LE/EECS 3221 – OPERATING SYSTEM FUNDAMENTALS

SAMPLE RELEASE

WINTER 2022

Chapter 1 – Introduction to Operating Systems

Important Terminology

- **User Mode:** A mode of a program in which only a subset of the machine instructions is available. Forbidden instructions are those that affect control of the machine, or do I/O.
- **Multiplexing:** Sharing resources by means of time or space.
    - Time ➔ When a resource is time multiplexed, different programs or users take turns using it.
    - Space ➔ Programs each get part of the resource. This raises issues of fairness, protection, and so on, and it is up to the OS to solve them.
- **Multiprogramming:** Running multiple programs by having the CPU switch between different jobs, while waiting for I/O from a previous task.
- **Simultaneous Peripheral Operation on Line (SPOOL):** A technique for reading and writing to faster storage while the job is running, and accessing the slower devices separately.
- **Time Sharing:** Multiprogramming with quicker response time due to faster job switching.
- **Simple Personal Computer:** An abstracted model with a CPU, Memory, and I/O Device. The components are connected and communicate with each other through a system bus.
- **Processor:** The brain of the computer, which executes computer programs.
- **Random Access Memory (RAM):** A type of main memory which is volatile. This means that it loses its memory content when it loses power.
- **Memory Management Unit (MMU):** Translates memory addresses between disk and main memory. It also has the responsibility of making sure that physical addresses used by the programs will not overlap.
- **I/O Device**
    - **Controller:** A physical chip which runs the device, accepting commands from the OS, which it will then execute accordingly.
    - **Device Driver:** A software that runs in kernel mode, interacting with the controller, giving it commands and accepting responses.
- **Busy Waiting / Poling / Programmed I/O:** When a driver issues a command to the controller, it will sit in a tight loop, polling the device to see if it is done.
- **I/O by Interrupts:** The driver starts the device and asks it to give an interrupt to the CPU when finished.
- **Direct Memory Access Controller:** A chip that can control the flow of bits between memory and some controller without constant CPU intervention.
- **I/O By Direct Memory Access (DMA) Controller:** Makes use of a DMA chip. The CPU sets the DMA chip, telling it how many bytes to transfer, the device and memory address involved, and the direction. When the DMA chip is done, it causes an interrupt to the CPU.
- **Basic Input Output System (BIOS)**, a $PROM \in System$ (In Motherboard). Performs the hardware and software housekeeping.
- **Master Boot Record:** Is the first sector (512-byte block) on the boot device. It Holds information on the organization of logical partitions and passes control to the first block of the active partition
- **Active Partition:** The partition from which the computer starts up.
- **Secondary Boot Loader:** Loads and starts the OS. The OS checks the bios for the configuration.

- **Process:** A program in execution.
- **Address Space:** A list of memory locations from 0 to some maximum, which the process can read and write. The address space is decoupled from the machines physical memory and may be either larger or smaller than the physical memory.
- **Process Table:** A table represented by an array of structs, containing all the information about each process, other than the contents of its own address space. The information has to deal with process management, memory management, and file management
- **Process Tree:** A hierarchical structure representing the relationship of many processes which can create children.
- **Fork()**: A system call used to create children.
- **Uni-Programmed:** Occurs if the parent process is always blocked until the child process finishes.
- **Multiprogrammed:** Occurs when the parent is not blocked, and can fork multiple child processes. This process can lead to a **deadlock**
- **Process Signaling:** A mechanism for processes to communicate with one another, through sending signals to each other via the **kill()** system call.
- **Directory:** As a way of grouping files together.
- **Special Files:** Provided in order to make I/O devices look like files
- **Block Special Files:** Used to model devices that consist of a **collection of randomly addressable blocks**, such as disks.
- **Character Special Files:** Used to model printers, modems, and other devices **that accept or output a character stream**. Their location resides in $/dev$.
- **Pipe:** A pipe is a sort of pseudofile that can be used to connect two processes (communication). The processes communicate by writing into / reading from the pseudofile / pipe.
- **Memory Structure (Processes):** Between the stack and data segments is a gap of unused address space. The stack grows into this gap automatically, as needed Expansion of the data segment is done explicitly.
- **System Call:** System calls are the interface between user programs (user mode) and the OS. A system call is like a special procedure call, one that can enter the kernel.
- **System Call Dispatcher (Envelope):** Forwards the system calls to the appropriate OS components
- **TRAP Instruction:** Switches a program from user mode to kernel mode, and start execution at a fixed address within the kernel.
- **Monolithic System:** Entire OS runs as a **single program** in kernel mode.
- **Layered Systems:** An OS organized into a hierarchy of levels, each one constructed by the one below it.
- **Microkernel:** Splitting the OS into small, well-defined modules, only one of which – the microkernel – runs in kernel mode and the rest run as relatively powerless ordinary user processes.
- **Client-Server Model:** Processes are seen as servers, which provide some service, or clients, which uses / requests a service
- **Virtual Machines:** An operating system that runs on virtualized resources. In a Type 1 hypervisor, there is no underlying support, and it must perform all the functions by itself. In a Type 2 hypervisor, it makes use of a host-OS and its file system to create processes, store files, and so on.

Chapter 2 – Processes and Threads

Important Terminology

- **Multiprogramming System:** In such a given system, the CPU switches from process to process quickly, running each for milliseconds
- **Pseudo-Parallelism:** At any one instant, the CPU only runs one process, however, in the course of one second, it may work on several of them giving the illusion of parallelism
- **Foreground Process:** Interacts with users and performs work for them.
- **Daemon:** Not associated with particular users, but instead have some specific function
- **Scheduler:** The lowest level of the operating system is the scheduler, with a variety of processes on top of it. All the interrupt handling and details of actually starting and stopping processes are hidden away in the scheduler. The rest of the operating system is nicely structured in process form.
- **Interrupt Vector:** A fixed location near the bottom of the memory, containing the address of the interrupt service procedure.
- **CPU Bound:** Typically have long CPU bursts and thus infrequent I/O waits.
- **IO Bound:** Typically have short CPU bursts and thus frequent I/O waits
- **Non-Pre-emptive Scheduling (Batch, Real-Time):** Picks a process to run and lets it run until it blocks (either on I/O or waiting for another process) or voluntarily releases the CPU. After clock-interrupt processing has been finished, the process that was running before the interrupt is resumed, unless a higher-priority process was waiting for a now-satisfied timeout
- **Pre-emptive Scheduling (Interactive):** Picks a process and lets it run for a maximum of some fixed time. If it is still running at the end of the time interval, it is suspended and the scheduler picks another process to run (if one is available).
- **Throughput:** The number of jobs per hour that the system completes.
- **Turnaround Time (Average Waiting Time):** The statistical average time from the moment that a batch job is submitted until the moment that it is completed. It measures how long the average user has to wait for the output.
- **Response Time:** The time between issuing a command and getting a result.
- **Priority Aging:** When a job is waiting, we improve its priority*;* hence it will eventually have the best priority and will then run.
- **Starvation:** Starvation means that after a certain time, some process never runs, because it never has the highest priority. The formal way to say that is that a system is free of starvation if no job can remain in the ready state forever.
- **Aging:** The technique of estimating the next value in a series by taking the average of the current measured value and the previous estimate.
- **Thread:** Considered a, "mini process." A thread can be thought of as a process with a shared address space and multiple threads of control.
- **Semaphore:** A high-level abstraction to **control access to shared resources**. It uses **atomic operations**
- **Binary Semaphores (Mutexes):** An abstraction for the **Test and Set Lock. It is used to maintain mutual exclusion**
- **Counting Semaphores:** Used to facilitate more than one process sharing a resource. **That is, it allows a number of processes in the critical section.**

- **Data Race:** Situations where two or more processes are reading or writing some shared data, and the final result depends on who runs precisely when are called race conditions. (i.e., Printer Spooling)
- **Critical Region: That part of the program where the shared memory is accessed is called the critical region or critical section**
- **Spin Lock:** A lock that uses busy waiting is called a **spin lock** as a result.
- **Test and Set Lock (TSL Instruction):** Reads the contents of the memory word $LOCK$ into register RX and then store a non-zero value at the memory address $LOCK$. The operations of reading the word and storing into it are guaranteed to be indivisible – no other processor can access the memory word until the instruction is finished.
- **Bounded Buffer Problem:** The producer-consumer problem has **two classes of processes** which share a common fix-sized buffer. **Producers** put information in the Buffer. When the buffer is full, the producer waits for a non-full buffer. **Consumers** take information out of the Buffer When the buffer is empty, the consumer waits for a non-empty buffer. The problem can be generalized to M producers and N consumers
- **The Dining Philosophers Problem:** Five philosophers are seated around a circular table. Each philosopher has a plate of spaghetti. ***The spaghetti requires two forks to eat it.*** Between each pair of plates is one fork. The life of a philosopher consists of alternating periods of eating and thinking. **When a philosopher gets sufficiently hungry, she tries to acquire her left and right forks, one at a time in either order.** If successful in acquiring two forks, she eats for a while, then puts down the forks, and continues to think. The key question is: Can you write a program for each philosopher that does what it is supposed to do and never gets stuck?

Important Formulas

- **Probabilistic Viewpoint**
  - Suppose that a process spends a fraction p of its **time waiting for I/O to complete**.
  - With ***n* processes in memory at once**, the probability that all n processes are waiting for I/O (in which case the CPU will be idle) is p^n
    - $CPU\ Utilization = 1 - p^n$
- **Shortest Currently Runnable Process (Weighted Sum):** $aT_0 + (1-a)T_1$
  - Suppose that the estimated time per command for some process is T_0
  - Suppose its next run is measured to be T_1
  - Through the choice of a, we can decide to have the estimation process forget old runs quickly, or remember them for a long time
  - Easy to implement when $a = \frac{1}{2}$.
    - All that is needed is to add the new value to the current estimate and divide the sum by 2
- **Penalty Ratio:** $r = \frac{T}{t}$
  - T = How long this process has been in the system
  - t = Running time of the process to date
  - **Note:** We must worry about a process that just enters the system, since, for such processes, $t = 0$ and hence the ratio is undefined. So, we define $t = \max(1, t)$ to keep the ratio defined at all times.
- Guaranteed Scheduling: **If n users are logged in while you are working, you will receive about $\frac{1}{n}$ of the CPU power.**
  - To make good on this promise, the system must keep track of how much CPU each process has had since its creation. It then computes the amount of CPU each one is entitled to, namely the time since creation divided by n. ➔ $\left[\frac{T}{n}\right]$
  - Since the amount of CPU time each process has actually had is known, it is fairly straightforward to compute the **ratio of actual CPU time consumed to CPU time entitled**: $R = \left(\frac{t}{\left(\frac{T}{n}\right)}\right)$
- $\sum Sequential\ Process\ Time = \frac{CPU\ Time}{IO\ Wait\ (In\ Decimal)}$
- $\sum Parallel\ Process\ Time = \frac{CPU\ Time}{\left(\frac{1-p^n}{\#\ jobs}\right)}$
- $Mean\ Turnaround\ Time = \frac{a+(a+b)+(a+b+c)+\cdots+(a+b+c+d)}{\#(Jobs)}$
- $CPU\ Utilization = \frac{\#(Cycles\ CPU\ was\ Doing\ Work)}{\#(Total\ Cycles)}$
- $Process\ Turnaround\ Time = Cycle_{Finish} - Cycle_{Arrive} + 1$

Chapter 6 – Deadlocks

Important Terminology

- **Deadlock:** 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.
- **Resource:** A resource is anything that must be acquired, used, and released over the course of time. A resource can be a hardware device or a piece of information.
- **Preemptable Resource:** One that can be taken away from the process owning it with no ill effects. Memory is an example of a preemptable resource.
- **Nonpreemptable Resource:** A non-preemptable resource is one that cannot be taken away from its owner without potentially causing failure.
- **Mutual Exclusion Condition:** Each resource is either currently assigned to exactly one process or is available.
- **Hold-and-Wait Condition:** Process currently holding resources that were granted earlier can request new resources.
- **No-Preemption Condition:** Resources previously granted cannot be forcibly taken away from a process. They must be explicitly released by the process holding them.
- **Circular Wait Condition:** There must be a circular list of two or more processes, each of which is waiting for a resource held by the next member of the chain.
- **Ostrich Algorithm:** Stick your head in the sand and pretend there is no problem.
- **Deadlock Detection and Recovery:** The system does not attempt to prevent deadlocks from occurring. It lets them occur, tries to detect when this happens, and then takes some action to recover after the fact.
- **Recovery Through Preemption:** To temporarily take a resource away from its current owner and give it to another processes. Manual intervention may be required. Do note that the ability to take a resource away from a process, have another process use it, and then give it back without the process noticing is highly dependent on the nature of the resource
- **Checkpointing:** Checkpointing a process means that its state is written to a file so that it can be restarted later. It contains not only the memory image, but also the resource state (which resources are currently assigned to the process.)
- **Recovery Through Rollback:** A process that owns a needed resource is rolled back to a point in time before it acquired that resource by starting at one of its earlier checkpoints.
- **Recovery Through Killing Processes:** The crudest but simplest way to break a deadlock is to kill one or more processes. With a little luck, the other processes will be able to resume operating. Alternatively, a process not in the cycle can be chosen as the victim in order to release its resources.
- **Resource Trajectories:** The main algorithms for deadlock avoidance are based on the concept of safe states
- **Claim of the Process:** That is, each process, before making any resource requests, must tell the resource manager the maximum number of units of each resource the process can possibly need
- **Safe State:** A state is said to be safe is there is some scheduling order in which every process can run to completion even if all of them suddenly request their maximum number of resources immediately.

- **Bankers Algorithm:** The banker's algorithm considers each request as it occurs, seeing whether granting it leads to a safe state. If it does, the request is granted; otherwise, it is postponed until later. To see if a state is safe, the banker checks to see if he has enough resources to satisfy some customer. If all loans can eventually be repaid, the state is safe and the initial request can be granted.
- **Livelock:** Although the two processes that are being polite are never deadlocked per se (they are not blocked), there is still no progress that is possible. Thus, we have a livelock.

For access to the remaining notes, please contact me over LinkedIn or GitHub