**Lesson 0**

Operating Systems

An operating system is a program that manages the computer hardware

provides a basis for application programs

acts as an intermediary between computer-user and hardware

provides an environment within which other programs can do work

**Computer system**

|  |  |  |
| --- | --- | --- |
| Hardware | provides basic computing resources | CPU, memory, I/O devices |
| Operating system | Controls and coordinates use of hardware among various applications and users |  |
| Application programs | define the ways in which the system resources are used to solve the  computing problems of the users | Word processors, compilers, web browsers, database systems, video games |
| Users | People, machines, other computers |  |

**OS: Definition for COS3721**

The operating system is the one program running at all times on the computer - usually called the kernel.

Along with the kernel there are two other types of programs:

System programs: associated with the operating system but not part of the kernel.

Application programs: include all programs not associated with the operation of the system

**Lesson 0**

Computer System Operation

For a computer to start running it needs an initial program to run at boot time.

This initial program or bootstrap program tends to be simple.

It is stored in ROM or EEPROM and is known as firmware within the computer hardware.

It initializes all aspects of the system.

The bootstrap must know how to load the operating system. To accomplish this the bootstrap

program must locate and load the operating system kernel into memory.

Types of Events

|  |  |  |
| --- | --- | --- |
| **Interrupt**  (Asynchronous) | signaled by an interrupt from either hardware (keyboard, timer, etc.) or software | Hardware triggers an interrupt by sending a signal to the CPU. Software may trigger an interrupt by executing a special operation called a system call or monitor call. |
| **Trap**  (Synchronous) | raised by a user program (exception in a user process) | It's caused by division by zero or invalid memory access. It's also the usual way to invoke a kernel routine (a system call) because those run with a higher priority than user code. |

**Lesson 0**

Cached Data

Coherence

Single-processor system

There is typically one cache and only one process being access sequentially by the CPU. The CPU will update a file in the cache and later update it in memory reliably and consistently.

**Coherence:** Issues will only occur in a multitasking environment where the CPU has to switch between various processes. When trying to access a copy of a file in each process concurrently, the system needs to ensure that each process is accessing the most recently updated copy of the file.

Multi-processor system

Each CPU could contain its own local cache. Each CPU will update a file in the cache and later update it in memory. Copies of the same file could be stored in different caches in a multi-processor system.

**Coherence:** When trying to update a copy of the file in each CPUT concurrently, the entire system needs to communicate with all CPU’s that contains a copy of the file to ensure that all copies of the file are updated simultaneously.

Distributed system

No cache or memory is typically shared; however, several copies of the same file could be stored on different computers/nodes in a distributed system. Each node will update a file in its local cache and later update it in its local memory.

**Coherence:** When trying to update these files concurrently, the entire system needs to needs to communicate with all nodes that contain a copy of the file to ensure that all copies of the file are updated simultaneously.

**Lesson 0**

System Calls

**Fork**

Every time fork() is used, create a new level in a binary tree, with the initial process creating two child nodes. The formula for calculating the number of processes is:

Total Number of Processes =

Where is the total number of fork system calls and 1 represents the initial parent process. So, including the parent process for figure 3.32 we have:

P

/ \

P P

/ \ / \

P P P P

/ \ / \

P P P P

**Lesson 0**

Amdahl's law

Amdahl's law states that the overall performance improvement gained by optimizing a single part of a system is limited by the fraction of time that the improved part is actually used. It is best described by the speedup equation:

Where is the portion of the application that must be performed serially and is the number of processing cores.

**40 percent parallel with:**

(a) eight processing cores ()

(b) sixteen processing cores ()

**Lesson 0**

CPU Scheduling

For CPU scheduling, OS uses a number of scheduling strategies like FCFS, SJF, Round Robin, Priority scheduling, which handle the removal of running process and select the next process from remaining processes. The performance of operating system is greatly depending upon the proper CPU utilization.

**[1] Shortest Job First**

(SJF) is a CPU Scheduling algorithm that aims to predict the length of the next CPU burst. The dynamic method describes two formulas for calculating SJF, mainly:

Simple average

Exponential average (aging)

Where:

actual length of CPU burst

predicted value for the next CPU burst

**[2]** First-Come, First-Served Scheduling

(FCFS) is the simplest scheduling algorithm. The process that requests the CPU first is allocated the CPU first.

Example: Show the scheduling order of the processes using a Gantt chart

|  |  |
| --- | --- |
| Process | Burst Time |
| P1 | 24 |
| P2 | 3 |
| P3 | 3 |
|  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

0 24 27 30

If the processes arrive in order, the waiting time is 0 milliseconds for process P1, 24 milliseconds for process P2, and 27 milliseconds for process P3. The average waiting time is milliseconds.

**[3]** Round Robin

(RR) scheduling algorithm is similar to FCFS scheduling, but

preemption is added to enable the system to switch between processes. A small

unit of time, called a time quantum or time slice, is defined. A time quantum

is generally from 10 to 100 milliseconds in length. The ready queue is treated as a circular queue. The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to 1 time quantum.

To implement RR scheduling, we again treat the ready queue as a FIFO

queue of processes. New processes are added to the tail of the ready queue.

**[4]** Priority Scheduling

The SJF algorithm is a special case of the general priority-scheduling algorithm. A priority is associated with each process, and the CPU is allocated to the process with the highest priority. Equal-priority processes are scheduled in FCFS order. An SJF algorithm is simply a priority algorithm where the priority (p) is the inverse of the (predicted) next CPU burst. The larger the CPU burst, the lower the priority, and vice versa.

|  |  |  |  |
| --- | --- | --- | --- |
| Process | Burst Time | Priority | Arrival |
| P1 | 10 | 3 |  |
| P2 | 1 | 1 |  |
| P3 | 2 | 4 |  |
| P4 | 1 | 5 |  |
| P5 | 5 | 2 |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

0 1 6 16 18 19

The average waiting time is 8.2 milliseconds.

Example:

The following processes are being scheduled using a preemptive, round robin scheduling algorithm. Each process is assigned a numerical priority, with a higher number indicating a higher relative priority. In addition to the processes listed above, the system also has an idle task (which consumes no CPU resources and is identified as ). This task has priority 0 and is scheduled whenever the system has no other available processes to run. The length of a time quantum is 10 units. If a process is preempted by a higher-priority process, the preempted process is placed at the end of the queue.

a. Show the scheduling order of the processes using a Gantt chart.

b. What is the turnaround time for each process?

c. What is the waiting time for each process?

d. What is the CPU utilization rate?

a. Gantt chart

|  |  |  |  |
| --- | --- | --- | --- |
| Process | Priority | Burst Time | Arrival |
| P1 | 40 | 20 | 0 |
| P2 | 30 | 25 | 25 |
| P3 | 30 | 25 | 30 |
| P4 | 35 | 15 | 60 |
| P5 | 5 | 10 | 100 |
| P6 | 10 | 10 | 105 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

0 20 25 35 45 55 60 75 80 90 100 105 115 120

*Remember that a time quantam is 10 units, the smallest amount of work that must be done before starting another process.*

*The remainder is placed in a queue.*

b. Turnaround = end time – arrival times

P1: 20-0 = 20,

P2: 80-25 = 55,

P3: 90 - 30 = 60,

P4: 75-60 = 15,

P5: 120-100 = 20,

P6: 115-105 = 10

c. Wait time = Turnaround time - burst time

P1: 0,

P2: 40,

P3: 35,

P4: 0,

P5: 10,

P6:

d.

105/120 = 87.5 percent

**Lesson 0**

Virtualization

<https://www.cs.umd.edu/class/spring2019/cmsc412/Slides/Set19.pdf>

**Hypervisors**

Type 0 hypervisors

Implemented by Firmware. Low overhead but generally fewer features. These VMMs, which are commonly found in mainframe and large to midsized servers.

Type 1 hypervisors.

Special purpose software or general-purpose operating systems that are built to provide virtualization.

Type 2 hypervisors

Applications that run on standard operating systems but provide VMM

features to guest operating systems. More overhead and fewer features than

**Trap-and-emulate**

A CPU is able to execute instructions at two levels: user mode and kernel mode (elevated privileges). A VMM or hypervisor will have two separate modes: virtual user mode and virtual kernel mode.

If the guest attempts a privileged instruction, the hypervisor will gain control, analyse the error, execute the operation and return to the guest in user mode. This is trap-and-emulate.

The issue occurs when some CPU’s do not separate between privileged and nonprivileged instructions. Early intel x86 CPU’s are among these. In these cases, trap-and-emulate cannot be utilized. Instead, binary translation is utilized.

**Binary Translation**

Binary translation executes with the following logic:

- If guest VCPU is in user mode, guest can run instructions natively

- If guest VCPU in kernel mode (guest believes it is in kernel mode)