Mass-Storage Structure (ch11):

**Magnetic Disks** provide bulk of secondary storage of modern computers. as a sector-addressable address space

**Disk drives** are addressed as large 1-dimensional arrays of logical blocks

Transfer Rate –theoretical –6 Gb/sec

Effective Transfer Rate –real –1Gb/sec

Seek time from 3ms to 12ms –9ms common for desktop drives

Average seek time measured or calculated based on 1/3 of tracks

Latency based on spindle speed1 / (RPM / 60) = 60 / RPM

Average latency = ½ latency

**Access time has two major components**: **Seek time** is time to move the heads to the cylinder containing the desired sector•**Rotational latency** is additional time waiting to rotate the desired sector to the disk head

**I/O request includes** input or output mode, disk address, memory address, number of sectors to transfer

•OS maintains queue of requests, per disk or device

•Idle disk can immediately work on I/O request, busy disk means work must queue

SSTF and LOOK is common and has a natural appeal

•SCAN and C-SCAN perform better for systems that place a heavy load on the disk(Less starvation)

-----------------------------------------------------------------------------------------------------------------------------------------------------------------------

Ch12

**Architecture of I/O Systems**-Key components: **System bus**: allows the device to communicate with the CPU, typically shared by multiple devices •**A device port typically consisting of 4 registers**:•Status-indicates a device busy, data ready, or error condition•Control: command to perform•Data-in: data being sent from the device to the CPU•Data-out: data being sent from the CPU to the device; **Controller**: receives commands from the system bus, translate them into device actions, and reads/writes data onto the system bus•**The device itself**

**I/O Services Provided by OS:** •Naming of files and devices.•Access control•Operations appropriate to the files and devices•Device allocation•Buffering, caching, and spooling to allow efficient communication with devices•I/O scheduling•Error handling and failure recovery associated with devices•Device drivers to implement device-specific behaviors

**Communication using Polling**•CPU busy-waitsuntil the status is idle•CPU sets the command register and data-out if it is an output operation•CPU sets status to command-ready => controller sets status to busy•Controller reads the command register and performs the command, placing a value in data-in if it is an input command•If the operation succeeds, the controller change the status to idle•CPU observes the change to idle and reads the data if it was an input operation•Good choice if data must be handled promptly, like a modem or keyboard•What happens if the device is slow compared to the CPU?

**Communication using Interrupts**•Rather than using busy waiting, the device can interrupt the CPU when it completes an I/O operation•On an I/O interrupt:•Determine which device caused the interrupt•If the last command was an input operation, retrieve the data from the device register•Start the next operation for that device•Interrupt mechanism also used for exceptions•Terminate process, crash system due to hardware error•Page fault executes when memory access error•System call executes via trap to trigger kernel to execute request•Multi-CPU systems can process interrupts concurrently•If operating system designed to handle it•Used for time-sensitive processing, frequent, must be fast

**Direct Memory Access**•Used to avoid programmed I/O(one byte at a time) for large data movement•Solution: Direct Memory Access(DMA)•Use a sophisticated DMA controller that can write directly to memory. Instead of data-in/data-out registers. It has an address register.•The CPU tells the DMA the locations of the source and destination of the transfer.•The DMA controller operates the bus and interrupts the CPU when the entire transfer is complete, instead of when each byte is ready.•Cycle stealing from CPU but still much more efficient•The DMA controller and the CPU compete for the memory bus, slowing down the CPU somewhat, but still providing better performance than if the CPU had to do the transfer itself.

**6 Steps of DMA:** Device controller is told to transfer disk data to buffer at address X → Device driver tells disk controller to transfer C(counter) bytes from disk to buffer at address X **→** Disk controller initiates DMA transfer → Disk controller sends each byte to DMA controller **→** DMA controller transfers bytes to Buffer X, increasing memory address and decreasing C until C=0 → when c =0 DMA interrupts CPU to signal transfer

**Each OS has its own I/O subsystem structures and device driver frameworks**

**•Devices vary in many dimensions**(•Character-stream or block•Sequential or random-access •Synchronous or asynchronous(or both)**•Most devices are asynchronous, while I/O system calls are synchronous** => The OS implements blocking I/O•Sharableordedicated•Speed of operation•read-write, read only, or write only

**I/O devices can be grouped by the OS into**(•Block I/O•Character I/O (Stream)•Memory-mapped file access•Network sockets)

**Block devices include disk drives(Commands include** read, write, seek •Raw I/O,direct I/O,or file-system access•Memory-mapped file access possible•File mapped to virtual memory and clusters brought via demand paging•DMA)

**Blocking-**process suspended until I/O completed•Easy to use and understand•Insufficient for some needs

**Nonblocking-**I/O call returns as much as available•User interface, data copy (buffered I/O)•Implemented via multi-threading•Returns quickly with count of bytes read or written•select() to find if data ready then read()or write()to transfer

**Asynchronous-**process runs while I/O executes•Difficult to use•I/O subsystem signals process when I/O completed

**Why buffer on the OS side?**•To cope with speed mismatches between devices•To cope with devices that have different data transfer sizes•To minimize the time a user process is blocked on a write

**Caching-**faster device holding copy of data•Always just a copy•Key to performance•Sometimes combined with buffering

**•Spooling-**hold output for a device•If device can serve only one request at a time •e.g., Printing

**-----------------------------------------------------------------------------------------------------------------------------------------------------------------------**

**Ch13&14- File System**

**Unstructured File:**Unix implements files as a series of bytes

**Structured File**:IBM mainframes implements files as a series of records or objects

**File attributes:** name, type, location, size, protection, creation time

**File Data Structures:1.Open file table**-shared by all processes with an open file. •open count •file attributes, including ownership, protection information, access times•location(s) of file on disk •pointers to location(s) of file in memory **2.Per-process file table** •for each file,•pointer to entry in the open file table•current position in file (offset) •mode in which the process will access the file (r, w, rw) •pointers to file buffer

**File operations: -Data operations:**•**Create**(), **Delete**(), **Open**(), **Close**(), **Read**(fileID,from,size,bufAddress)-random/direct access –OS reads “size” bytes from file position “from” into “bufAddress”; sequential access •OS reads “size” bytes from current file position, into “bufAddress” and increments current file position by size.**Write**(copies from buffer to file). **Seek**(updates fp) **-Naming operations:**•HardLink(), SoftLink(), Rename(), SetAttribute(), GetAtrribute()

**Access Methods**•Common file access patterns from the programmer's perspective •Sequential: data processed in order, a byte or record at a time. •Most programs use this method •Example: compiler reading a source file. •Direct: address a block based on a key value. •Example: database search, hash table, dictionary •Common file access patterns from the OS perspective: •Sequential: keep a pointer to the next byte in the file. Update the pointer on each read/write. •Direct: address any block in the file directly given its offset within the file.

**Single-Level Directory:** One name space for the entire disk, every name is unique. 1.Use a special area of disk to hold the directory. 2.Directory contains pairs. 3.If one user uses a name, no one else can. 4.Some early computers used this strategy. Early personal computers also used this strategy because their disks were very small. Has a naming and grouping **problem**.

**Two Level Directory:**Each user has a separate directory, but all of each user's files must still have unique names(•Path name•Can have the same file name for different user•Efficient searching•No grouping capability)

**Multilevel Directories -tree structured**(Unix, and all other modern operating systems). 1.Store directories on disk, just like files except the file descriptor for directories has a special flag bit. 2.User programs read directories just like any other file, but only special system calls can write directories. 3.Each directory contains pairs in no particular order. The file referred to by a name may be another directory. 4.There is one special root directory.•**Limitations** with basic tree structure •Difficult to share file across directories and users •Can’t have multiple file names

**Directory Operations:**•Search for a file: locate an entry for a file •Create a file: add a directory listing •Delete a file: remove directory listing •List a directory: list all files (ls command in UNIX) •Rename a file •Traverse the file system

**Directory Implementation**•**Linear** listof file names with pointer to the data blocks•Simple to program•Time-consuming to execute•Linear search time•Could keep ordered alphabetically via linked list or use B+ tree•**Hash Table**–linear list with hash data structure•Decreases directory search time•Collisions–situations where two file names hash to the same location•Only good if entries are fixed size, or use chained-overflow method

**Contiguous Allocation**•OS maintains an ordered list of free disk blocks•OS allocates a contiguous chunk of free blocks when it creates a file•Need to store only the start location and size in the file descriptor•**Advantages**•Simple –only starting location (block #) and length (number of blocks) are required•Access time? Number of seeks? (sequential and random access)•**Disadvantage**s•Changing file sizes•Fragmentation? Disk management?•External fragmentation, need for compaction

An **extent** is a contiguous block of disks•Extents are allocated for file allocation•A file consists of one or more extents

**Contiguous** great for sequential and random•**Linked** good for sequential, not random•Declare access type at creation -> select either contiguous or linked•**Indexed** more complex•Single block access could require 2 index block reads then data block read•Clustering can help improve throughput, reduce CPU overhead

Burst time is not a cpu scheduling criterion

Cpu scheduling algo’s are used for picking one of the ready processes in main mem to run next

To handle deadlocks, os most often pretend deadlocks never occur

Which of the following isnt usually stored in a twolevel pafe table? Virtual page number

An interrup is a signal that causes the control unit to branch from a specific location

When a process is created using the classiscal fork sys call, Process ID is not inherited by child process

Text segment of a process address space contains the executable code associated with the process

If each processor performs all tasks within the os, it would lead you to believe its SMP type

Embedded computers run on a real time OS

Message-passing model is easier to implement that shared mem model for inter-computer communication

For a single processor system, there will never be more than one running process.

Long term scheduling is performed on submitted jobs, when process must be moved from waiting to ready, on process in the ready queue.

A trap/exception is a software generated interrupt caused by an error

A situation where several processes access and manipulate the same data concurrently and the outcome of the exception depends on the particular order in which access takes place is called a race condition

The major difficulty in designing a layered os approach is making sure that each layer hides certain data structures, hardware, and operations.

The monitor construct ensures that only one process can be active at a time within the monitor

A thread control block has the identical structure as the PCB.

System calls can be run in either user mode or kernel mode F

Each thread of a process has its own virtual address space. F

All processes in unix first translate to a zombie process upon termination T

The main diff between test and set and use of semaphores is that semaphores require OS to do busy waiting rather than the user program F

Non preemptive scheduling algos are better for interactive jobs since they tend to favor jobs that require quick responses F

Fragmentation doesnt occur in a paging system F

A TLB miss could occur even though the requested page was in memory T

A deadlock free solution eliminates the possibility of starvation F

The code that changes the system clock runs in user mode F

A thread can be blocked on multiple condition variables simultaneously F

It is possible to have concurrency without parallelism T

Aging can alleviate the starvation problem of a low priority job T

Deadlock avoidance algos in general result in better utilization of resources since its used as much as possible

Paging may suffer from internal fragmentation while segmentation may suffer from external frag T

For machines with 32 bit addresses, since 4gb physical memories are common and cheap, virtual memory is really no longer needed. F

**Part 2-1:** Process concept, states and scheduling, operations, inter-process communication

Process execution must progress in sequential fashion

Program is passive, process is active (with a counter and resources)

Each process has its own address space: (1.code, 2.data section(global var), 3.Heap(dynamically allocated mem),4.stack(temp data - function param, return addr. Local var.))

**Process Context**: the program counter and CPU registers are part of this

**Process Control Block(PCB)**: OS data structures to keep track of all processes

•The PCB tracks the execution state and location of each process•The OS allocates a new PCB on the creation of each process and places it on a state queue•The OS deallocates the PCB when the process terminates. Gets read by os when popped out of queue.

The **PCB contains**:•Process state –running, waiting, etc.•Program counter –location of instruction to next execute•CPU registers –contents of all process-centric registers•CPU scheduling information –priorities, scheduling queue pointers•Memory-management information –memory allocated to the process•Accounting information –CPU used, clock time elapsed since start, time limits•I/O status information –I/O devices allocated to process, list of open files

**A uniprocessor system can have only one running process at a time**

**Schedulers:**

**Short-term scheduler(or CPU scheduler) –**selects which process should be executed next and allocates CPU•Sometimes the only scheduler in a system•Short-term scheduler is invoked frequently (milliseconds) (must be fast) Can be invoked from running to wait, running to ready, waiting to ready, terminated. •**Long-term scheduler (or job scheduler)** –selects which processes should be brought into the ready queue•Long-term scheduler is invoked infrequently (seconds, minutes) (may be slow)•The long-term scheduler controls the degree of multiprogramming.

**Processes can be described as: I/O-bound process** –spends more time doing I/O than computations, many short CPU bursts **OR CPU-bound process** –spends more time doing computations; few very long CPU bursts•Long-term scheduler strives for good process mix

**Context Switch**•When CPU switches to another process, the system must save the stateof the old process and load the saved state for the new process via a context switch•Contextof a process represented in the PCB•Context-switch time is overhead; the system does no useful work while switching•The more complex the OS and the PCB ➔the longer the context switch

Generally, process identified and managed via a **process identifier**

**fork()system call creates new process•exec()system call used after a fork()to replace the process’ memory space with a new program**

If no parent waiting (did not invoke wait()) process is a **zombie•**If parent terminated without invoking wait, process is an **orphan.**

**Shared Memory:** communication is under the control of the users processes not the operating system. Major issues is to provide mechanism that will allow the user processes to synchronize their actions when they access shared memory. Can work on things at the same time rather than one at a time. Fixes concurrency iss.

**Message Passing:** communicate and to synchronize their actions. processes communicate with each other without resorting to shared variables. Needs a **communication link: (direct)**A link is associated with exactly one pair of communicating processes•Between each pair there exists exactly one link•The link may be unidirectional, but is usually bi-directional. Either **Blocking(synchronous)(**sender is blocked till message received. Receiver blocked until message avail**.)** or **Non-Blocking(asynchronous)(**sender send mssg. And continue)(receiver receives valid or null mssg**)**: