Process Scheduling

Dickinson College
Computer Science 354
Spring 2006

slides courtesy of Professor Grant Braught

Road Map

- Past:
  - What an OS is, why we have them, what they do.
  - Base hardware and support for operating systems
  - Process Management
- Present:
  - Process Scheduling
- Future:
  - Concurrent programming
  - Memory management
  - Storage management
  - Protection and Security

Process Scheduling

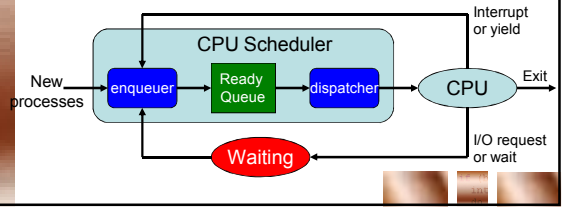
- Outline
  - Types of scheduling
  - Context switching
  - Performance metrics
  - CPU Scheduling algorithms
  - Real OS Examples
  - Project #2

Types of Scheduling

- There are three types of scheduling that occur on different time scales:
  - CPU scheduling
    - a.k.a. short-term scheduling or just scheduling
  - Job scheduling
    - a.k.a. Long-term scheduling
  - Medium-term scheduling
    - Not every OS uses all three types.

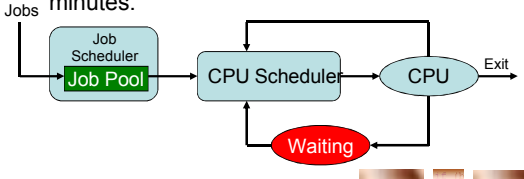
CPU Scheduling

- The short-term (CPU scheduler or *scheduler*) selects, from the ready queue, the next process that will run on the CPU.
  - The CPU scheduler may run 10-100 times per second.



Job Scheduling

- A job scheduler selects jobs (i.e. processes) from a pool of new, but as yet unexecuted, processes to add to the ready queue.
  - Job scheduling may only occur every several minutes.

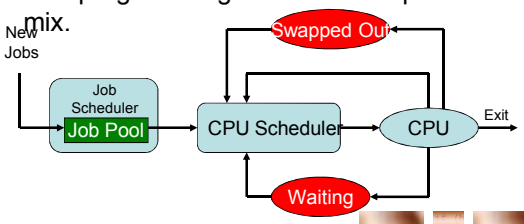


Job Scheduling Criterion

- The job scheduler determines the degree of multiprogramming in the system.
  - The job scheduler attempts to maintain a balance of *CPU Bound* processes and *I/O Bound* processes in the system.

Medium-term Scheduling

- The medium-term scheduler *swaps* partially executed processes in and out to dynamically adjust the degree of multiprogramming or to alter the process mix.



Random OS Cartoon

Context Switching

- Any time the CPU is switched from running one process to running another process, a *context switch* must occur.
  - A context switch is a two part process:
    - The context of the running process must be saved.
    - The context of the next process to run must be restored.

Saving a Process' Context

- A running process' context is saved immediately following any interrupt or system call.
  - Saving the context requires backing up any values that may/will be overwritten by another process.
  - Values are backed up by copying them to the process' PCB.
    - Some of the values that are copied include:
      - The value of the PC just before the interrupt or system call
      - The current values in the SP and GP registers.
      - Address space information (base/limit registers)

Restoring a Process' Context

- Restoring a context:
  - A process' context is restored just before control of the CPU is given to that process.
    - The values of the SP and GP registers are copied from the PCB back into the registers.
    - The machine is switched to user mode.
    - The PC is set to the value from the PCB.

Speeding up Context Switches

- Context switching takes time.
- Some techniques that have been used to speedup context switching include:
  - Partial context switches
  - Kernel register set
  - Multiple register sets
  - Machine language instruction
  - Threads

CPU Scheduling

- In designing the CPU scheduler there are two major design questions that must be answered:
  - Under what circumstances will the scheduler be invoked?
    - Non-preemptive vs. Preemptive scheduling
  - When the scheduler is invoked, what criterion will it use to select, from the ready queue, the next process to run?
    - Scheduling Algorithms

When Should the Scheduler Run?

- There are four circumstances under which the scheduler can be used to select a new process to run:
  1. The running process *blocks*. (running→waiting)
  2. A new process is created. (new → ready)
  3. The running process is interrupted. (running → ready)
     A process may also unblock. (waiting → ready)
  4. A process exits. (running→terminated)

Non-Preemptive Scheduling

- With non-preemptive scheduling a process that is in the running state will remain in the running state until it:
  - Terminates
  - Makes a *blocking system call*
  - So, in a non-preemptive scheduler, scheduling occurs under circumstances #1 and #4.
    - Note: Some systems allow a process to voluntarily yield the CPU. This action would also require the scheduler to choose a new process to run.

Preemptive Scheduling

- With preemptive scheduling, the scheduler will run under all of the circumstances (#1-#4).
  - Preemptive scheduling enables a number of things that non-preemptive scheduling cannot:
    - Time sharing
    - Priority scheduling

Real World Scheduling Analogies

- Which type of scheduling (preemptive / non-preemptive) occurs in the following settings?
  - Restaurant
  - Hospital emergency room
  - Professor's office hours

Random OS Quote

- The two main design principles of the NeXT machine appear to be revenge and spite.

  Don Lancaster

Scheduling Metrics

- There are a number of metrics that are commonly used to evaluate the performance of a scheduling algorithm:
  - ✓CPU Utilization
  - ✓Throughput
  - ✓Turnaround Time
  - ✓Wait Time
  - ✓Waiting Time
  - ✓Response Time

CPU Utilization

- CPU Utilization is the percentage of time that the CPU spends executing code on behalf of the users.
  - ✓Running user code
  - ✓Processing system calls
  - ✓Handling interrupts that signal completion of a requested operation.

Throughput / Turnaround Time

- Throughput is the average number of processes completed per time unit.
  - ✓E.g. 10 jobs / minute
- Turnaround time is the total time from when a process first enters the ready state to the last time it leaves the running state.
  - ✓Typically averaged across a number of jobs.

Wait Time / Waiting Time

- Wait time is the time a process spends in the ready queue before its first transition to the running state.
- Waiting time is the total time that a process spends in the ready queue during its entire execution.
  - ✓Both of these are typically reported as an average across a number of jobs.

Response Time

- Response time is the average length of a visit to the ready queue.

CPU Scheduling Algorithms

Scheduling Algorithms

- Basic Strategies:
  - First-Come-First-Served (FCFS)
  - Shortest Job Next (SJN)
  - Priority
  - Round Robin (RR)
- Combined Strategies:
  - Multi-level Queues
  - Multi-level Feedback Queues

Comparing Scheduling Algorithms

- There are a variety of techniques for evaluating and comparing the performance of different scheduling algorithms:
  - Deterministic Modeling
  - Simulation
  - Implementation
  - Theoretical approaches (e.g. Queuing models)

Modeling Processes

- In order to use deterministic modeling or simulation it is necessary to model of how processes behave.
  - Typically processes alternate between *bursts* of CPU operations and blocking I/O requests.

Process 1:	Process 2:
CPU 10	CPU 100
I/O Disk1 200	I/O Disk1 100
CPU 20	CPU 100
I/O Disk1 150	
CPU 10	

Deterministic Modeling

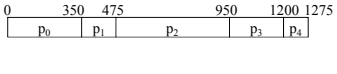
- For deterministic modeling we will make the simplifying assumption that every process consists of only a single burst of CPU activity.
  - This allows the operation of the scheduler to be modeled more easily by hand.

Arrival Order	CPU Burst
0	350
1	125
2	475
3	250
4	75

FCFS Example

Arrival Order	CPU Burst
0	350
1	125
2	475
3	250
4	75

Gantt Chart:



$$Throughput = \frac{5\ jobs}{1275\ tu} = 0.004\ \frac{jobs}{tu}$$

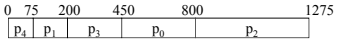
$$Turnaround = \frac{(350 + 475 + 950 + 1200 + 1275)\ tu}{5\ jobs} = 850\ \frac{tu}{job}$$

$$Wait = \frac{(0 + 350 + 475 + 900 + 1200)\ tu}{5\ jobs} = 595\ \frac{tu}{job}$$

SJN Example

Arrival Order	CPU Burst
0	350
1	125
2	475
3	250
4	75

Gantt Chart:



$$Wait = \frac{(0+75+200+450+800)\ tu}{5\ jobs} = 305\ \frac{tu}{job}$$

$$Turnaround = \frac{(75+200+450+800+1275)\ tu}{5\ jobs} = 560\ \frac{tu}{job}$$

- SJN guarantees minimum average wait time.

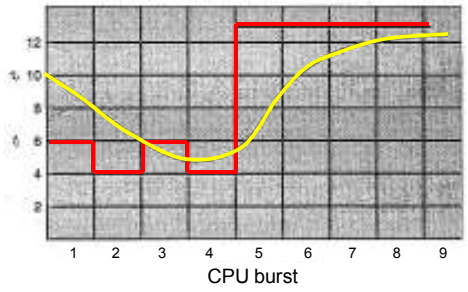
Shortest Next CPU Burst Next

- The duration of past CPU bursts can be used as a predictor of the duration of the next CPU burst.
  - One approach uses an *exponential average*:

$$\tau_{n+1} = \alpha \cdot t_n + (1-\alpha)\tau_n$$

  - t_n = actual length of n^{th} (previous) CPU burst.
  - τ_n = predicted length of n^{th} CPU burst.
  - τ_{n+1} = predicted length of next CPU burst.
  - α = history scaling factor

Shortest Next CPU Burst Next



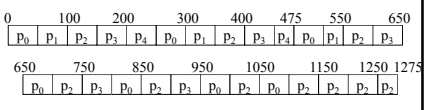
Priority Scheduling

- With priority scheduling every process is assigned a priority value.
  - At each scheduling opportunity, the process with the highest priority is selected to run.
    - Priority scheduling can result in *starvation*.
    - Dynamic priorities and *aging* can be used to combat starvation.

RR Scheduling (w/ 50tu Time Slice)

Arrival Order	CPU Burst
0	350
1	125
2	475
3	250
4	75

Gantt Chart:



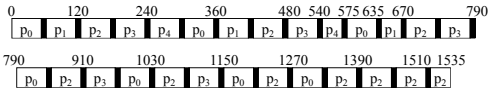
$$Wait = \frac{(0+50+100+150+200)\ tu}{5\ jobs} = 100\ \frac{tu}{job}$$

$$Waiting_{p0} = (0+200+175+125+100+100+50)\ tu = 750\ tu$$

$$\text{Response} = \frac{Waiting_{p0} + \ldots + Waiting_{p4}}{total\ visits\ to\ ready\ queue}$$

RR Scheduling (w/ 10 tu scheduling overhead)

Gantt Chart:

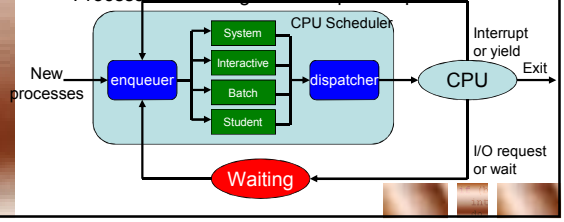


$$CPU = \frac{(350+125+475+250+75)\ tu}{1535\ tu} \times 100 = \frac{1275}{1535} \times 100 = 83\%$$

- Throughput, turnaround, wait, waiting and response time calculations must now also include the overhead.

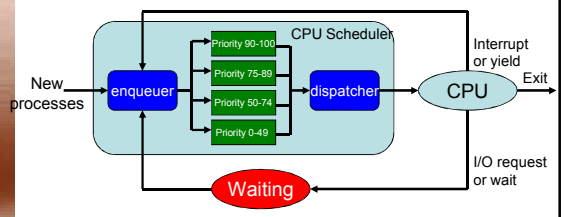
Multi-level Queues

- The ready queue is divided into multiple levels.
  - The queue for each level has a different priority.
    - Any job in higher priority queue will run before any job in lower priority queue.
  - Each queue may use its own internal scheduling algorithm.
  - Processes are assigned to a specific queue.



Multi-level Feedback Queues

- Similar to multi-level queues:
  - Each new process is assigned a priority.
  - The OS adjusts the priority of each process as it runs.
    - Thus, processes move among the different queues as they run.



Random OS Quote

- "The Internet? We are not interested in it"

  Bill Gates, 1993

- "Sometimes we do get taken by surprise. For example, when the Internet came along, we had it as a fifth or sixth priority."

  Bill Gates, 1998

Examples of Real OS CPU Scheduling Algorithms

Linux Scheduling (pre. V2.5)

- Early Linux kernels used preemptive scheduling with a multi-level queue with three levels:
  - FIFO: Highest priority level. Used for short, time-critical system threads.
  - RR: Medium priority. Used for longer running system threads.
  - OTHER: Lowest priority. Used for all user threads.
    - Internally, the OTHER queue uses a dynamic priority scheduling scheme.

Linux OTHER Queue Scheduling

- Each thread, i, has a number of credits, p_i
  - New threads are given a default number of credits, K.
- The system timer is used to create fixed size time slices.
  - On each timer interrupt the credits of the running thread is decremented
  - If a thread's credits reach 0, it is blocked.
- Scheduling is preemptive:
  - In each time slice, the thread in the ready queue that has the most credits is selected to run.
    - If no threads are ready (i.e. all threads are blocked), then a *recrediting* operation is performed.

Linux OTHER Recrediting

- If there are no threads ready to run recrediting occurs:
  - ✓ During recrediting, *every thread* in the system is assigned credits using the following formula:

$$p_i = \frac{p_i}{2} + K$$

  - ✓ Threads that were blocked because they had 0 credits now return to the ready queue with K credits.

Solaris

- Solaris uses preemptive scheduling with a multi-level queue with four levels.
  - ✓ The levels in order of decreasing priority are:
    - ❖ Real time: Provides guaranteed bound on response time.
    - ❖ System: Kernel threads
    - ❖ Interactive: User threads that are running in a windowing environment.
    - ❖ Time Sharing: Non-interactive user threads.

Solaris Interactive and Time Sharing Scheduling

- Priority within the interactive and time sharing queues is based on a *dispatch table*.

Priority	Time Slice	Time Slice Expired	Return from Blocked
0	200	0	50
5	200	0	50
10	160	0	51
15	160	5	51
20	120	10	52
25	120	15	52
30	80	20	53
35	80	25	54
40	40	30	55
45	40	35	56
50	40	40	58
55	40	45	58
59	20	49	59

Windows XP Scheduling

- Windows XP uses preemptive scheduling with a multi-level queue with six levels.
  - ✓ Within each level, priority scheduling with seven relative priorities is used.
    - ❖ Priorities over 15 are fixed. Priorities less than 15 are dynamic and are adjusted based on process behavior.

	Real Time	High	Above Normal	Normal	Below Normal	Idle
Time Critical	31	15	15	15	15	15
Highest	26	15	12	10	8	6
Above Normal	25	14	11	9	7	5
Normal	24	13	10	8	6	4
Below Normal	23	12	9	7	5	3
Lowest	22	11	8	5	4	2
Idle	16	1	1	1	1	1

Base Priority

Fixed Priority

User Processes

Windows XP Scheduling

- Windows XP scheduling does two unique things:
  - ✓ Priority Adjustments:
    - ❖ Like other schedulers:
      - Relative priority of thread using up its time slice is decreased.
      - Relative priority of thread returning from blocked is increased.
        - However, in XP the amount of the increase depends on why the thread was blocked.
  - ✓ Within the Normal priority class, threads in the foreground process get a longer time slice (3x) than other threads.

Random OS Quote

- Here's a few of 11 rules that Bill Gates gave to high school kids regarding stuff they won't learn in school.
  - ✓ Rule 1: Life is not fair...get used to it.
  - ✓ Rule 4: If you think your teacher is tough, wait till you get a boss. He doesn't have tenure.
  - ✓ Rule 11: Be nice to nerds. Chances are you'll end up working for one.

Project #2
CPU Scheduling Simulation

Scheduling Simulations

- Scheduling simulations account for several important factors that are frequently ignored by deterministic modeling:
  - ✓Scheduling overhead
  - ✓I/O Operations
  - ✓Process Profiles
    - ❖CPU Bound vs. I/O Bound

Project #2

- In project #2 you will extend a provided scheduling simulator by implementing and testing a round robin scheduler.

Using the Simulator

- Source code for the simulator is provided on the project web-page.
- To run the simulator:
  - ✓ javac *.java
  - ✓ java SystemDriver <procFile> <devFile> <sched>
    - ❖ <procFile>: File holding a list of files describing the processes to be scheduled.
    - ❖ <defFile>: File holding a list of the available I/O devices.
    - ❖ <sched>: The name of the scheduling algorithm to use.

Provided Example

- The simulator comes with an example that can be run with the following command line:

```
java SystemDriver processes.dat devices.dat FCFSnoIO
```

  - ✓Sample Results:

```
System Time: 64
Kernel Time: 12
User Time: 45
Idle Time: 7
CPU Utilization: 70.31%
```

Configuration Files

- processes.dat

```
# These three processes have
# a variety of arrival times
# and CPU burst length.
# The processes do not
# perform any I/O.

process1.dat
process2.dat
process3.dat
```

- devices.dat

```
# List of all of the I/O
# devices that the system
# supports.  You may add or
# delete devices at will.
# However, processes may
# only use the devices
# specified in this file.

# List of I/O Devices.
I/O DISK1
I/O DISK2
I/O CDROM
```

Process Description Files

```
# Process name: must be unique among all
# processes for any given simulation.
PROCESS5

# Process arrival time:
15

# Process profile. This must begin with
# START and end with EXIT. Also every process
# must start and end with a CPU burst. That is,
# the first line after START and the last line
# before EXIT must be CPU bursts. In between
# the lines may be: # CPU <time>
# IO <device> <time>
START
CPU 25
IO DISK1 100
CPU 15
IO CDROM 50
CPU 25
EXIT
```

Provided Sample Processes

- process1.dat:

```
PROCESS1
0
START
CPU 20
EXIT
```

- process2.dat:

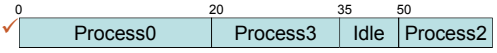
```
PROCESS2
50
START
CPU 10
EXIT
```

- process3.dat:

```
PROCESS3
10
START
CPU 15
EXIT
```

Sample Scheduler (FCFSnoIO)

- The FCFSnoIO scheduler provided with the simulator schedules processes using FCFC.
  - The processes may not perform any I/O because this scheduler does not provide the ability to block processes that are waiting on I/O.



Understanding the Sample Results

- Results:

```
System Time: 64
Kernel Time: 12
User Time: 45
Idle Time: 7
CPU Utilization: 70.31%
```

  - Overhead for each system call is 2 time units.
    - One system call per process at arrival time.
    - One system call per process at exit time.
  - Interrupt handling also adds 2 time units of overhead.

Code Tour of FCFSnoIO

- Important points:
  - Kernel interface
    - No-arg constructor
    - systemCall method
      - callID
        - START_PROCESS
        - TERMINATE_PROCESS
        - IO_REQUEST
        - MAKE_DEVICE
      - SystemTimer object
    - interrupt method
      - deviceID
    - running method
    - terminate method

What Needs to be Done?

- You need to:
  - Add handling of devices and I/O requests.
  - Implement round robin scheduling with time slicing.
  - Compute relevant statistics:
    - Throughput, turnaround time, wait time, waiting time, response time.