**Operating System Principles: NOTES**

**Week 1 – Preliminary**

* Computer System Structure
  + 4 Components
    - Hardware
      * Provides basic computing resources
        + E.g. CPU, memory, I/O Devices
    - Operating System
      * Controls and coordinates use of hardware among various applications and users
    - Application programs
      * Define the way in which the system resources are used to solve the computing problems of the user
        + E.g. work processors, compilers, web browsers, database systems, video games
    - Users
      * + E.g. People, machines, other computers

A close up of a piece of paper

Description automatically generated

* Operating System Definition
  + **OS** is a resource allocator
    - Manages all resources
    - Decides between conflicting requests for efficient and fair resource use
  + **OS** is a control program
    - Controls execution of programs to prevent errors and improper use of the computer
* Operating System Operations
  + Interrupt driven
  + **Interrupts**
    - Signals sent to the CPU by external devices, normally I/O devices. They tell the CPU to stop its current activities and execute the appropriate part of the operating system.
    - Hardware interrupt done by one of the devices
    - Software interrupt (hardware and software)
      * Software error (E.g. divide by 0)
      * Request for operating system services
      * Other process problems including infinite loops, processes modifying each other or other OS
    - Functions of interrupts
      * Transfer control of the interrupt service router, through the interrupt vector, which contains addresses of all service routines
      * Interrupt architecture must save the address of the interrupted instruction
    - The OS preserves the state of the CPU by storing registers and the program counter of the interrupted process

A screenshot of a cell phone

Description automatically generated

* + User mode vs Kernel Mode

A screenshot of a cell phone

Description automatically generated

* + - Mode bit used to define mode
    - Dual-Mode operations allow OS to protect itself and other system components
* Operating System Services
  + OS’ provide an environment for execution of programs and services to programs and users
  + One set of operating system-services provides functions that are helpful to the user

E.g.

* + - User Interfaces
      * Almost all operating systems have a user interface
        + Varies between Command-Line Interface (CLI), Graphical User Interface (GUI), Batch
    - Program execution
      * The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)
    - I/O Operations
      * A running program may require I/O, which may involve a file or an I/O device
    - File-System manipulation
      * Programs need to read and write files and directories, create and delete them, search them, list file information, permission management
    - Communications
      * Processes may exchange information, on the same computer or between computers over a network
        + Communications may be via shared memory or through message passing (packets moved by the OS)
    - Error detection
      * OS needs to be constantly aware of possible errors
        + Error detection may occur in the CPU and memory hardware, in I/O devices, in user programs
        + For each type of error, OS should take appropriate action to ensure correct and consistent computing
        + Debugging facilities can greatly enhance the users’ and programmers’ abilities to efficiently use the system
  + Another set of operating system-services exists for ensuring the efficient operation of the system itself
    - Resource allocation
      * When multiple users/jobs are occurring concurrently, resources must be allocated to each of them

E.g. Resources

* + - * + CPU Cycles
        + Main memory
        + File storage
        + I/O devices
    - Accounting
      * To keep track of which users use how much and what kind of computer resources
    - Protection and security
      * Owner’s self-control information in a multi-user system
      * Concurrent processes should not interfere with each other
        + Protection

Ensuring that all access to system resources is controlled

* + - * + Security

Secure against outsiders, need for user authentication. Extends to defending external I/O devices from invalid access attempts

A screenshot of a cell phone

Description automatically generated

* System Calls
  + Programming interface to the service provided by the OS (programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on)
  + Typically written in high-level languages (C, C++)
  + Mostly accessed by programs via a high-level API rather than direct system call use
  + Example
    - System call sequence to copy the contents of one file to another file

A screenshot of a cell phone

Description automatically generated

* + API Example

A screenshot of a cell phone

Description automatically generated

* + System Call Implementation
    - Typically, a number associated with each system call
      * System call interface
        + Maintains a table indexed according to these numbers
    - The system call interface, invokes the intended system call in OS kernel and returns the status of the system call as well as any return values from it
    - The caller just needs to obey the API and understand what the OS will do as a result of the call

A close up of a logo

Description automatically generated

* + - Types of system calls
      * Process control
        + Create/terminate process
        + End
        + Abort
        + Wait
        + Allocate/free memory
        + Debug
        + Lock
      * File management
        + Create/delete file
        + Open/close file
        + Read/write/reposition file
        + Get/set file attributes
      * Device management
        + Request/release device
        + Read/write/reposition
        + Get/set device attributes
        + Logically attach/detach devices
      * Information maintenance
        + Get/set date/time
        + Get/set system data
        + Get/set processes/file/device attributes
      * Communications
        + Create/delete communication connection
        + Send/receive messages
        + Create and gain access to memory regions
        + Transfer status information
        + Attach/detach remote devices
      * Protection
        + Control access to resources
        + Get/set permissions
        + Allow/deny user access

A screenshot of a cell phone

Description automatically generated

* System Programs
  + Provide a convenient environment for program development and execution. Can be divided into:
    - File manipulation
      * Text editors to create/modify files
      * Special commands to search/manipulate text in files
    - Status information sometimes stored in a file modification

E.g.

* + - * + System information – date/time/amount of available memory/disk space/number of users
        + Others provide performance/logging/debugging information
    - Programming language support
      * Compilers/assemblers/debuggers/interpreters
    - Program loading and execution
      * Absolute loaders, relocatable loaders, linkage editors, overlay-loaders. Debugging systems for higher-level and machine language
    - Communications
      * Provide mechanism for create virtual connections among processes/users/computer systems
        + E.g. From machine to machine

Send messages

Browse web pages

Send emails

Log in remotely

Transfer files

* + - Background services
      * Launch at boot time
      * Provide dis checking/process scheduling/error logging/printing
      * Run in user context, not kernel context
      * Known as services/subsystems/daemons
    - Application programs
      * Don’t pertain to system
      * Run by users
      * Not considered a part of the OS
      * Launched by command line/mouse click
  + System programs often define the user’s view of the OS, not the actual system calls they are performing
  + Some system programs are UIs to system calls, some are more complex
  + File management
    - Create/delete/copy/rename/print/dump/list (manipulate) files and directories
* Operating system design and implementation
  + Design and implementation of OS not solvable but approaches have been proven successful
  + Internal structure of OS varies
  + Design starts with defined goals and specifications
  + Goals
    - User goals
      * Convenient usage of OS
      * Easy to learn
      * Reliable
      * Safe
      * Fast
    - System goals
      * Easy to design/implement/maintain OS
      * Flexible
      * Reliable
      * Error-free
      * Efficient
  + OS Design and Implementation principles
    - **Policy: *What*** will be done?
    - **Mechanism: *How*** to do it?
  + The separation of policy from mechanism is a very important principle
    - Policies are likely to change across places or over time. If not separated, each change in policy would require a change in the underlying mechanism. If separated, a general mechanism insensitive to changes in policy would be more desirable. A change in policy would then require redefinition of only certain parameters of the system.
  + Much variation
  + Usually a mix of languages (low levels in assembly, main body is C, System programs in C/C++/PERL/Python etc.)
  + High level languages easier to port to other hardware but slower
  + Emulation can allow OS to run on non-native hardware

**Week 2 – Memory Management**

* Address Binding
  + Programs on disk, ready to be brought into memory to execute
    - First address space is 0000 but first address of user process need not be 0000
  + In most cases, a user program will go through several steps before executing

A screenshot of a cell phone

Description automatically generated

* Logical vs physical address space
  + Logical
    - Generated by the CPU
  + Physical
    - Address seen by the memory unit
* Contiguous Allocation
  + Main memory divided into two partitions
    - One for OS
    - One for user process (several can reside in memory at the same time)
  + How to allocate available memory to the processes to be brought into memory
    - In contiguous memory allocation, each process contained in single contiguous section of memory
  + Relocation registers used to protect user processes from each other, and from changing OS code and data
    - Base registers contain value of the smallest physical address
    - Limit register contain range of logical addresses (all logical addresses must be less than the limit address)

A close up of a map

Description automatically generated

* + OS maintains information about
    - Allocated partitions
    - Free partitions
  + When partitions are freed, adjacent free partitions are combined
  + First Fit
  + Best Fit
  + Worst Fit
  + External Fragmentation
    - Total memory space exists to satisfy request, but not contiguous
* Paging
  + A memory management scheme which allows physical address space of a process to be non-contiguous
    - Divide physical memory into fixed-sized blocks called frames
      * The size of a page is typically a power of 2, between 512 bytes and 16 megabytes
    - Divide logical memory into blocks of same size called pages
      * Backing store likewise split into pages
  + For each process, set up a page table to translate logical to physical addresses, page table is kept in main memory
  + Contiguous allocation
    - Page-table base register (PTBR)
      * Points to the page table
    - Page-table length register (PTLR)
      * Indicates size of page table
    - Translation look-aside buffers (TLBs)
      * Fast look up hardware cache
    - Structure of Page Table – Two-Level Paging
      * Number of entries = size of the logical address space / page size (32-bit logical address, page size is 4 KB , 232/212)
      * 4 bytes of logical memory equates to 4 MB of physical memory in the page table
      * Two level
        + Logical memory address on a 32-bit machine with 4K page size =

Page number of 20 bits 220=232/212

Page offset of 12 bits (212)

* + - * Since the page table is paged, the page number is further divided into
        + 10-bit page number
        + 10-bit page offset
      * Thus a logical address would be

|  |  |  |
| --- | --- | --- |
| **Page Number** | | **Page Offset** |
| P­1 | P2 | d |
| 10 | 10 | 12 |

* + - Virtual Memory – swapping, page fault
  + Methods
    - Keep track of all free frames
    - To run a program of size N pages, need to find N free frames and load program
      * Set up a page table to translate logical to physical

A screenshot of a cell phone

Description automatically generated

* + Address generated by CPU is divided into:
    - Page number (p)
      * Used as an index into a page table which contains base address of each page in physical memory
    - Page offset (d)
      * Combined with base address to define the physical memory address that is sent to the memory unit
      * For given local addresses of space 2m and page size 2n

A picture containing screenshot

Description automatically generated

* + Calculating internal fragmentation
    - Page size = 2048 bytes
    - Process size = 72766 bytes
    - 35 pages + 1086 bytes
    - Internal fragmentation of 2048 – 1086 = 962 bytes
  + Internal fragmentation
    - Worst case fragmentation = 1 frame – 1 byte
    - On average fragmentation 1 / 2 frames

**Week 3 – Mass Storage**

* Mass Storage
  + Magnetic Disks
    - Provide the bulk of secondary storage of modern computers
    - Drives rotate at 60-250 times per second
    - Transfer rate is the rate at which data flows between the drive and the computer
    - Positioning time (random-access time) includes
      * Seek time
        + Time to move the disk arm to desired cylinder
      * Rotational latency
        + Time for desired sector to rotate under the disk head
  + Solid-State Disks
    - Non-volatile memory used like a hard drive
    - Can be more reliable than HDDs
    - Less capacity
    - Much faster
    - More expensive per MB
    - Shorter life span
      * Busses can be too slow, so connect directly to PCI
    - No moving parts so no seek time or rotational latency
* Disk Scheduling
  + Related to seek time
  + The operating system is responsible for using hardware efficiently
  + For disk drives, this mean having fast access time, and large disk bandwidth
    - Minimise seek time
    - Disk bandwidth
      * Total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer
  + Whenever a process needs I/O to from the disk, it issues a system call to the OS. The request specifies:
    - Input or output
    - Transfer’s disk address
    - Transfer’s memory address
    - Number of sectors to be transferred
  + Request might need to wait for desired disk drive or controller
  + Driver controllers have small buffers to manage I/O request queues
  + FCFS (First-come First-Served)
  + SSTF (Shortest-seek-time-first)
    - Shortest seek time from the current head position
  + SCAN
    - Disk arm starts at one end of the disk and moves towards the other end
    - Servicing requests until it gets to the end
    - Then reverses
  + C-SCAN (Circular Scan)
    - Provides a more uniform wait time than scan
    - At the end of the disk, it goes back to the start and goes again
  + Look
    - Version of SCAN
    - Arm only goes as far as the last request in each direction
    - Then reverses
    - Doesn’t go to the end of the disk
  + C-Look
    - Version of C-SCAN
    - Arm only goes as far as the last request in each direction
    - Then goes back to the start
    - Doesn’t go to the end of the disk
* RAID Structure
  + Redundancy Array of Inexpensive Disks
    - Multiple disks drives provide reliability via redundancy
    - Increases the mean time to failure
  + RAID 0
    - No redundancy striping
  + RAID 1
    - Has a copy of all disks
  + RAID 5
    - Blocks are interleaved with the parity
  + RAID 10
    - Data split across 2 drives, both have copies

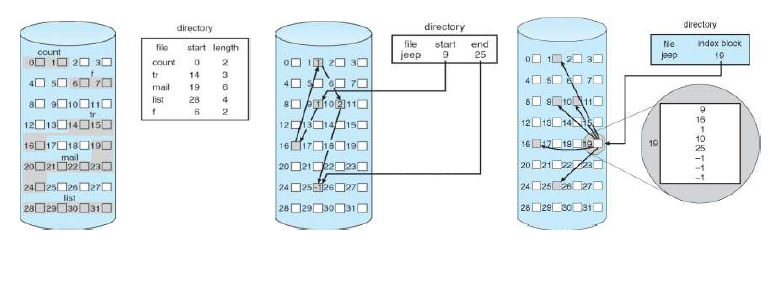
**Week 4 – File System**

* Access Methods
  + Sequential access
    - Read\_next()
      * Reads the next portion of the file and advances file pointer
    - Write\_next()
      * Appends to the end of the file and advances to the file end
    - Reset()
      * To the beginning of the file
  + direct-access file – for fixed length records
    - read(n)
    - write(n)
* Disk and Directory Structure
  + Partitions
    - Disks can be subdivided into minidisks/slices
      * Each can hold a file system, so multiple file system types on the same device
      * Leaving part of the device for other uses
        + Swap
        + Unformatted disk space
      * Disk or partitions can use RAID
  + Volume
    - Entity containing a file system
    - Can be thought as a virtual disk:
      * Subset of a device
      * Whole device
      * Multiple devices linked into a RAID set
  + Single-Level Directory
    - For all users, all files in the one spot
    - Has a naming problem – must have unique file names
    - Has a grouping problem – hard to remember all names
  + Two-Level Directory
    - Separate directory for every user
    - Path names allow different users to have the same file name
    - Efficient searching
    - No grouping capability
  + Tree-Structured Directory
    - Allow each user to have sub directories
    - Efficient searching
    - Grouping capability
    - Has a current directory
  + Acyclic-Graph Directory
    - Has shared subdirectories and files
    - Graph with no cycle
    - Shared subdirectory and file have one copy
    - Deletion problems
      * When can the space allocated to a share file be deallocated and reused
        + Can leave dangling pointers
  + General Graph Directory
    - Acyclic graph structure has hard times preventing cycles
    - Avoid the possible infinite loop by limiting the number of directories that will be accessed during a search
* File system is generally composed of many different levels

A screenshot of a cell phone

Description automatically generated

* Allocation Methods
  + Contiguous
  + Linked
  + Indexed



**Week 5 – I/O System**

* I/O Systems
  + How does the processor give commands and data to a controller to accomplish an I/O transfer?
  + Devices have registers
    - Typically
      * 1-4 bytes long
      * FIFO buffer
    - Data-in register
      * Read by the host to get input
    - Data-out register
      * Written by the host to send output
    - Status register
      * Read by the host to indicate the status such as current commands completed
    - Control register
      * Read by the host to start a command and change mode of device (E.g. start parity check)
  + Device drivers communicate with registers in 2 ways
    - Direct I/O instructions
      * Use instructions to specify the transfer of a byte or word to an I/O port address
      * Triggers bus lines to select proper device and move bits into or out of a device register
    - Memory-mapped I/O
      * Device control registers are mapped into the address space of the processor
      * CPU executes I/O requests using the standard data-transfer instructions to read and write the device control-registers
  + Direct memory access is used to avoid programmed I/O (one byte at a time) for large data movements
    - Direct virtual memory access
* Interrupts
  + CPU has a wire called interrupt-request line
    - Causes
      * Software error (divide by 0)
      * Request for OS service
      * Other processor problems (infinite loops)
    - Handling
      * Preserves state of CPU
        + Stores registers and program counter of interrupted process
      * Transfers control to interrupted process, through interrupt vector (has the addresses of all service routines)

**Week 8 – CPU Scheduling**

* CPU Scheduling
  + Short term schedulers select from the process in the ready queue, and allocates CPU time to one of them (queue can be ordered in many ways)
  + Dispatcher modules
    - Gives control of CPU to the processor selected by the short-term schedulers
    - Dispatch latency
      * Time taken to respond to the request given by the module
  + Scheduling algorithm optimisation criteria
    - Max CPU utilization
      * Keep it as busy as possible
    - Max Throughput
      * Number of processes completed per time unit
    - Min turnaround time
      * Time to execute process
    - Min waiting time
      * Amount of time in ready queue
    - Min response time
      * Time taken from response submission to response
* Scheduling Algorithms
  + Scheduling all processes currently in ready state when CPU is free
    - FSFC – non-pre-emptive
    - Shortest Job First – non-pre-emptive
    - Priority scheduling – non-pre-emptive
  + Scheduling all processes currently in ready state when the CPU is free **OR** a new process arrives
    - Shortest remaining time first – pre-emptive
    - Priority scheduling – pre-emptive
  + Scheduling periodically
    - Round robin – pre-emptive
      * Only given a small time on the CPU
* Gantt Chart
  + Measure waiting time and turnaround time
* Queue
  + Partitioned into:
    - Multi-level queue
      * All tasks have priorities based on process type
      * 1. System processes
      * 2. Interactive processes
      * 3. Batch processes
      * 4. Student processes
      * Ready queue partitioned:
        + Foreground (interactive)

RR scheduling algorithm

* + - * + Background (batch)

FCFS scheduling algorithm

* + - Multi-level feedback queue
      * Processes can be moved between queues
      * Q0 – RR algorithm with quantum 8 milliseconds (moved to Q1 if unfinished)
      * Q1  - RR algorithm with quantum 16 milliseconds (moved to Q2 if unfinished)
      * Q2 – FCFS

**Week 12 – From Practice Exam**

* Zombie process
  + Terminated but parent has not called wait()
  + All processes turn to zombies when terminated, for brief time, until wait() is called and the process and its entry in the process table are released
* Long term schedules should strive for process mix
  + For CPU utilization and device utilization
    - I/O bound processes spend a lot of time on I/O, many short CPU bursts
    - CPU-bound processes spend a lot of time on computation, few very long CPU bursts
* Concurrency without parallelism
  + Concurrency – more than one processes or thread is progressing at the same time
  + Parallelism – processes running simultaneously on multiple cores
* Desirable solution to the critical-section problem – trying to access shared resources
  + Mutual exclusion
    - Only 1 process in the critical section at a time
  + Process
    - If the critical section is free and processes are waiting, allow new entry
  + Bounded waiting
    - Process can only wait a bounded amount of times other processes enter the critical section before it
* Resource Pre-emption
  + To eliminate deadlocks
  + Pre-empt some resources from processes and give these resources to other processes until the deadlock cycle is broken
  + Issues
    - Select a victim for deadlock/pre-emption
    - Rollback
      * Must revert the stuck process to a safe state and restart
    - Starvation
      * The same process might be chosen as the victim multiple times

**Week x – Assignment Work**