**Lesson 0**

Operating Systems

An **operating system** is

a) collection of programs that manages hardware resources

b) system service provider (basis) to the application programs  
c) link to interface the hardware and application programs

(Provides an environment within which other programs can do work)

d) acts as an intermediary between computer-user and hardware

Other notes:

* To access the services of the operating system, the interface is provided by the system calls
* Resource management can be done via time and space division multiplexing (**Time**-**Division Multiplexing** (TDM))
* *Sharing technique. Breaks up. Time available into stream of fixed slots*

*And distributes slots among various activities that need to be accomplished.*

* *Real time operating systems: VxWorks, Windows CE, RTLInux (NOT Palm OS)*

**Errors**

* Errors are written by the OS to a log file

Types Errors handled by the OS

a) power failure

b) lack of paper in printer

c) connection failure in the network

Types of operating systems

|  |  |  |
| --- | --- | --- |
| Type | Purpose | Example |
| uniprogramming systems | have only a single processor  Those systems which allows more than one process execution at a time, are called multiprogramming systems. | Linux |
| uniprocessing systems | contain two or more processors that share physical memory and peripheral devices. The most common multiprocessor design is symmetric multiprocessing (or SMP), where all processors are considered peers and run independently of one another  Clustered systems are a specialized form of multiprocessor systems and consist of multiple computer systems connected by a local area network | OSX |

**Lesson 0**

Process Concept

**Processes**

In operating system, each process has its own \_\_\_\_\_\_\_\_\_\_

a) address space and global variables

b) open files

c) pending alarms, signals and signal handlers

A process can be terminated due to \_\_\_\_\_\_\_\_\_\_

a) normal exit

b) fatal error

c) killed by another process

**Process State**

As a process executes, it changes state. The state of a process is defined in part by the current activity of that process. Each process may be in one of the following states:

|  |  |
| --- | --- |
| New | The process is being created |
| Running | Instructions are being executed  *if process is using the CPU, it is in running state.* |
| Waiting | The process is waiting for some event to occur (such as I/O completion or reception of a signal) |
| Ready | The process is waiting to be assigned to a processor  Waiting to be assigned to |
| Terminated | The process has finished execution |

What is interprocess communication?

communication between two processes

A set of processes is deadlock if \_\_\_\_\_\_\_\_\_\_  
a) each process is blocked and will remain so forever

8. A process stack does not contain \_\_\_\_\_\_\_\_\_\_

d) PID of child process

**Lesson 0**

Multithreaded Programming

A thread is a basic unit of CPU utilization; it comprises a thread ID, a program counter, a register set, and a stack. It shares with other threads belonging to the same process its code section, data section, and other operating-system resources, such as open files and signals.

**Thread pool**

Uses an existing thread rather than creating a new one to complete a task

**Deadlocks**

For a deadlock to arise, which of the following conditions must hold simultaneously?

- Mutual exclusion

- No preemption

- Hold and wait

Multithreaded programs are less prone to deadlocks

**Lesson 0**

Process Scheduling

In a single-processor system, only one process can run at a time;

any others must wait until the CPU is free and can be rescheduled.

The objective of multiprogramming is to have some process running at all times, in order to maximize CPU utilization.

The idea is relatively simple.

A process is executed until it must wait, typically for the completion of some I/O request.

Diagram

Description automatically generated

5.1.2 CPU Scheduler

Whenever the CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed. The selection process is carried out by the short-term scheduler (or CPU scheduler). The scheduler selects a process from the processes in memory that are ready to execute and allocates the CPU to that process.

**Lesson 0**

CPU Scheduling

For CPU scheduling, OS uses several 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.

**Starvation**

A major problem with priority scheduling algorithms is indefinite block- ing, or starvation. A process that is ready to run but waiting for the CPU can be considered blocked. A priority scheduling algorithm can leave some low- priority processes waiting indefinitely. In a heavily loaded computer system, a steady stream of higher-priority processes can prevent a low-priority process from ever getting the CPU

1. The SJF could result in starvation when shortest jobs keep on joining the ready queue which already contains a process with highest burst time.
2. RR could not result in starvation since each process ready for execution has equal chances to be executed based on the time quantum.

**Pairs of scheduling criteria conflict in certain areas**

a. CPU utilization and response time: CPU utilization is increased if the overheads associated with context switching is minimized. The context switching overheads could be lowered by performing context switches infrequently. This could, however, result in increasing the response time for processes.

b. Average turnaround time and maximum waiting time: Average turnaround time is minimized by executing the shortest tasks first. Such a scheduling policy could, however, starve long-running tasks and thereby increase their waiting time.

c. I/O device utilization and CPU utilization: CPU utilization is maximized by running long-running CPU-bound tasks without performing context switches. I/O device utilization is maximized by scheduling I/O-bound jobs as soon as they become ready to run, thereby incurring the overheads of context switches.

**Table

Description automatically generated**

**Turnaround time**

**Table

Description automatically generated**

**CPU Utilization rate**

**Graphical user interface, text, application

Description automatically generated**

**[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

5.1.4 Dispatcher

Another component involved in the CPU-scheduling function is the dispatcher. The dispatcher is the module that gives control of the CPU to the process selected by the short-term scheduler. This function involves the following:

• Switching context

• Switching to user mode

• Jumping to the proper location in the user program to restart the program

The dispatcher should be as fast as possible, since it is invoked during every process switch. The time it takes for the dispatcher to stop one process and start another running is known as the dispatch latency

**Scheduling Criteria**

Different CPU-scheduling algorithms have different properties, and the choice of a particular algorithm may favor one class of processes over another

Many criteria have been suggested for comparing CPU-scheduling algorithms. Which characteristics are used for comparison can make a substantial difference in which algorithm is judged to be best. The criteria include the following:

• CPU utilization. We want to keep the CPU as busy as possible. Concep- tually, CPU utilization can range from 0 to 100 percent. In a real system, it should range from 40 percent (for a lightly loaded system) to 90 percent (for a heavily used system).

• Throughput. If the CPU is busy executing processes, then work is being done. One measure of work is the number of processes that are completed per time unit, called throughput. For long processes, this rate may be one process per hour; for short transactions, it may be ten processes per second.

• Turnaround time. From the point of view of a particular process, the important criterion is how long it takes to execute that process. The interval from the time of submission of a process to the time of completion is the turnaround time. Turnaround time is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/O.

• Waiting time. The CPU-scheduling algorithm does not affect the amount of time during which a process executes or does I/O; it affects only the amount of time that a process spends waiting in the ready queue. Waiting time is the sum of the periods spent waiting in the ready queue.

• Response time. In an interactive system, turnaround time may not be the best criterion. Often, a process can produce some output fairly early and can continue computing new results while previous results are being

Scheduling Algorithms

5.3.1 First-Come, First-Served Scheduling

By far the simplest CPU-scheduling algorithm is the first-come, first-served (FCFS) scheduling algorithm.

**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

UNIX system calls

|  |  |  |  |
| --- | --- | --- | --- |
| Process control | Windows  examples | UNIX  examples | Explanation |
| end, abort |  |  |  |
| load, execute |  |  |  |
| create/terminate process, | CreateProcess()  ExitProcess() | fork()  exit() |  |
| get/set process attributes |  |  |  |
| wait for time |  |  |  |
| wait/signal event, event | WaitForSingleObject() | wait() |  |
| allocate/free memory |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| File management | Windows  examples | UNIX  examples | Explanation |
| create/delete file | CreateFile() | open() |  |
| open, close |  |  |  |
| read, write, reposition | ReadFile()  WriteFile()  CloseHandle() | read()  write()  close() |  |
| get/set file attributes |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Device management | Windows  examples | UNIX  examples | Explanation |
| end, abort |  |  |  |
| load, execute |  |  |  |
| logically attach/detach devices, |  |  |  |
| get/set device attributes |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Information maintenance | Windows  examples | UNIX  examples | Explanation |
| get/set date or time |  |  |  |
| get/set system data |  |  |  |
| get process, file, or device attributes | GetCurrentProcessID() | getpid() |  |
| set process, file, or device attributes |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Communications | Windows  examples | UNIX  examples | Explanation |
| end, abort |  |  |  |
| load, execute |  |  |  |
| create/terminate process, | CreatePipe()  CreateFileMapping()  MapViewOfFile() | pipe()  shmget()  mmap() | Creates or opens a named or unnamed file mapping object for a specified file. |
| get/set process attributes |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Protection | Windows  examples | UNIX  examples | Explanation |
|  | SetFileSecurity() | chmod() |  |
|  | InitializeSecurityDescriptor() | umask() |  |
|  | SetSecurityDescriptorGroup() | chwon() |  |
|  |  |  |  |

**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.

**Kernel**

a) kernel is the program that constitutes the central core of the operating system

b) kernel is the first part of operating system to load into memory during booting

c) kernel remains in the memory during the entire computer session

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

To dynamically add probes to a running system, both in user processes and in the kernel, the CL tool used is:

DTrace (DTrace.exe) is a command-line tool that displays system information and events.

DTrace is an open-source tracing platform ported to windows.

DTrace was originally developed for the Solaris operating system

Types of kernels

|  |  |  |
| --- | --- | --- |
| Type | Purpose | Example |
| Monolothic | provide a communication facility between the client program and the various services that are also running in user space.  client program and service never interact directly, they communicate indirectly by exchanging messages with the microkernel. | Linux |
| Hybrid | Apple Mac OS X operating system uses a hybrid structure. It is a layered system in which one layer consists of the Mach microkernel | OSX |
| Mircokernel | take the separation of mechanism and policy to one extreme by implementing a basic set of primitive building blocks. These blocks are almost policy free, allowing more advanced mechanisms and policies to be added via user-created kernel modules or via user programs themselves. |  |
| Monolithic with module | modules should be a well-defined portion of the system, with carefully defined inputs, outputs, and functions |  |

**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**

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**

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)

Introduction

System Structures

Process Concept

Multithreaded Programming

Process Scheduling

Synchronization

Influential Operating systems

Deadlocks

Memory-Management Strategies

Virtual-Memory Management

File System

Implementing File System

System protection

System Security

The Linux System