the old process and loading the saved state for the new process.



Different Process Scheduling

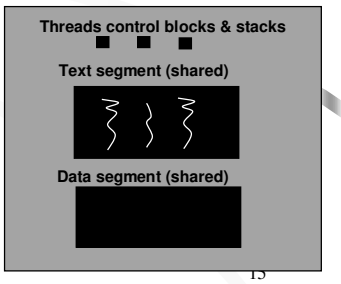
- Short Term Scheduling:
  - **Every context switch.**
  - **Which process to pick now (!!!).**
  - **Has to be very very quick.**
- Medium Term Scheduling:
  - **Memory management issues**
    - **Better to temporarily take some processes out of the main memory to disk.**
- Long Term Scheduling:
  - **Whether to admit the process to the system.**
    - **Mainframe settings.**
    - **Too many process in the system / for a user.**

State Diagram with Medium Term Scheduling



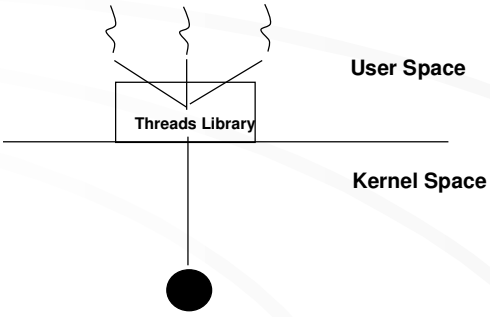
Threads

- A thread (or lightweight process) is a basic unit of CPU utilization that has a **program counter**, a **register set**, and a **stack space**.
- With the *"right"* support, a process could have several threads *concurrently* executing in it, all of them sharing the same **address space**.
- Reasons:
  - convenience
  - Potential Performance



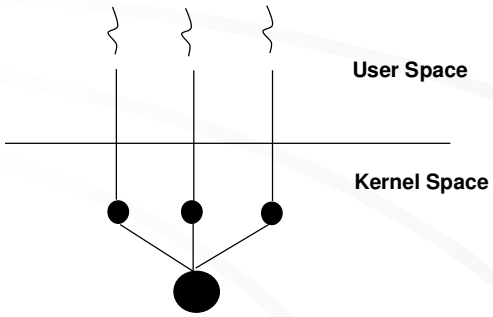
User Level Threads

- Implemented as a library
- No knowledge at the operating system level.



System Level Threads

- Implemented as part of the operating system kernel.
- Similar system calls interface for threads.



Comparison

- User level threads:
  - More efficient (order or magnitude) - almost like calling a procedure.
  - The user can better control the scheduling algorithm.
  - However, user does not know that the whole process was “out”...
- Kernel level threads:
  - Blocking I/O of one thread does not affect another thread.
  - Thread scheduling is fairer.

Processes or Threads ?

Assuming you need several threads/processes to cooperate

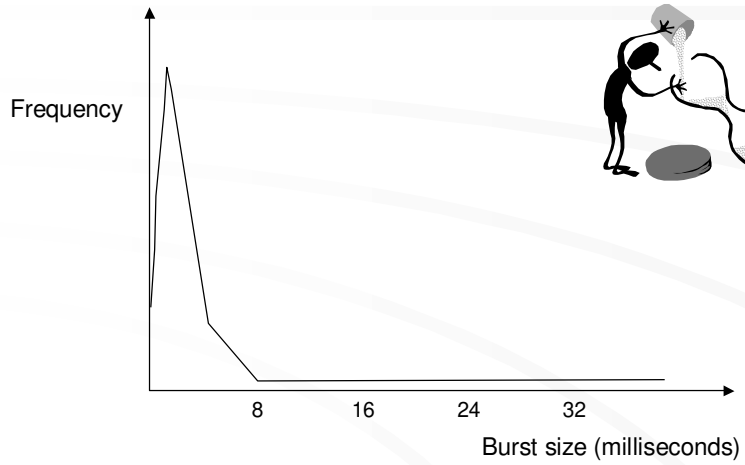
- Everything can be done using processes:
  - **Shared memory.**
  - **IPC.**
  - **Non blocking I/O.**
- A full context switch is more expensive compared with a thread switch (another order of magnitude).
- Better protection with processes (different address space).

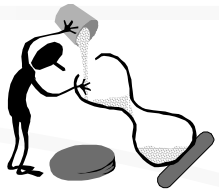
Short Term Scheduling

Criteria & Optimization:

- Max CPU Utilization.
- Max throughput - (# of completed process per unit of time).
- Min turnaround time - (actually, average slowdown).
- Min waiting time - (average time a process has been waiting in the ready queue).
- Min response time - (actually min avg response or min max response).

CPU-Burst Time Distribution



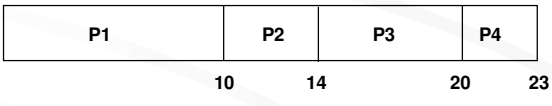


First-come, First-Served (FCFS)

- Schedule the processes according to their arrival order (to the ready queue)
  - **Also called FIFO (first in, first out).**
- Example:

Process:	**p1**	**p2**	**p3**	**p4**
Burst time:	**10**	**4**	**6**	**3**

- Schedule:



Waiting time: **0** **10** **14** **20**

Average waiting time: 11 **(assuming they all came almost together)**

First-come, First-Served (cont.)

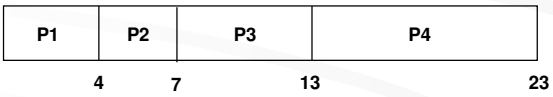
- Another example:

  Same processes, different order.

  Process: p1 p2 p3 p4

  Burst time: 4 3 6 10

- Schedule:



Waiting time: 0 4 7 13
Average waiting time: 6.

Question: What is the best arrangement for these processes (minimizing average waiting time?)

Shortest Job First

- Associate with each process the length of its next CPU burst.
  - **Non-preemptive: once CPU is given the process, it cannot be preempted until it completes this CPU burst.**
  - **Preemptive: if a new process arrives with CPU burst length less than the remaining time of the currently executing process then switch them. Also known as Shortest Remaining Time First.**

So, what can you say about these methods?

Non-preemptive Shortest Job First

- Example:

Process:	p1	p2	p3	p4
Arrival time:	0	2	4	5
Burst time:	7	4	1	4

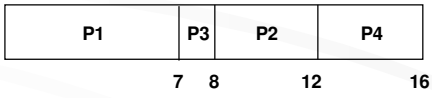
- Schedule: ?

Non-preemptive Shortest Job First

- Example:

Process:	p1	p2	p3	p4
Arrival time:	0	2	4	5
Burst time:	7	4	1	4

- Schedule:



Waiting time:	0	6	3	7

Average waiting time: 4.

Preemptive Shortest Job First

- Example:

Process:	**p1**	**p2**	**p3**	**p4**
Arrival time:	**0**	**2**	**4**	**5**
Burst time:	**7**	**4**	**1**	**4**

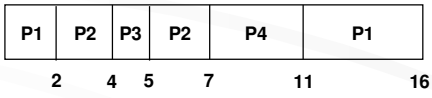
- Schedule: ?

Preemptive Shortest Job First

- Example:

Process:	**p1**	**p2**	**p3**	**p4**
Arrival time:	**0**	**2**	**4**	**5**
Burst time:	**7**	**4**	**1**	**4**

- Schedule:



Waiting time: 9 1 0 2
Average waiting time: 3.

Problem: How to Determine the Length of the Next CPU Burst?

- **No accurate way - can only estimate.**
- **Estimation: using the length of previous CPU bursts of the same process, using exponential decaying function.**

Problem: How to Determine the Length of the Next CPU Burst?

- **No accurate way - can only estimate.**
- **Estimation: using the length of previous CPU bursts of the same process, using exponential decaying function.**

Priority Scheduling

- Each process has a priority number.
- The process with the highest priority is scheduled:
  - **Non-preemptive.**
  - **Preemptive.**
- Shortest Job First is a priority scheduling. How is that?

Priority Scheduling

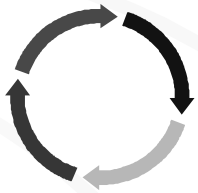
- Each process has a priority number.
- The process with the highest priority is scheduled:
  - **Non-preemptive.**
  - **Preemptive.**
- Shortest Job First is a priority scheduling where the priority is the predicted next CPU burst time.
- What is the problem with priority scheduling in general?

Priority Scheduling

- Each process has a priority number.
- The process with the highest priority is scheduled:
  - **Non-preemptive.**
  - **Preemptive.**
- Shortest Job First is a priority scheduling where the priority is the predicted next CPU burst time.
- Problem: Starvation of low priority processes.
- Solution: Aging - as time progresses, increase the priority of a process

Round Robin

- Each process gets a CPU **time quantum**, usually 10-100 milliseconds. After that, it 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*** units at once. No process waits more that **(*n*-1)*q*** time units to be scheduled.
- Performance:
  - **Large *q* => FCFS.**
  - **Small *q* => Context switch overhead!**



Round Robin (cont.)

- Example:

Process:	**p1**	**p2**	**p3**	**p4**
Required time:	**6**	**3**	**1**	**7**

- Schedule: ?

Round Robin (cont.)

- Example:

Process:	**p1**	**p2**	**p3**	**p4**
Required time:	**6**	**3**	**1**	**7**

- Schedule: It depends! (on the quantum size)

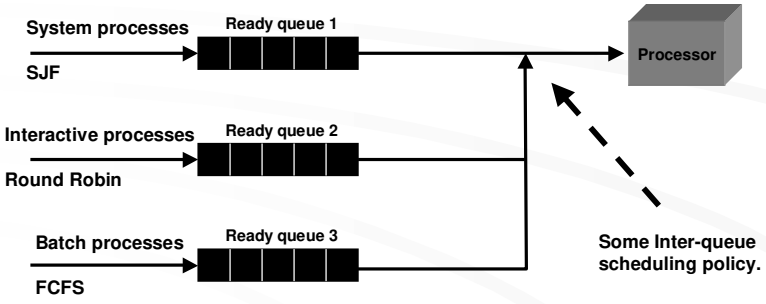
Quantum size:	**1**	**2**	**3**	**4**	**5**	**6+**
Average turnaround time:	**11**	**11.5**	**10.75**	**11.5**	**12.25**	**10.5**

Question: In general, what would be a good quantum
Hint: think of context switch overhead?

Multilevel Queue

- Ready queue is partitioned into separate queues. For example:
  - **Foreground (interactive)**
  - **Background (batch)**
- Each queue has its own scheduling policy.
- Inter-queue scheduling. For example:
  - **Fixed priority scheduling (starvation?).**
  - **Time slice between the queues. For example:**
    - 80% to interactive, 20% to batch.
    - Think about interference.

Multilevel Queue (Example)



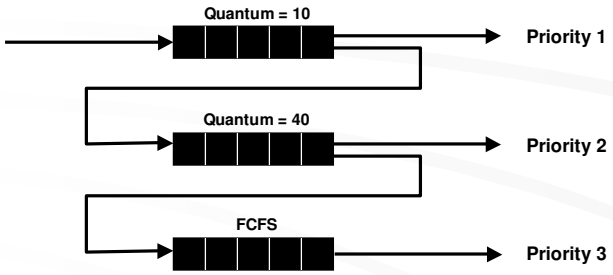
Multilevel Feedback Queue

- A process can move between the various queues. A way to implement aging.
- Multilevel feedback queue scheduler is defined as follows:
  - **number of queues.**
  - **scheduling policy for each queue.**
  - **method used to upgrade a process.**
  - **method used to degrade a process.**
  - **method used to introduce a process (which queue)**
  - **inter-scheduling between the queues (usually strict priority).**

Multilevel Feedback Queue (cont.)

- Example.
  - **Three queues:**
    - Q1 - time quantum is 10 milliseconds, FCFS.
    - Q2 - time quantum is 40 milliseconds, FCFS.
    - Q3 - First Come First Serve.
  - **Scheduling:**
    - A new process enters Q1. If it does not complete in first quantum, it is degraded to Q2. If it does not complete in first quantum, it is degraded to Q3.
    - Inter queue scheduling is strict priority.

Multilevel Feedback Queue (cont.)



How to Evaluate Scheduling Algorithms?

- Deterministic modeling (as we did today):
  - **predefined workload and a defined criterion.**
- Queuing models:
  - **predefined arrival and service processes and a defined criterion.**
- Simulation:
  - **Simulation assumptions.**
- Implementation.