# Lecture 1: intro.

Diagram

Description automatically generatedOperating system: a program that manages the computer hardware, provides a basis for app & acts as an intermediary between the device & the user.

Computer system components:

* Hardware.
* Operating system.
* Application programs.
* Users.

## User’s view:

* It varies according to the interface used.

GUI:

* OS is designed (mostly) for ease of use.
  + The goal is to maximize the work the user can do.
* Resource utilization: how various hardware & software resources are shared.
  + This system doesn’t focus on it.

Terminal:

* OS is designed to maximize resource utilization.
  + Ensures all available resources are used fairly & efficiently.
* Users connect to a mainframe/minicomputer through terminals.
* Users share resources & can share info.

## System view:

OS acts as a resource allocator.

* Manages resources (CPU time, memory, storage, I/O devices, ...).
* Decides how to allocate them to programs/users.
* OS is a control program.
* Control program: a prog that manages the execution of user programs to prevent errors & incorrect use of the comp.
* Concerned with the operation & control of I/O devices.

Fundamental Goals of computer:

* Execute user programs.
* Make solving user problems easier.

## Computer system organization.

A modern general purpose computer system consists of

* Diagram

  Description automatically generatedOne or more CPUs.
* Several device controllers connected through a common bus.
  + Each one oversees a specific type of device.
* Memory that is shared through that bus.

These controllers & the CPU can execute in parallel & they compete for memory cycles.

Bootstrap: an initial program that is required to start any comp.

* Typically stored in the ROM.
* Tends to be simple.

Any event is signaled by an interrupt from hardware || software.

* Hardware triggers it by sending a signal to the CPU through the system bus.
* Software triggers it by executing a system call (monitor call).

### Storage structure.

The CPU can only load instructions from memory.

Types of memory:

* Random Access Memory (RAM) || Main memory.
  + Volatile memory.
  + The only large storage that the CPU can access directly.
* Dynamic Random Access Memory (DRAM).
  + A type of ram but slower.
  + Requires constant refreshing.
* Read Only Memory (ROM).
  + Can’t be changed.
  + Stores static programs.
  + Non-volatile.
* Secondary storage.
  + Extension of main memory.
  + Non-volatile.
  + Types of storage:
    - Hard Disk Drives (HDD).
      * Disk is logically divided into tracks & sectors.
      * Disk controller determines the logical interaction between disk & computer.
    - Solid State Drives (SSD).
      * Faster than HDD.
      * Uses various technologies.

All forms of memory provide an array to store data.

* Each index has an address.
* Interactions are achieved through load & store instructions.
  + Load: moves data from main memory to a register.
  + Store: moves content from a register to main memory.
* Main memory is too small to store all programs & it’s a volatile storage.

### Computer-System Architecture.

1. Single general-purpose processor.
   1. Used in most systems.
2. Multiprocessor systems
   1. Also known as parallel systems || tightly coupled systems.

Advantages:

* + Diagram

    Description automatically generatedIncreased throughput.
  + Economy of scale.
  + Increased reliability (less degradation & fault tolerance).

Its Types:

1. Asymmetric multiprocessing: each processor is assigned a specific task.
2. Symmetric multiprocessing: each processor performs all tasks.
3. Multi-core design.
   1. One chip that has more than one core.
   2. Each core can do a separate task.

Diagram

Description automatically generated

## Clustered Systems.

* Diagram

  Description automatically generatedUses multiple CPUs to achieve any task.
* Composed of two or more individual system || nodes.

## OS structure.

Multiprogramming: the system runs multiple programs simultaneously.

* Increases CPU utilization.
  + By organizing tasks (code & data) & CPU always has a task to execute.
* Provide an environment that utilizes system resources effectively.
* Pool: an area on the disk where tasks/jobs are stored waiting for memory space to be available.
* Job Scheduling: selecting one job for the CPU to execute.
* Some tasks depend on other tasks.
  + Normal systems: CPU would idle.
  + Multiprogrammed systems: the CPU switches to another task.

Multitasking: a logical extension of multiprogramming.

* The CPU executes multiple jobs by switching among them.
* The switches happen so fast that the user doesn’t notice ( < one second).
* CPU scheduling: determines which process should be allocated to the CPU.
  + Used when several jobs are ready to run at the same time.
* Swapping: moves processes in & out of the CPU.
* Virtual memory: allows execution of processes that aren’t fully in memory.

## OS Operations.

TRAP: a software-generated interrupt caused by an error || a specific request from the user.

* Diagram

  Description automatically generatedTraps can signal Events (for the system to process).

### Dual mode operation:

* User mode.
* Kernel mode.

Mode bit: a bit that indicates the current mode ( 0 => kernel, 1 => user).

System process management activities:

* Scheduling processes & threads on the CPU.
* Creating & deleting user and system processes.
* Suspending & resuming processes.
* Providing mechanisms for process synchronization.
* Providing mechanisms for process communication.

System memory management activities:

* Keeping track of the used parts (memory).
* Deciding which processes/data to move in/out of memory.
* Allocating/deallocating memory space.

System file management activities:

* Creating & deleting files.
* Creating & deleting directories.
* Supporting primitives (manipulating files/directories).
* Mapping files onto secondary storage.
* Backing up files.

System mass-storage management:

* Free space management.
* Storage allocation.
* Disk scheduling.

## Caching.

* Data that is repeatedly used gets saved in the CPU’s cache.
* CPU’s cache is always checked before the main memory.
* Cache is typically smaller than RAM.

## Protection & security.

Protection: the method of controlling access to processes || resources.

Security: the methods of defending the system from external & internal attacks.

* Some operating systems have built-in security, others leave it to the user.

## Distributed system.

* A collection of physically separate computers that are connected.
* Users can access various resources.
* Benefits:
  + Increases computation speeds.
  + Increases functionality.
  + Increases data availability.
  + Increases data reliability.
* Some OSs have file access built in, others leave the choice to the user.

## Real time embedded systems.

Embedded computers: devices that have a very specific task.

* Systems are usually primitive.
* The OS provides limited features.
* Has little to no user interface.

## Multimedia systems.

* Systems that incorporate multimedia data.
  + The use audio & video files + conventional files (text, documents, …).
  + Has some limitations (delivery has some time restrictions).

## Handheld systems.

These devices have limited size.

* Small amount of memory.
  + Between 1 MB & 32 GB.
  + Must manage memory efficiently.
  + Most devices don’t use virtual memory.
* Slow processors.
  + Fast processors require more power & that needs a larger battery.
* Small displays.

## Client-server Computing:

Diagram

Description automatically generatedCompanies are shifting away from centralized systems and moving to PC servers.

## Peer-to-Peer computing:

* All nodes can act as a client & a server.

Diagram

Description automatically generated

## Cloud computing:

* Provides computing, storage, and apps across the internet.
* Uses virtualizations as a base for its functions.
  + Ex. Amazon EC2.
* Types:
  + Public cloud: available via a pay wall.
  + Private cloud: could for a company.
  + Hybrid cloud: includes both public & private.
  + Software as a Service (SaaS).
  + Platform as a Service (PaaS).
  + Infrastructure as a Service (IaaS).

Types of systems:

* Distributed system.
* Handheld systems.
* Multimedia systems.
* Real time embedded systems.
* General use systems.

# Lecture 2: OS structures.

## OS Services:

* OS provides an environment for program execution.
* Graphical user interface

  Description automatically generatedOS Provides some services for the programs & the users.

1. User interface (UI).

* Command line interface (CLI).
  + Uses text commands.
* Batch interface.
  + Commands & directives are saved in files and executed.
* Graphical user interface (GUI).

1. Program execution.

Helps the system load, execute, and terminate programs.

1. I/O operations.

* Required by some programs.
* Used to control external devices.
* The user can’t directly control these devices.
  + More efficient & protection.

1. File-system manipulation.

* Provides a way for programs to read, write, create, delete, search, and list files & directories.
* Some OS provides a variety of file systems.
* Some provide specific features || performance characteristics.

1. Communications.

* Exchanges info between processes.
  + Processes can be on the same device or another device.
* Can be implemented via a shared memory || message passing.

1. Error detection.

* Errors can occur in the CPU, memory, hardware, I/O devices, and user programs.
  + The OS takes a different action for each type of error.

## OS System Services:

* These services ensure the efficient operation of the system itself.
* Systems with multiple users share resources to increase efficiency.

1. Resource allocation.

* Helps regulate resources between the different users & programs running at the same time.
* CPU cycles, main memory, & file allocation have a special allocation code.
* I/O devices have a general request & release code.
* To determine how to use the CPU OSs have CPU scheduling routines.

They rely on:

* + The speed of the CPU.
  + The jobs needed.
  + The number of registers available.
* There are different routines for printers, modems, and USB storage drives.

1. Accounting.

* Keeps logs on what users use & how much they use them.
* May be used for accounting || accumulating usage statistics.
* Usage statistics may be valuable for researchers.
  + To improve computing services.

1. Protection & security.

* Helps users control the use of their info/files.
* Protection: ensures all access to system resources is controlled.
* Security:
  + Requires each user to authenticate themselves.
  + Defends against external I/O devices.

## System calls.

* Provide an interface to the OS’s services.
* Available as routines (written in C & C++).
* Low level tasks need to be written in assembly.

### Call categories:

1. Process control.
   1. End/abort.
   2. Load/execute.
   3. Create || terminate process.
   4. Get || set process attributes.
   5. Wait for some time.
   6. Wait for an event || signal an event.
   7. Table

      Description automatically generatedAllocate || free memory.
2. File manipulation.
   1. Create || delete file.
   2. Open || close file.
   3. Read || write || reposition text.
   4. Get || set file attributes.
3. Device manipulation.
   1. Request || release device.
   2. Read || write || reposition.
   3. Get || set device attributes.
   4. Logically attach || detach devices.
4. Information maintenance.
   1. Get || set time and date.
   2. Get || set system data.
   3. Get process, file, device attributes.
   4. Set process, file, device attributes.
5. Communication.
   1. Create || delete communication connection.
   2. Send || receive messages.
   3. Transfer status information.
   4. Attach || detach remote devices.
6. Protection.

### C Library:

* Provides a portion of the system call interface for many versions of UNIX & Linux.
* It intercepts program calls and invokes the necessary system calls(s).

## System programs:

* Provide a convenient environment for program development & execution.
* Some are user interfaces to system calls.
* Most users view OS as system programs & not system calls.

They are divided into:

* File manipulation.
  + Create, delete, copy, rename, print, dumb, list files & directories.
* Status information.
  + Date, time, available memory, disk space, number of users.
  + Details about performance, logs, debugging info.
  + Formats & prints the output to the terminal || other devices.
  + Some systems implement a registry to store & retrieve configurations.
* File modification.
  + Text editors to create & modify files.
  + Special commands to search file content || perform transformations on text.
* Programming language support.
  + Compilers, assemblers, debuggers, interpreters.
* Program loading & execution.
  + Absolute loaders, linkage, editors, overlay loaders, debugging systems.
* Communications.
  + Allows the creation of virtual connections among processes, users, and computer systems.
  + Allows users to:
    - send messages to each other (screen, browser).
    - Send e-mails.
    - Log in remotely.
    - Transfer files.
* Application programs.

Diagram, schematic

Description automatically generatedVirtual machines: the abstraction of a computer’s hardware into several different execution environments.

* Each environment appears as if it’s running on its own.
* Takes a layered approach to its logical conclusions.
  + Treats the hardware & the OS as hardware.
* Provides an interface identical to the hardware.
* The OS creates virtual memory.
* Each guest is provided with a virtual copy of the computer.

## OS debugging:

* Debugging: Finding & fixing errors and bugs.
* OS generates log files containing the error’s info.
* App failures can generate core dump files (captures process memory).
* OS failure can generate crash dump files (contains kernel memory).
* Performance tuning can optimize system performance.
* Kernighan’s Law: debugging is twice as hard as writing code.
  + Cleverly written code is hard to debug it.

## System boot.

Booting: the procedure of starting a computer by loading the kernel.

Bootstrap: a small piece of code that locates the kernel, loads it into memory, and starts its execution.

* Some computers use a two-step process.
  + Simple bootstrap loader fetches more complex boot program from disk.
  + Complex boot program loads the kernel.
* Firmware is used to hold the initial boot code.

## OS design & implementation.

* They are not solvable, but some approaches have been successful.
* Internal structure can vary widely.
* Starts by defining goals & specifications.
* Affected by hardware & system type.
* User goals:
  + Should be convenient to use.
  + Should be easy to learn.
  + Should be reliable, safe, fast.
* System goals:
  + Should be easy to design, implement, and maintain.
  + Should be flexible, reliable, error free, and efficient.
* Policy: decides what will be done.
* Mechanism: determines how to do something.
* The separation between the two allows maximum flexibility.

# Lecture 3: process management.

## Process:

* A program in execution.
* OR the unit of work in most systems.

### Its properties:

* More than the program’s code (text section).
* Requires resources (CPU time, memory, etc.) to accomplish its task.
* Resources are allocated when the process is created || while its executing.
* Box and whisker chart

  Description automatically generated with low confidenceSystems: a collection of processes.
* OS processes execute system code.
* User processes execute user code.
* All processes are executed concurrently.
* Processes include:
  + Text section (program code).
  + Stack (temporary data).
  + Data section (global variables).
  + Heap (contains dynamically allocated memory).

### Process concept:

* Early computers allowed only one program to execute at a time.
* The executed program has complete control of the system & access to all system resources.
* Current computers allow multiple programs to be executed concurrently.
* OS may need to support its own internal programmed activities.

## OS process responsibilities:

* The creation & deletion of user/system processes.
* The scheduling of processes.
* Communication & deadlock handling for processes.

## Program & processes.

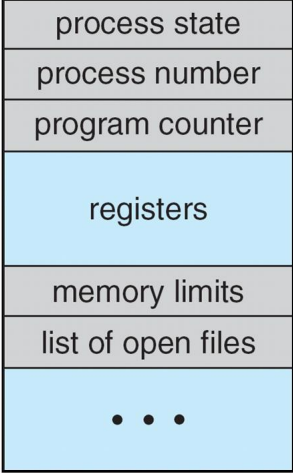
* Not all programs are processes.
* A program is a passive entity (Executable file).
* A process is an active entity.
* A program becomes a process when it’s loaded into memory.

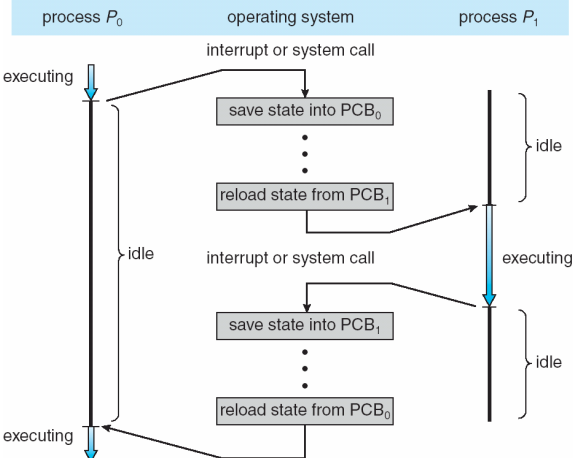
## Process state.

* A process changes state as it executes.
* Diagram

  Description automatically generatedState types:
  + New: the process is being created.
  + Running: instructions are being executed.
  + Waiting: the process is waiting for an event.
  + Ready: the process is waiting to be assigned to a processor.
  + Terminated: the process has finished execution.

## Process control block (task control block).

* Each process is represented by a process control block (PCB).
* Contains info associated with a specific process.
* Includes these:
  + Process state.
  + Program counter (location of next instruction).
  + CPU registers.
  + CPU scheduling info, priorities, scheduling queue pointers.
  + Memory management info.
  + Accounting info.
  + I/O status info.

CPU switching between processes.

## Process scheduling:

* Selecting an available process for program execution.
* Single processor systems can’t have more than one running process.

### Scheduling queue.

* Chart, waterfall chart

  Description automatically generatedJob queue: contain new Processes that enter the system.
* Device queue: set of processes waiting for an I/O device.
  + Each device has its own queue.
* Ready queue: contains processes that are in the RAM & ready to execute.
  + Its header contains pointers to the first & final PCBs.
  + Each PCB has a pointer to the next one.
* Queues are stored as a linked list.
* time sharing: CPU quickly switches between processes to maximize its use.
  + process scheduler selects and sends the process.

## Process events:

1. process issues an I/O request.
   1. It gets placed in the device queue (I/O queue).
2. The process creates a new subprocess.
   1. It waits for the subprocesses to terminate.
3. The process is removed forcibly from the CPU.
   1. A result of an interrupt.
   2. Process is places back in the ready queue.

## Schedulers.

Selects processes from all the different queues.

* Long-term scheduling (job scheduler).
  + Selects processes from Pool (job queue) & loads them into Ready queue.
* Medium-term scheduling.
* Short-term scheduling (CPU scheduler).
  + Selects a process from the ready processes & allocates it to the CPU.

### Medium-term scheduling (swapping):

* Can be added if the number of running tasks needs to be decreased.
* Diagram

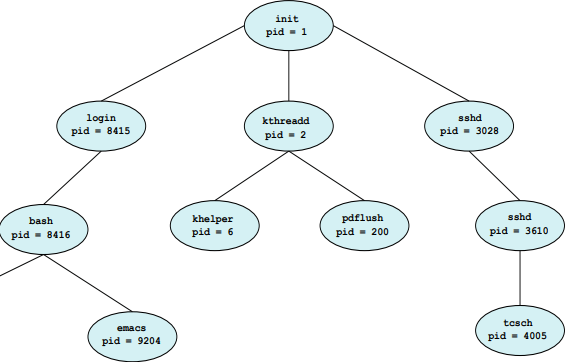
  Description automatically generatedRemoves processes from memory.
* Stores them on a disk.
* Brings them back later to continue execution.

## Context Switch.

* Interrupts cause the OS to change the CPU’s current task.
* The system needs to save the current context so it can be restored later.
* Context is represented in the PCB of the process.
* Includes:
  + Value of CPU registers.
  + The process state.
  + Memory management information.
* These operations are called State Save & State Restore.
* Context switch: doing a State Save of the current process & a State Restore of a different process.

## Operations on processes.

### Process creation.

* A process may create other processes (using a system call).
  + Original process is called a parent process & the new process is a child.
* This structure is called a tree.
* Most OSs use process identifier (pid) to identify processes.
  + Typically, an integer number.
* After creation:
  + The parent can run concurrently with its children.
  + The parent can wait until some/all children have terminated.
* Child process address:
  + Its address is a duplicate of the parent’s address (has the same data & program as the parent).
  + The child process has a new program loaded into it.

### Process termination.

* Process terminates when it finishes execution.
* Uses exit() system call.
  + May return a status value to its parent process using wait() system call.
* All resources of the process are de-allocated by the OS.
* Parent-child termination:

Parent may terminate a child.

* + The child exceeded its usage of some resources.
  + The task assigned to the child is no longer required.
  + The parent is exiting & the OS doesn’t allow a child to continue without its parent.

## Inter-process communication (IPC).

* Processes may be an independent process || a cooperating process.
* Independent process:
  + Can’t affect || be affected by other processes.
  + Any process that doesn’t share data with other processes.
* Cooperating process:
  + Can affect || be affected by other processes.
  + Any process that shares data with other processes.
  + Requires an IPC mechanism.

### IPC models:

* Shared memory model.
  + A region of memory that is shared between cooperating processes.
  + Processes exchange info by reading & writing data to that region.

Diagram

Description automatically generated

* Message passing model.
  + Processes sends messages to communicate.

## Shared-memory systems.

* Normally OS tries to prevent a process from accessing another process’s memory.
* Shared memory requires two || more processes to agree to remove this restriction.
* Processes can exchange data by writing & reading to a shared area.
* Processes are responsible for:
  + Determining the Shared area & data form.
  + ensuring that they don’t write to the same location simultaneously.

## Message-passing systems.

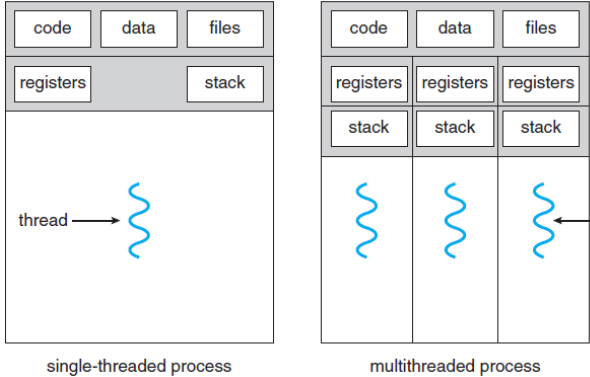
* has a mechanism that allows processes to communicate & synchronize actions without sharing the same address space.
* Useful in distributed environment.
* Processes may be on different computers.

## Communication in Client-Server systems.

* Sockets.
* Remote procedure calls.
* Pipes.
* Remote method invocation.

# Lecture 4: Threads.

Thread: a basic unit of CPU utilization.

* Shares its resources (code & data sections, etc.) with other threads belonging to the same process.
* A traditional (heavyweight) process has a single thread of control.
* The more threads a process has the more tasks it can perform.
* Most software packages are multithreaded.
  + Implemented as a separate process.
  + Has several threads of control.

It’s made of:

* Thread ID.
* Program counter.
* Register set.
* Stack.

## Multithreaded programming benefits:

* Responsiveness.
  + Apps can run even if a part is blocked or performing a long operation.
  + Useful when designing user interfaces.
* Resource sharing.
  + Threads share memory & resources of the process.
  + Allows an app to have several different threads with the same space.
* Economy.
  + Creating & managing threads is less resource costly than processes.
* Scalability.
  + Threads may run in parallel on different processing cores.

## Multicore (multiprocessor)programming:

* Evolved from single CPU systems.
* Each core appears as a separate processor to the OS.
* Can be in the same CPU chip || different CPU chips.
* Provides:
  + A mechanism for more efficient use of the multiple cores.
  + Improved concurrency.
* System can assign a different thread to each core.

## Parallelism & concurrency.

* Table

  Description automatically generatedParallel systems.
  + Can perform more than one task simultaneously.
* Concurrent systems.
  + Supports more than one task (alternates between them).
  + CPU schedulers provide the illusion of parallelism.
    - Table

      Description automatically generatedRapidly switches between processes in the system.
    - Used in single processor systems.

## Programming challenges.

* Identifying tasks.
  + Examining apps to find areas that can be divided into separate/concurrent tasks.
  + Independent tasks can run in parallel on individual cores.
* Balance.
  + Tasks must perform equal work (equal value).
  + Some tasks are not worth a separate core to run (small value).
* Data splitting.
  + Data accessed/manipulated by tasks must be divided.
* Data dependency.
  + Data accessed my tasks must be examined for dependencies between two || more tasks.
  + Programmers must ensure tasks execution is synchronized.
* Testing & debugging.
  + Testing & debugging multithreaded apps is more difficult than single.

## User threads.

* Management is done by user-level threads library.
* Primary thread libraries Include:
  + POSIX Pthreads.
  + Windows threads.
  + Java threads.

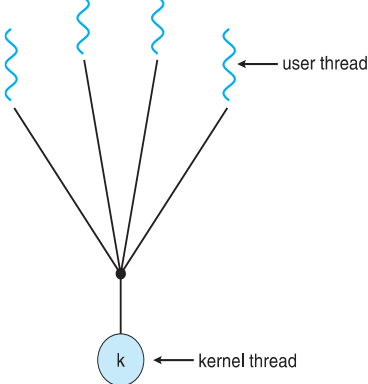
## Kernal threads.

* Supported by the kernel.
* Examples (windows, Solaris, Linux, Tru64 UNIX, Mac OS X).

Thread library: provides programmers with an API for creating & managing threads.

## Multithreading models.

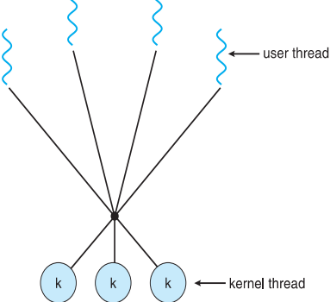
### Many-to-one model.

* Maps many user level threads to one kernel thread.
* Thread management is done by thread library in user space (efficient).
* The process will block if a thread makes a blocking system call.
* Multiple threads are unable to run (only one thread can access kernel).

### Diagram Description automatically generatedOne-to-one model.

* Maps each user thread to a kernel thread.
* Provides more concurrency than Many-to-One model.
* Allows other threads to run when one makes a blocking call.
* Allows multiple threads running in parallel.
* Creating a user thread requires creating a kernel thread (disadvantage).
* Most implementations restrict the number of threads.
* Used in Linux & Windows.

### Many-to-many model.

* maps many user threads to many kernel threads.
* Number of kernel threads is less || equal to user threads.
* Number of kernel threads changes based on:
  + The application.
  + The machine (multi || single).

# Lecture 5: Process Synchronization.

Co-operating process: a process that can affect || be affected by other processes.

* Can directly share a logical address space (code & data).
* Can share data through files & messages.
* Concurrent access to shared data can lead to data inconsistency.

## Process co-operation types:

* Competition: Processes unaware of each other.
  + Independent processes (not intended to work together).
  + Ex) two processes trying to access a printer.
* Co-operation by sharing: Processes indirectly aware of each other.
  + Interactions happen by accessing some object (files, databases, variables, etc.).
  + Processes may use & update the shared data without the other processes knowing.
* Co-operation by communication: Processes directly aware of each other.
  + Used between processes that depend on each other (one process output is another’s input).
  + Processes sends & receives messages.

## Concurrency problems.

* Chart, box and whisker chart

  Description automatically generatedConcurrency: the ability to run multiple tasks at the same time.
* Uni-processor CPU:
  + tasks are divided into sub-tasks (different processes).
  + Interleaving: Each process can use the processor(CPU) for a limited time.
  + Can cause data inconsistency.
* Multi-processor CPU:
  + Timeline

    Description automatically generatedTasks run in parallel on different processors.
  + May lead to overlapping & data inconsistency.

### The Producer consumer problem.

* Producer: a process that produces information & stores it in a buffer.
* Diagram

  Description automatically generated with medium confidenceConsumer: a process that consumes info from the buffer.
* They don’t take turns accessing the buffer.
* Problems:
  + Producer tries to place an item when the buffer is full.
  + Consumer tries to take an item from an empty buffer.
* Text, letter

  Description automatically generatedText, letter

  Description automatically generatedSolutions:
  + Use an integer counter to keep track of the items.
    - Code for both parties =>.
  + Secondary problem: This can cause problems if both parties try to change the counter value.
    - This is caused by how machine language handles increments & decrements.
    - Increment.
      * Register1 = counter.
      * Register1 = register1 + 1.
      * Counter = register1.
    - Decrement.
      * Register2 = counter.
      * Register2 = register2 - 1.
      * Counter = register2.
    - Text, letter

      Description automatically generatedThis slow process can cause the counter to land on a wrong value.

### Race condition problem.

* Happens when multiple processes read & write to shared data item and the result depends on timing.
* Processes are either unaware || indirectly aware of each other.
* Ex) two processes placing an order at the same time.
* Solution:
  + Synchronize processes & co-ordinate between co-operating processes.

### The Critical-Section problem.

In a system consisting of N processes (P0, P1, …, Pn-1).

Critical section: a segment of code in each has a segment of code.

* While in it Processes may change common variables, update tables, etc.
* OSs allows only one process to execute its critical section at a time.
* Table

  Description automatically generated with medium confidenceProblem:
  + Designing a protocol for process co-operation.
* Solution:
  + Creating areas before & after the critical section.
    - Chart, bar chart, histogram

      Description automatically generatedEntry section.
    - Exit section.
    - Remainder section.
  + Graphical user interface, text, application

    Description automatically generatedEach process must ask for permission (in entry section) before entering the critical section.
  + Processes move from the critical section to the exit & remainder sections.
* Solution requirements:
  + Mutual exclusion:
    - If a process is executing in the critical section, no other process can execute in the critical section.
  + Progress:
    - If there is no process executing in its critical section, the selection of next process to execute can’t be delayed.
  + Bonded waiting:
    - After a process requests to enter its crit section & before the request is granted, the number of times other processes are allowed to enter their crit section is limited.
    - Assumes each process executes at a non-zero speed.
    - Doesn’t have any assumptions about relative speed of the other processes.

## OS approach to the critical section:

* Preemptive kernels:
  + Allows a process to be interrupted while it’s running in kernel mode.
* Non-preemptive kernels:
  + Doesn’t allow a process running in kernel mode to be interrupted.
  + Processes run until they exit kernel mode || temporarily loses control of the CPU || gets blocked.

## Solution types:

* Software solution: code that runs before a process enters crit section.
  + Shared lock variable.
  + Peterson’s solution.
  + Semaphore.
* Hardware solution: utilizing system components to avoid race condition.
  + Disable interrupts.
  + Atomic hardware instructions.

### Peterson’s solution (Software):

* Table

  Description automatically generatedRestricted to two processes that alternate execution.
* Current process is represented by Pi.
* Other process is represented by Pj ( j = i -1).
* Not guaranteed to work on modern computer architectures.
* Shared data items:
  + Int turn.
  + Boolean flag[2].
* Variable “turn” indicates whose turn it is to enter crit section.
* The “flag” array (if true) is used to indicate if a process is ready to enter its crit section.

Graphical user interface, text, application, letter

Description automatically generated

### Mutex locks (Hardware).

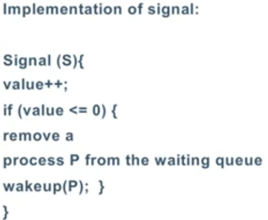
* Mutex is short for mutual exclusion.
* Lock: A Boolean variable in the system.
* A process must acquire a lock before entering its crit section (lock must be available).
* The lock is released when the process exits that section.
* “acquire()” function acquires the lock.
* “release()” function releases the lock.

Text, letter

Description automatically generated

### Semaphores:

* Semaphore: an integer variable (S) that is accessed only through two operations.
  + Wait() OR ‘P()’ .
  + Signal() OR ‘V()’.
* Only one function can modify the same semaphore value.

Semaphore usage:

* Can be used to solve various synchronization problems.
* States:
  + When a process is in its crit section, S = 0.
  + If a process enters queue, S gets decremented by one.
  + If a process exits queue, S gets incremented by one.
  + If the queue is empty & there is no process in crit section, S = 1.
* Counting semaphore: can range over an unrestricted domain.
  + Used to control access to a resource consisting of a finite number of instances (Semaphore = number of instances).
* Binary semaphore: either 1 || 0 (can be used instead of mutex lock).
* Text

  Description automatically generated with medium confidenceText

  Description automatically generated with low confidenceDeadlock: two or more processes are waiting indefinitely for an event that is caused by one of them.

## Software solution disadvantage:

* Locks are shared variables.
* Multiple processes might enter their crit section if the lock was 0.

# Lecture 6: process Scheduling.

## Scheduling:

* A fundamental OS function (central to its design).
* All computer resources are scheduled before use.

## Diagram Description automatically generatedCPU- I/O Burst Cycle.

* Process execution begins with CPU burst.
* Process execution consists of:
  + a cycle of CPU execution.
  + I/O wait.
* Processes alternate between these two states.
* Processes terminate by sending a system request.

## CPU scheduler (short term scheduler).

* Used if the CPU is idle (the OS selects a process from the ready queue).

### Preemptive scheduling.

* CPU scheduling decision circumstances:

1. When a Process switches from running to waiting.
2. When a Process switches from running to ready.
3. When a Process switches from waiting to ready.
4. when a process terminates.

* Preemptive Scheduling: scheduling that takes place in scenario 2 & 3.
* Non-preemptive scheduling: scheduling that takes place in scenario 1 & 4.

### Dispatcher.

* Part of CPU scheduling.
* Dispatcher: A module that gives control of the CPU to the process (selected by short term scheduler).
* Dispatch latency: the time it takes to stop one process & start another.

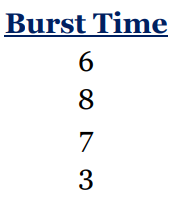
### Scheduling Criteria.

* CPU utilization: keeping the CPU as busy as possible.
  + Should be maximized.
* Throughput: num of processes that complete their execution (in a time unit).
  + Should be maximized.
* Turnaround time: amount of time to execute a specific process.
  + Should be minimized.
* Waiting time: the time a process waits in ready queue.
  + Should be minimized.
* Response time: time between request submission & first response.
  + Should be minimized.

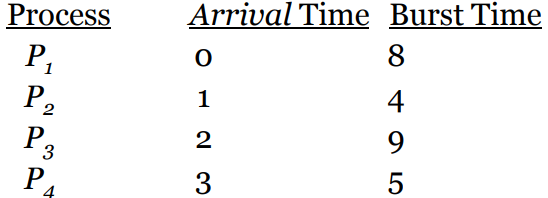
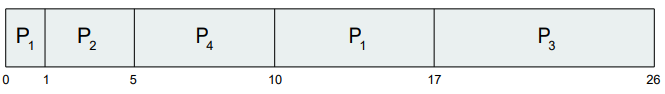
## scheduling algorithms.

### A picture containing chart Description automatically generatedFirst come, first served (FCFS) scheduling.

* processes are scheduled based on arrival.
* Can be represented using Grantt chart.
* Avg waiting time = total waiting / num of processes.
  + Can be reduced by changing order of arrival.
* Convoy effect: processes wait for a long process taking the CPU.
  + Results in lower CPU & device utilization.



### Shortest job first (SJF) scheduling.

* Processes are scheduled based on burst time.
* Uses FCFS if two processes have the same burst time.
* Used frequently with long-term scheduling.
* Preemptive SJF:
  + Processes enter the CPU by order of arrival.
  + If a process shorter than the one in the CPU arrives, they swap.

### Priority Scheduling.

* Graphical user interface, application

  Description automatically generatedUses a priority number that’s linked to each process.
  + Priorities have a range (Ex. 0 to 7).
  + Chart

    Description automatically generated with low confidenceFor high priority Some systems use the smallest number and others use the biggest.
* CPU is allocated to the highest priority.
* SJF is a special case of priority scheduling.
  + The higher the burst time, the lower the priority.
* Priority types:
  + Internally defined:
    - Uses a measurable quantity to compute the priority of a process.
    - Ex) time limits, memory requirements, num of open files, etc.
  + Externally defined:
    - Priority is set by a criteria outside the OS (often political).
    - Ex) process importance, type & amount of funds.
* Priority scheduling types:
  + Preemptive priority.
    - Processes enter the CPU until a process with a higher priority arrives (they swap).
  + Non-preemptive priority.
    - When a process arrives (with high priority), it gets placed at the front of the ready queue and it waits for the CPU.
* Blocked processes: a process that is ready to run but is waiting for the CPU.
* Indefinite blocking (Starvation): happens when the CPU leaves a low priority process waiting indefinitely.
  + These processes either wait for an opening || the computer will crash.
* Aging: gradually increasing the priority of a process that has been waiting for a long time.

### Round-Robin (RR) scheduling.

* Text

  Description automatically generatedLike FCFS but with preemption to allow the system to switch between processes.
* Designed for timesharing systems.
* Time slice (Time quantum): a small unit of time.
  + Generally, between 10 to 100 milliseconds.
* A picture containing box and whisker chart

  Description automatically generatedReady queue is treated as circular queue.
* If time slice is large, RR is like FIFO (arrival order).
* Each process gets 1/n of the CPU time.
* Processes wait (n-1) \* time slice.
* Implementation:
  + Processes are placed in ready queue by order of arrival.
  + CPU scheduler takes the first process and sets a timer to interrupt after 1 time slice.
  + If process burst time is less than time slice, the process leaves the CPU.
  + If process burst time is more, process gets interrupted & process will be placed at the end of the queue.

### Multilevel Queue scheduling.

* Used when processes are classified into different groups.
  + Chart, bar chart

    Description automatically generatedThey may have different response time requirements & scheduling needs.
* It divides the ready queue into several separate queues.
* Processes are permanently assigned to one queue.
* Each queue has its own scheduling algorithm.
* Uses fixed priority preemptive scheduling between queues.

### Multilevel feedback queue scheduling.

* A picture containing text

  Description automatically generatedAllows processes to move between queues.
* Separates processes based on CPU bursts characteristics.
* Uses aging to move older processes.
* Its parameters:
  + Number of queues.
  + Each queue scheduling algorithm.
  + Aging method.
  + Reverse aging method (moving a process to a lower priority queue).
  + Queue selection method.

### Multiple processor scheduling.

* Uses load sharing to utilize all CPUs.
* Scheduling becomes more complex.

## Algorithm evaluation.

* Before selecting an algo we must define the importance of some elements.
* They may include:
  + Maximizing CPU utilization (max response time is 1 second).
  + Maximizing throughput.

### Deterministic modeling.

* Analytic evaluation: uses the given algo & the system’s workload to produce a number that’s used to evaluate the algo’s performance for that workload.
* Deterministic modeling is a type of analytic evaluation.
  + It sets a workload and tests all available algorithms (requires exact numbers).
  + It’s fast, simple & provides exact numbers (performance numbers).

### Simulations

* Queueing models are limited & simulations are more accurate (still limited).
* They gather stats used to indicate the algo’s performance.
* Simulation data is collected from:
  + Random number generators (using probability distribution).
  + Tracing tape record sequences of real system events.
* Implementation:
  + Implements a new scheduler & tests it in real systems.
    - High cost & risk.
    - Can produce different results based on environments.
  + Some schedulers are flexible & can be modified to fit the system.
  + Some have APIs to modify priorities.

# Lecture 7: Deadlocks.

Deadlock: a situation where a waiting process can’t change its state (its waiting for resources used by other processes).

## System model.

* Consists of a finite number of resources.
* Resources are partitioned into several types (classes).
* Resource types can have more than one instance (CPU type has two instances if the system has two CPUs).
* Some instances are not identical (printers on different floors).
* A process can’t request resources more than what’s available.
* Steps:
  + Request: process requests resources.
    - If resources are being used, the process must wait until it’s free.
  + Use: the process uses the resources to finish its task.
  + Release: process releases resources.
* Deadlocked state set:
  + A set of processes where every process in the set is waiting for an event caused by another process in the same set.
  + Events may be resource acquisition & release.
  + Resources may be physical (ex. printers) || logical (ex. files).

## Deadlock characterization.

### Deadlock conditions.

* Mutual exclusion.
  + At least one resource must be held in a non-sharable mode.
* Hold and wait.
  + A process must be holding at least one resource & waiting for extra resources used by other processes.
* No preemption.
  + Resources can only be released when the process is done using it.
* Circular wait.
  + Waiting processes must exist in a circular pattern (P0 => P1 => P2… Pn => p0).

### Resource allocation graph.

* Consists of a set of vertices (V) & a set of edges (E).
* Vertices are divided into active processes (P) & resources (R).
* Request edge: A link from Pi to Rj is denoted by Pi -+ Rj (process i requested resource j).
* Assignment edge: A link from Rj to Pi is denoted by Rj -+ Pi (resource j is allocated to process i).
* Circles represent processes & rectangles represent resources.
* Resources with different instances have dots inside their rectangles.
* Steps:
  + Process (P) requests resource type Ri.
  + A request edge is inserted in the resource allocation graph.
  + When the resource is available, that edge becomes an assignment edge.
  + When the process finishes its task, the assignment edge is deleted.
* Cases:
  + Resource types have one instance:
    - If there is a cycle, a deadlock has occurred (each process in the cycle is deadlocked).
  + Resource types have several instances:
    - If there is a cycle, a deadlock may have occurred.

## Handling deadlocks.

1. Using protocols to prevent/avoid deadlocks.
2. Allowing the system to enter deadlock, detect it, and recover.
3. Ignore any deadlock.
   1. Used by most OSs.
   2. The application developers must write programs to handle deadlocks.

Deadlock prevention: a set of methods that make sure that at least one of the deadlock conditions is not true.

* They place constraints on how requests are made.

Deadlock avoidance: the OS requires any process to send extra info in advance (which resources it will request & use).

for more info about deadlock avoidance & prevention (book pages 76 – 82).



## Banker’s algorithm.

* The resource allocation graph is not valid for systems with multiple instances of each resource type.
* Banker’s algorithm: an avoidance algorithm that’s used for systems with multiple instances of each resource type.
  + Can be used in banking systems (stops system from allocating all cash).
  + Table

    Description automatically generatedn = number of processes.
  + m = number of resource types.
* Used Data structures:
  + Available: the number of available resources.
    - A vector of length ‘m’.
    - If Available[j] = k, then k instances are available of that type.
  + Max: the maximum demand of each process.
    - A matrix of size n \* m.
    - If Max[i][j] = k, then process Pi can request up to k instances or resource Rj.
  + Allocation: the number of resources of each type that are allocated.
    - A matrix (n \*m).
    - If Allocation[i][j] = k, then Pi has k instances of Rj.

Text

Description automatically generated with medium confidence

* + Need: the remaining resource needs of each process.
    - A matrix (n \* m).
    - If Need[i][j] = k, then Pi may need k more instances of Rj.
    - Need[i][j] = Max[i][j] – Allocation[i][j].

## Text Description automatically generatedSafety algorithm.

* Used to find if a system is in a safe state or not.
* Safe state: a state where the system can allocate resources to each process in some order and still avoid deadlock.
* May require m \* n2 operations.

## Resource-Request algorithm.

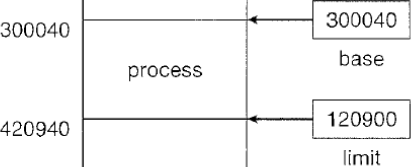
* Used to determine if requests can be safely granted.
* Requesti is the request vector for Pi.
* Actions after a request:

1. If **Request**i ≤**Need**i, go to step 2.
   1. Otherwise, raise an error condition (process has exceeded its maximum claim).
2. If **Request**i ≤ **Available**, go to step 3.
   1. Otherwise, Pi must wait (resources are not available).
3. Have the system pretend to have allocated the requested resources to process Pi by modifying the state as follows:
   1. **Available** = **Available**–**Request**i.
   2. **Allocation**i = **Allocation**i + **Request**i.
   3. **Need**i = **Need**i –**Request**i.

# Lecture 8: Main Memory.

* CPU fetches the next instruction from memory (using program counter).
* CPU can only access the main memory & registers.
* Cashe: memory buffer used to reduce the speed difference.

## Basic hardware:

* To make sure that each process has a separate memory:
  + Base register: holds the smallest legal physical memory address.
  + Limit register: specifies the size of the range.
* These registers are only loaded by the OS.

## Address binding:

* Programs are binary exe files on disks.
* They are moved to processes to be executed.
* Input queue: list of waiting processes.

## Logical vs Physical address space.

Logical address (virtual address): an address generated by the CPU.

Logical address space: set of all logical addresses generated by a program.

physical address: the address as it’s seen by the memory unit.

Physical address space: set of all physical addresses corresponding to one logical address space.

A picture containing text, screenshot, diagram, line

Description automatically generatedMemory management unit (MMU): a hardware device responsible for mapping logical addresses to physical ones.

* Base register = relocation register.
  + MMU adds its value to each logical address.
* User program never sees the physical address.

## Dynamic loading.

* Used to optimize memory space utilization.
* Keeps all processes on disk in a relocatable load format.
* Main program is loaded into memory & executed.
* If it needs a process, it checks if it’s loaded or not.
  + If not, the relocatable linking loader loads the process into memory.
  + Program’s address tables are updated.
* Advantage:
  + All unused processes are never loaded.
  + Useful for codes that handle rare cases.

## Swapping.

* Swapping processes in & out of memory.
* Processes swapped out are stored in a backing store.
* Used with priority scheduling.
* Sometimes it’s called roll out, roll in.
* Backing store: an area in a fast disk.
  + Provides direct access to memory images.
  + System maintains a ready queue of all the memory images in the backing store.
* The dispatcher checks if the next process is in memory.
  + If there is no space for it, it swaps out a process to free some space.
* These systems have a high context switch time.
  + Because of the transfer time (from store to memory).
  + To reduce time systems only swap processes when they are needed.
    - Using memory requests & releases.

## Contiguous memory allocation.

* Main memory must house the OS & user processes (problem).
* Memory allocation must be efficient.
* Contiguous memory allocation divides memory into two parts (low & high).
* OS is usually placed in the low memory (with the interrupt vector).

### Memory mapping & protection.

* To protect the OS & user data in memory from being modified, each new address is generated using the relocation & limit registers.
* Relocation register = the smallest physical address.
* Limit register = range of logical addresses.
* Logical addresses must be less than the limit register.

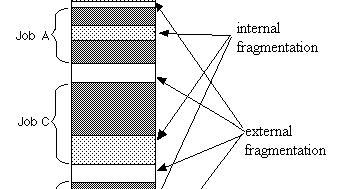
### Memory types:

* Fixed partition scheme:
  + Divide memory into several fixed size partitions.
  + Each partition can only contain one process.
  + Degree of multi-programming is bound by the number of partitions.
  + Originally used by IBM’s OS (MFT).
* Variable partition scheme.
  + Partitions vary in size.

### Memory allocation.

* OS has a table containing all available & occupied memory.
* Available memory is considered as one large block.
* Processes are placed into the input queue.
  + When there is enough memory space, the OS loads the process.
* the OS orders the input queue according to the scheduling algorithm.
* OS might skip over large processes.

### Fragmentation.

* External fragmentation: storage is fragmented onto large number of small holes.
  + Happens due to loading and removing processes.
* Internal fragmentation: a process occupies an area bigger than what it needs.
  + System sees the free area as allocated.

### Compaction.

* A solution to external fragmentation.
* Shuffles the memory, making the free memory a continuous part.
* Not always possible.
  + If relocation is static & done at load time.
* Possible if relocation is dynamic & done at execution time.
* Moves the program & its data + changes the base register.
* Simple method:
  + Moving all processes to one end of the memory & the free holes to the other end.
  + This method is expensive.

## Paging.

* Memory management scheme that allows a process to have non-continuous addresses.
* Avoids external fragmentation & the need for compaction.
* Solves the problem of fitting memory chunks of varying sizes onto the backing store.
  + Processes swapped out of memory need space in the backing store.
* Used in most OSs.

### Paging’s Basic Implementation.

* Frames: fixed sized physical memory blocks.
  + Backing store is divided into frames (their sizes = memory frames).
* Pages: same sized logical memory blocks.
* For a program to execute its pages are loaded into available memory frames.
* Addresses are divided into two parts:
  + Page number (P): used as an index to the page table.
  + Page offset (d).
* A picture containing diagram, plan, technical drawing, sketch

  Description automatically generatedPage table: contains the base address of each page (in physical memory).
* A picture containing text, diagram, technical drawing, plan

  Description automatically generatedProcess physical memory address = (page index \* page size) + offset.
  + Ex. page size = 4.
    - process (a) = (5 \* 4) + 0 = 20.
    - process (b) = (5\*4) + 1 = 21.
    - process (m) = (2 \*4) + 0 = 8.
* Steps:
  + Process reaches the system.
  + If the required frames are available, they are allocated to it.
  + LOOP: Process page is loaded into one frame.
  + Frame number is put in the page table of that process.
  + End LOOP.

### Hardware support.

* Most OS allocate a page table for each process.
* Process control block: stores a pointer to the page table + other register values.
* Dispatcher reloads the user registers & defines the correct page table.

### Protecting memory.

* valid-invalid bit: An additional bit is added to each entry in the page table.
  + Valid bit indicates that its page is in the process’s logical address space.
  + A picture containing text, screenshot, diagram

    Description automatically generatedInvalid bit indicates that its page isn’t in the process's logical address space (illegal address).
  + The OS uses this bit to allow/disallow access to pages.

### Shared pages.

* Pages can share common code.
* Important for time sharing environments.
* Reentrant code: non-self-modifying code.
  + Can be shared because it can’t change during execution.
* Pages don’t share data || registers.
* Saves space.

## Segmentation.

* A memory management scheme that supports the user’s view of memory.
  + A picture containing text, diagram, circle, font

    Description automatically generatedUser’s view: memory is a collection of variable-sized segments (no ordering).
* Logical address space: collection of segments.
* Segments have names & lengths (offset).
* User specifies addresses by a name & an offset.

### Segmentation’s Basic implementation.

* Its table consists of base & offset.
* Physical address = byte number + base.
  + Segment 0, byte 400 = 400 +1400 = 1800.
  + Segment 1 byte 0 = 0 + 6300 = 6300.
* Trap address: an address that exceeded its segment limit.
  + A screenshot of a computer

    Description automatically generated with low confidenceSegment 0, byte 1222 = 1222+1400 = 2622 (> 2400).