Operating Systems Final Review

**Chapter 1**

* Each process has an address space that contains executable program
* **Virtual memory -> the OS keeps part of address space in main memory and part of it on disk; gives the ability to run programs larger than the machine’s physical memory by moving pieces back and forth between RAM and disk**
* Operating System -> code that carries out system calls
* Monolithic systems: - entire OS runs as a single program in kernel mode
* all procedures compiled and then linked into an executable program
* has a main program that carries out service procedures; service procedures carry out system calls; utility procedures help service procedures
* for each system call, there is one service procedure that will execute it
* Microkernels: -split up the OS into small modules
* the microkernel runs in kernel mode, and all others run as user processes
* advantage: if one process crashes, does not interfere with other processes
* outside the kernel, 3 layers of user level processes (drivers, servers, which do most work on OS, and user programs)
* mechanism and policy: mechanism for doing something in kernel, but policy made in user- level
* Client-Server: - 2 classes of processes (clients and servers)
* servers provide service and clients use services
* clients send message to servers, telling what it wants, and server does the work and returns answer
* clients and servers do not need to be on same machine, since clients send messages to server and does not care how or where messages are read, as long as server replies back
* Virtual Machines: - users wanting to work interactively at a terminal
* VM/370: timesharing system that separates multiprogramming from extended machine with more convenient interface than hardware
* Virtual machine monitor runs on bare hardware and does multiprogramming, and provides several virtual machines
* Virtual machines are exact copies of bare hardware and can run several operating systems that run on bare hardware
* Different virtual machines can run different operating systems
* Type 1 hypervisor: has no host operating system and must perform functions by itself, manage storage on raw disk partitions
* Type 2 hypervisor has host operating system and uses it to create processes, store files on, etc.; reads installation CD-ROM for chosen guest operating system, and installs guest OS on virtual disk (big file in host operating system’s file system)
* Exokernel: allocates resources to virtual machines and checks attempts to use them and make sure no virtual machine is using someone else’s resources; keep track of which resources belong to which virtual machine

**Chapter 2: Processes and Threads**

Processes:

* **Pseudoparallelism -> illusion of parallelism when one CPU works on several processes in a certain amount of time**
* **Multiprocessor systems -> have 2 or more CPUs sharing same physical memory**
* **Process -> an instance of executing a program; each has own virtual CPU**
* **Multiprogramming -> switching back and forth from program to program**
* **Difference between program and process: program is a set of instructions to perform some task, and process follows instructions (execution) and gets resources; a program running twice counts as 2 processes**
* **4 events that cause processes to be created:**

1. **System initialization**
2. **Running process issuing system calls**
3. **User requests to start a process**
4. **Initiation of a batch job (OS creating a process and running a job)**

* **Parent and child process each have own distinct address space (child’s address space is a copy of parent’s, but both are separate)**
* **Process termination:**

1. **Normal exit (voluntary)**
2. **Error exit (voluntary)**
3. **Fatal error (involuntary)**
4. **Killed by another process (involuntary)**

* ***Init* is a process that is created when system is booted**
* **Processes can be in a hierarchy (ex: a group of processes can receive a signal, but each processes can do something different with that signal)**
* **Process states:**

1. **ready**
2. **running**
3. **blocked (must wait for external event to happen before able to run)**

* **Process table -> contains entries (1 per process) called process control blocks that contains info about process state, memory, status of open fiels, etc.**
* **Processes can be interrupted many times, but will return to same state as they were before the interrupt(s) occurred**
* **Each process has an address space and a single thread of contro**l

Threads:

* Mini-processes that share an address space
* Benefits: - can use when there are multiple activities going on at once

- easier to create and destroy than processes

speed up applications

* Cache - > collection, stored in main memory, of heavily used things
* Finite state machine -> each computation has a saved state and events can occur that can cause the state to be changed
* Multithreading -> allowing multiple threads in the same process
* Each thread has its own stack that contains info about procedures that have not yet finished
* Threads can create other threads
* Pthread functions (portable threaded programs):

- pthread\_create -> create new thread; returns thread identifier

- pthread\_exit -> terminate calling thread

- pthread\_join -> wait for specific thread to exit

- pthread\_yield -> release CPU to let another thread run

- pthread\_attr\_init -> create and initialize a thread’s attribute structure

- pthread\_attr\_destroy -> remove a thread’s attribute structure

\*must include pthread.h

* **Implementing threads in user space:**

**- kernel does not know about threads**

**- each process has its own thread table**

**-disadvantage: if a thread starts running, no other thread can run until the running thread gives up the CPU**

**- advantage: allows each process to have its own customized scheduling algorithm**

**-run-time system keeps running threads from its own process until kernel takes CPU away**

* **Implementing threads in kernel:**

**- kernel has a thread table, so threads make a kernel call and update table**

**-when a thread blocks, the kernel can run a thread from the same process or a thread from a different one**

**- advantage: kernel can easily check to see if a process has any other runnable threads if one thread causes a page fault**

* **Scheduling:**

**- compute-bound -> long CPU bursts (not much I/O waits)**

**- I/O bound processes -> short CPU bursts (frequent I/O waits)**

**- nonpreemptive scheduling algorithm -> pick a process to run and let it run until it blocks or voluntarily release CPU**

**- preemptive -> pick a process and let it run for a maximum amount of time**

**- fairness of CPU time is important**

**- policy enforcement -> seeing that the stated policy is carried out**

**- balance -> keep all parts of system busy**

**- turnaround time -> how long user waits for output**

* Categories of Scheduling:

1. Batch -> ex: inventory, interest, etc.; reduces process switching; throughput (maximize jobs per hour), keep CPU busy all the time
2. Interactive -> preemption is used (ex: servers); general purpose; respond to requests quickly and meet user expectations
3. Real Time -> preemption not always needed because the processes usually work quickly and may not run for a long time; meet deadlines and meet user expectations

- Batch System Scheduling:

* First-Come First-Served: processes assigned in order they requested it; single queue of ready processes; a blocked process that becomes ready is put at end of queue

- Advantage: easy to program and understand

- Disadvantage: can slow down performance for other processes

* Shortest Job First: scheduler picks shortest job first to run
* Advantage: Faster turnaround time for short jobs
* Disadvantage: Larger jobs have to wait longer
* Shortest Remaining Time Next: scheduler picks the job that has the shortest remaining time next

-Advantage: shortest jobs have good turnaround time

- Disadvantage: runtime needs to be known in advance

- Interactive Systems Scheduling:

* Round-Robin Scheduling: each process assigned a quantum (time interval); process put at end of list when it uses up quantum; assumes each process is equally important
* Advantage: easy to implement; only needs to maintain a list of runnable processes
* Disadvantage: length of quantum; some time will be wasted switching to processes if quantum time is too short; if quantum time is too long, poor response to short requests
* Priority Scheduling: each process assigned a priority and the one with highest priority is allowed to run; each process may also be assigned a maximum time quantum; can also have priority groups where each process in the higher priority run for a quantum, round-robin style, and the next group will run, and so forth; \*\*highest priority means the process with the highest number (ex: process w/ priority 4 runs before process w/ priority 3)
* Guaranteed Scheduling: scheduling guarantees good performance for processes; keeps track of how much CPU time each process has had since creation; process with lowest ratio runs next
* Advantage: fair performance
* Disadvantage: difficult to implement
* Lottery Scheduling: give lottery tickets for various system resources; ticket chosen at random, and process holding that ticket gets resource; more important processes can get more tickets to increase chance of winning; processes that are about to block can give other processes tickets so that they will have high chance of running next
* Advantage: very responsive (processes get CPU time depending on how many tickets), easier to implement
* Disadvantage: ??
* Fair-Share Scheduling: keeps track of which user owns which process; each user is allocated some fraction of the CPU no matter the number of processes

- Real-Time Scheduling:

* Deadlines must be met for processes
* Real-time system schedulable if (seconds of CPU time) / (period of time for process) <= 1
* Periodic -> occur at regular intervals

**Chapter 3: Memory Management**

* Page -> fixed sized unit in virtual memory
* Page frames -> corresponding unit in physical memory
* Page fault -> memory not in main memory and must be retrieved on the disk; OS chooses a page to remove and make room for incoming page

Page Replacement Algorithms:

* Optimal: each page is labeled with number of instructions that will be executed before that page is first referenced; page with highest label should be removed
* Advantage: it is optimal
* Disadvantage: cannot be implemented; don’t know when pages will be referenced
* Not Recently Used: uses R and M bits (R = reference bit, M = write bit, set when page is written to / modified); each page has R set to 0 initially; OS divides pages into 4 categories (class 0: not referenced, not modified; class 1: not referenced, modified; class 2: referenced, not modified, class 3: referenced, modified); removes a page at random from lowest numbered nonempty class
* Advantage: easy to understand
* Disadvantage: performance not optimal
* FIFO: list of all pages in memory with most recent at tail and least recent at head; on page fault, page at head is removed and new page added to tail

-Disadvantage: Don’t know if the oldest page is being heavily used

* Second Chance: modification of FIFO; inspects the R bit of oldest page -> if R = 0, page has not been referenced, and is replaced; if R = 1, page was referenced, R will be cleared, and page is put at end of list to be given another chance; must look at load time of each page to decide which page to remove
* Advantage: better than FIFO since it regards recently used pages
* Disadvantage: constantly moves pages around when R bits are set
* Clock: similar to second chance, but all page frames are in a circular list where the hand points to the oldest; R bit is checked, if R = 0, page is replaced and hand advances one position, if R = 1, bit is cleared and hand advances to next position; process continues until page with R = 0 is found
* Advantage: pages do not need to move as in second chance
* Least Recently Used: when a page fault occurs, replace the page that has not been used in the longest time, will look at counter of each page, and one with lowest counter is replaced
* Disadvantage: difficult to implement, needs special hardware to be used to have a 64 bit counter
* No Frequently Used: each page has a software counter initialized to 0; counter keeps track of how often pages are referenced; when page fault occurs, page with lowest counter is replaced

Design Issues:

* Local algorithms: allocate every process a fixed fraction of the memory
* Global algorithms: dynamically allocate page frames among the runnable processes
* Thrashing: when a program has page faults every few instructions
* Some page replacement algorithms can work with local or global algorithms
* **Page fault frequency algorithm -> tells when to increase/decrease process’s page allocation; does not say which page to replace on fault; page faults decrease as more pages are assigned**
* Load control: swap processes from memory to disk and free all page frames to reduce the number of processes competing for memory and try to stop thrashing
* Page size: may need smaller page size to reduce internal fragmentation (wasting space in pages), but small pages require many pages, which means a larger page table
* Separate instructions and data space: most computers have single address space for instructions and data; if address space is too small, may not be able to fit both; could have separate address spaces (one for instructions and one for data) and each one can have its own page table
* Shared pages: sharing pages is more efficient than having 2 copies of each page; not every page is sharable; if two processes are sharing some pages, if one process is removed from memory and its pages are removed, the second process will have many page faults; share read-only pages; once a page is modified, then the processes will be given copies of pages for writing
* Shared libraries: have the shared library loaded only once for all programs; only functions that have been called with be brought into RAM page by page, not entire library; use position-independent code (not using specific addresses in functions so that all processes can use those functions)
* Mapped files: a process can issue a system call to map a file onto its virtual address space
* Cleaning policy: make sure to have enough empty page frames; paging daemon (a background process) will check memory for free page frames, and use a replacement algorithm when there are not enough empty ones
* Virtual memory interface: allow for processes to share memory

**Chapter 4: File Systems**

Files:

* Store information on disk and read it back later
* Specific number of characters allowed for file name depending on system
* File extensions give type of file (ex: .c file = source file, .txt = plain text file, etc.)
* File Structure:

- Files are sequence of bytes

- Can also be sequence of records

- Can also be a tree structure

* File Types:

- Regular files -> stores user information; can be ASCII or binary

- Directories -> maintain file structure

- Character special files -> used to model serial I/O devices such as printers, terminals, and networks

- Block special files -> used to model disks

* File Access:

- Sequential access -> read all bytes/records in order; can be read as often as needed; set position of where to start reading

- Random-access -> bytes/records read in any order

* File Attributes:

- Also called metadata

- Information about a file such as permissions, date accessed, date modified, etc.

* File Operations:

- Create, delete, open, close, read, write, append, seek, get attributes, set attributes, rename

Implementing Files

* Contiguous allocation: each file stored as contiguous disk blocks
* Advantage: easy to implement and easy to find file blocks (just need address of first block for file and number of blocks in the file); good read performance since it is sequentially read
* Disadvantage: Disk becomes fragmented when files get deleted; file size needs to be specified to fill the holes in the disk
* Linked List Allocation: each file is a linked list of disk blocks; first word of each block used as a pointer to next block and rest of the block is for data
* Advantage: No space is lost due to fragmentation; every block can be used
* Disadvantage: Random access is very slow because must go through every block
* Linked List Allocation Using a Table in Memory: similar as above except the pointer word from each block is put into a table in memory, where each table entry will contain the number of the disk block to be pointed to next; a special marker is used to state the end of the chain
* Advantage: random access is much easier and space in each block is not wasted because of the pointers
* Disadvantage: entire table must be in memory all the time, which can take up a lot of space if the files are very large
* I-nodes: data structure that lists file attributes and disk addresses of data blocks
* Advantage: I-node array is much smaller than the table in memory for the pointers; I-node only needs to be in memory when the file is open
* Disadvantage: Having a fixed number of addresses is not good for larger files, so can reserve a block to have number of another block that can old more disk addresses
* Shared Files:

- Link: connection between a directory and a shared file

- File system becomes a directed acyclic graph instead of a tree

- Symbolic linking: a directory that is to link to a file will have a file of type LINK that contains the pathname of the file to be linked

* Advantage: Can be used to link files on machines anywhere by giving the network address of the machine the file resides in
* Disadvantage: Extra overhead to parse and follow the pathname of the file

- Hard linking: The directory points to the I-node that contains the file information of the file to be linked

* Disadvantage: If the directory that contains the file removes it, then the directory pointing to that file will now point to an invalid I-node
* Disk Space Management:

- Block size: small blocks = using many blocks for a file and reading will be slow because of seeking

- Keeping track of free blocks:

1) Linked list allocation of disk blocks -> each block holds as many free disk block numbers as will fit

2) Bitmap -> each bit represents a block

- Advantage: requires less space than linked list

- Disk quotas: the system administrator assigns each user a maximum number of files and blocks and OS makes sure that users do not exceed their quotas; a record of each user’s quota is kept including number of warnings, number of blocks, etc.

**Chapter 5: Input/Output**

* I/O Devices:

1. Block devices: stores info in fixed-size blocks, each w/ own address; can read/write each block independently (ex: hard disks, Blu-ray disks, and USB sticks)
2. Character device: delivers/accepts stream of characters w/o regard to block structures; not addressable and has no seek operation (ex: printers, network interfaces, etc.)

- Speed: B, GB, KB, or MB per second

* Direct Memory Access:

- OS can only use DMA if the hardware has a DMA controller

- CPU needs to address the device controllers to exchange data with them

- DMA process:

1) CPU programs DMA controller by setting its registers, also issues command to disk controller to read data from disk

2) DMA controller initiates transfer by issuing read request over bus to disk controller

3) Data transferred; write to memory

4) Disk controller sends acknowledgement signal to DMA controller

- Cycle stealing: the device controller steals a bus cycle from CPU

- Burst mode: in block mode, DMA tells the device to acquire the bus, issue transfers, then release bus; more efficient than cycle stealing because acquiring the bus takes time and multiple words can be transferred for one bus acquisition; disadvantage is that it can block CPU for a period of time if a long burst is being transferred

- Fly-by mode: DMA controller tells device controller to transfer data directly to main memory

- Most DMA controllers use physical memory for transfers

- CPU is usually faster than DMA controller

* Clocks

- Maintain time of day

- Prevent processes from arguing over the CPU

- Can take the form of a device driver

- Clock hardware:

1) One tied to 110 or 220-volt power line and cause interrupt on every voltage cycle at 50 or 60 Hz.

2) Other is built out of a crystal oscillator, counter, and holding register; a CPU interrupt occurs when counter gets down to 0

- One-shot mode: when clock starts, it copies the value of the holding register into counter and decrements counter at each pulse from crystal

- Square-wave: after getting to 0 and causing an interrupt, holding register is copied into counter and process repeats

- Clock ticks: periodic interrupts

- Advantage of programmable clock: interrupt frequency can be controlled by software

- UTC (universal coordinated time) = Jan. 1st , 1970

- counter incremented by one for every clock tick

- Clock software:

- clock driver: maintains time of day, prevent processes from running too long, account for CPU usage, handle alarm system call, provide watchdog timers for parts of the system, profiling, monitoring, and statistics gathering

- real time = time of day

- soft timers avoid interrupts

**Chapter 6: Deadlocks**

* Deadlocks: A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause
* Resources:

- Preemptable resource: can be taken away from process with no negative effects

- Nonpreemptable resource: cannot be taken away without causing a failure

* Deadlocks:

- 4 conditions must hold for a deadlock to occur:

1) Mutual exclusion condition -> each resource is either currently assigned to exactly one process or is available

2) Hold and Wait -> processes holding resources that were granted earlier can request new resources

3) No-preemption -> resources given to a process cannot be taken away from a process; must be released by the process holding them

4) Circular wait -> must be a circular list of 2 or more processes, each of which is waiting for a resource held by the next member of the chain

- 4 strategies to deal with deadlocks:

1) ignore the problem

2) detection and recovery

3) dynamic avoidance by careful resource allocation

4) prevention by structurally negating one of the four conditions

* Deadlock Detection Algorithm:

- does not prevent deadlocks, but tries to detect when they occur and try to recover from them

- Deadlock Detection with One Resource of Each Type:

* System has one resource of each type
* If no cycles exist, no deadlock
* Algorithm: for each node in the resource graph, and do depth-first search on it; if it comes to a node already visited, then there is a cycle

- Deadlock Detection with Multiple Resources of Each Type:

* Matrix-base algorithm
* Em = resource of a certain class
* Ai = available resources
* C is current allocation matrix: Cij = number of resource j in process i
* R is request matrix : Rij = number of resources j that Pi wants
* Each process will be marked when they are able to complete; any unmarked processes will be deadlocked

-Banker’s Algorithm for a Single Resource:

* Algorithm checks to see if request will lead to unsafe state (possible deadlock) ; if yes, request denied
* Table where each process has resources and each has a max

-Banker’s Algorithm for Multiple Resources:

* 2 matrices: one for current resources and one for needed resources
* Processes must state total resources needed before execution
* E = existing resources, P = possessed resources, and A = available resources
* Mark terminated processes and add resources to A matrix vector
* Deadlock Prevention:

- Mutual Exclusion Condition: avoid assigning a resource unless necessary and try to make sure that as few processes as possible may claim the resource

- Hold and Wait Condition: require all processes to request resources before starting execution; if all resources are available, process will be allocated what is needed and can run to completion, otherwise nothing will be allocated and the process just waits; another way is to require a process requesting a resource to first temporarily release resources it holds then get everything it needs at once

- No-Preemption -> some resources can be virtualized

- Circular Wait -> have a rule where a process can have only one resource (must release one to get another) ; or have processes number requested resources in order

* Starvation: a job cannot be completed due to certain policies, and it end sup being starved