Lecture 5: The Internal Operating System

Multitasking fundamentals
Scheduling
Memory management

Multitasking Fundamentals

- Multiple programs are each sharing resources that are designed to do one thing at a time.

- Individual resource managers:
  - CPU
  - Memory
  - I/O devices

Process (and Threads)

- A process is defined to include a program together with all the resources that are associated with that program as it is executed.

- Independent vs. cooperating processes

- User vs. system processes

Process and Program

User 1

Process 1

Executing instruction here

Editor program

Process 1 data

User 2

Process 2

Executing instruction here

Process 2 data
Process Control Block (PCB)

Information associated with each process:
- Process state
  - New, ready, running, waiting, ...
- Program counter
  - Address of the next instruction to be executed
- CPU registers
  - Accumulators, index registers, stack pointers, general-purpose registers
- CPU scheduling information
  - Process priorities, pointers to scheduling queues, etc.

Process Control Block (Cont.)

- Memory management information
  - Base and limit registers
- Accounting information
- I/O status information
  - List of I/O devices allocated to process, open files, etc.

PCB is sometimes denoted as task control block.

Process Control Block (Cont.)

<table>
<thead>
<tr>
<th>process state</th>
<th>process number</th>
<th>program counter</th>
</tr>
</thead>
<tbody>
<tr>
<td>registers</td>
<td>memory limits</td>
<td>list of open files</td>
</tr>
</tbody>
</table>

Process States

- new
- admitted
- interrupted
- exit
- terminated

- ready
- running
- I/O or event completion
- scheduler dispatch
- I/O or event wait
- waiting
**Context Switch**

- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process.

- Context-switch time is overhead; the system does no useful work while switching.

- Time dependent on hardware support (from 1µs to 1ms).
  - *e.g.*, consider an architecture having multiple sets of registers

**CPU Switch From Process to Process**

**CPU Scheduling**

- Why scheduling?
- For using resources efficiently
  
  - I/O is very much slower than the CPU (CPUs run at billions of instructions per second, hard disk and network accesses take milliseconds)
  - When a process makes a I/O request, it has to wait. In this time, the CPU would idle if the OS did not schedule a ready process on it.

**(Process and) Threads**

- A *thread* represents a piece of a process that can be executed independently of other parts of the process.
  
  - Share the same resources but allow concurrent operations
  - Easier context switching
  - Support for event-driven programs
Schedulers

- Long-term scheduler (or job scheduler)
  - Selects which processes should be brought into the ready queue.

- Short-term scheduler (or CPU scheduler, or dispatcher)
  - Selects which process should be executed next and allocates CPU.

- Medium-term scheduler
  - Controls swapping:
    - Removes processes from memory (swap out), and reloads them later (swap in)

Schedulers (Cont.)

- Short-term scheduler is invoked very frequently (milliseconds) \(\Rightarrow\) (must be fast)

- Long-term scheduler is invoked very infrequently (seconds, minutes) \(\Rightarrow\) (may be slow)

- The long-term scheduler controls the degree of multiprogramming

Scheduling Algorithms

Dispatching objectives:
- Ensure fairness
- Maximize throughput
- Minimize turnaround time
- Maximize CPU utilization
- Maximize resource allocation
- Promote graceful degradation
- Minimize response time
- Provide consistent response time
- Prevent starvation

Scheduling Algorithms (Cont.)

```plaintext
Processes 0 1 2 3 4 5 6 7
P1 P2 P3
```

CPU1
```
0 1 2 3 4 5 6 7
P1 P2 P3
```

OR ?
Scheduling Algorithms (Cont.)

- Non-preemptive
  - First-In, First-Out (FIFO), or First-Come, First Served (FCFS)
  - Shortest Job First (SJF)
  - Priority scheduling

- Preemptive
  - Round Robin
  - Multilevel feedback queues
  - Priority scheduling

First-Come, First-Served (FCFS) Scheduling

- Example:
  - Process Burst Time (ms)
    - $P_1$ 24
    - $P_2$ 3
    - $P_3$ 3

  Suppose that the processes arrive in the order: $P_1$, $P_2$, $P_3$.
  The Gantt chart for the schedule is:

  

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

  Much better than previous case
  Convoy effect: short process behind long process

FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order
$P_2$, $P_3$, $P_1$.

- The Gantt chart for the schedule is:

  

  - Waiting time for $P_1 = 6$ms; $P_2 = 0$ms, $P_3 = 3$ms
  - Average waiting time: $(6 + 0 + 3)/3 = 3$ms
  - Much better than previous case
  - Convoy effect: short process behind long process

SJF (nonpreemptive)

- Process Burst Time (ms)
  - $P_1$ 6
  - $P_2$ 8
  - $P_3$ 7
  - $P_4$ 3

Average waiting time = $(3 + 16 + 9 + 0)/4 = 7$ms

Problem: you have to know in advance the burst time!
### Round Robin (Preemptive)

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁</td>
<td>6</td>
</tr>
<tr>
<td>P₂</td>
<td>8</td>
</tr>
<tr>
<td>P₃</td>
<td>7</td>
</tr>
<tr>
<td>P₄</td>
<td>3</td>
</tr>
</tbody>
</table>

Round Robin Quanta: 3ms

### Non-Preemptive vs. Preemptive

- Non-preemptive scheduling requires no hardware support (timer). The OS is also less complex.
- Preemptive leads to shorter response times.
- However, operations by the kernel on some data have to be performed atomically (i.e. without being preempted while in the middle of managing that data) in order to avoid data inconsistencies.
- A common Unix solution is preemptive scheduling of processes and non-preemptable system calls.

### Multilevel Feedback Queue

- A process can move between the various queues; aging can be implemented this way.
- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method used to determine when to upgrade a process
  - method used to determine when to demote a process
  - method used to determine which queue a process will enter when that process needs service

### Example of Multilevel Feedback Queue

- Three queues:
  - Q₀ – RR with time quantum 8 ms
  - Q₁ – RR time quantum 16 ms
  - Q₂ – FCFS
- Scheduling:
  - A new job enters queue Q₀ which is served RR.
    - When it gains CPU, job receives 8 ms.
    - If it does not finish in 8 ms, job is moved to queue Q₁.
  - At Q₁, job is again served RR and receives 16 additional ms.
  - If it still does not complete, it is preempted and moved to queue Q₂ served in FCFS.
Memory Management

- Scheduling: how to share (schedule) the CPU
- Memory management: how to share the memory
  - Allocate (fit) the memory to the processes
  - Maximize the memory utilization

Memory Management

- Single tasking
- Single tasking with overlays (cf. book)
- Multitasking – memory partitioning
- Virtual storage

Overlays

![Diagram of overlays](image)

Memory Partitioning (Contiguous Allocation)

- Multiple-partition allocation
  - Hole — block of available memory; holes of various size are scattered throughout memory
  - When a process arrives, it is allocated memory from a hole large enough to accommodate it
  - Operating system maintains information about:
    - allocated partitions
    - free partitions (hole)
Fragmentation

- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used

  ![Internal Fragmentation Diagram]

- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous

  ![External Fragmentation Diagram]

  - Reduce external fragmentation by compaction
  - Shuffle memory contents to place all free memory together in one large block
  - Compaction is possible only if relocation is dynamic, and is done at execution time
  - I/O problem
    - Latch job in memory while it is involved in I/O
    - Do I/O only into OS buffers

Example of Compacting (External Fragmentation)

![Example of Compacting Diagram]

Example of Compacting: Solution 1

![Example of Compacting Solution Diagram]
Example of Compacting: Solution 2

Main Memory

- p1
- p2
- p3
- p4

Main Memory after move of P3

- p1
- p3
- p2
- p4
- p

Memory Relocation Issues

- The final location in memory where the program is loaded is not known beforehand
- Addresses from the program have to be adjusted to correspond to the actual memory allocation
- Virtual addresses vs. Physical addresses

Binding

- Compile time binding
  - If the compiler knows where the process will reside in memory, *absolute code* can be generated, i.e. the addresses in the executable file are the physical addresses
- Load time binding
  - Otherwise, the compiler must generate *relocatable code*, i.e. code that may reside anywhere in memory
- Execution time binding
  - If the process can be moved during execution from one memory segment to another
  - Special hardware support needed (MMU)

Virtual and Physical Addresses

- MMU (memory management unit) – run-time mapping of virtual addresses to physical addresses
Paging (NonContiguous Allocation)

- Memory management scheme that permits non-contiguous allocation
- Physical memory is divided in fixed-size blocks, called frames
- Virtual address space is divided in blocks of the same size as a frame, called pages
- Backing store is also divided in frames
- When a process is to be executed, it is loaded into available memory frames of the backing store
- A page table contains the base address of each page in physical memory

Paging (Cont.)

- Frames (fixed size)
- Memory
- Pages (fixed size)

Memory

Frames

How to map process pages to the frames in the memory?

Paging Examples

- size of logical space is $2^m$
- page size is $2^n$

Does It All Fit in the Memory?

- Not on the main memory (RAM)
- Yes, if we consider the secondary memory (disk)
- The OS can do swapping
Page Fault (Virtual Memory)

1. The page table is checked for memory reference M.
2. That page is not currently in memory, which causes a page fault trap.

Page Fault (Cont.)

2. OS memory manager locates desired page on auxiliary storage and loads it into free memory frame.
3. Resets the page table, and restart the instruction.

Summary

- Multitasking
- Process / Thread
- Scheduling
  - Dispatching
  - Context switch
  - Algorithms
- Memory management

Recommended Reading and Exercises

- Reading:
  - Chapter 15 (3rd edition) / 16(2nd edition)
    - 15.0 – 15.6 – 3rd edition
    - 16.0 – 16.7 – 2nd edition
- Exercises
  - At the end of the chapter