Chapter 1

- OS's main functions: resource allocator and control program
- Bootstrap program
    - Starts the kernel of the computer
- Computer system organization/operation
    - Uses the queue system
        - Grab resources and distributes it.
- Device controller, Device status table, Device drivers
    - Device controller – controller interacts with the memory indirectly
    - Device status table – table that contains information about each device such as type, address and state.
    - Device drivers – allow the devices to be interact with the computer in a special way or to get access to more features.
- Interrupt, Interrupt vector, interrupt handling
    - Interrupt – tells CPU that it needs attention to get more information or if something went wrong
    - Interrupt vector – interrupts are handled here so it is put in a queue.
    - Interrupt handling – goes through a interrupt service routine. For a hardware interrupt, the CPU is interrupted so it could send information to the kernel. For software interrupt, the CPU does not send any information to the kernel.
- Difference(s) between interrupt and polling
    - Interrupt – signal that tells CPU that it needs attention at that point in time due to an error, or lack of resources.
    - Polling – protocol that checks status of the device to see if it will need CPU's attention.
- Difference(s) between system and application programs
    - System programs allow communication between the user and the hardware
    Ex: Compilers, Assemblers
- Trap or exception
    - Trap and exception are known as the same thing, they are a software -generated interrupt that is caused by an error or a user request.
- System call
    - Request OS to allow the user to wait which is requested by the user
- Caching
    - Putting information into a cache so it can be used later on but faster due to the CPU being able to interact with it directly
- Direct Access Memory
    - High speed communication between I/O Devices and the CPU.
- Multiprocessors/parallel systems - advantages, and differences between Asymmetric and Symmetric Multiprocessing
    - Parallel processing is done quickly.
    - Asymmetric processing is where there is one "master" and then the rest are slaves. The "master" organizes the tasks that the "slaves" will carry out.
    - Symmetric processing produces all the tasks out at the same time by having separate cache and registers per CPU but still share the same memory.
- Dual-Core Design
    - An example of multiprocessing system -
- Clustered systems – why they are they considered high-availability systems? Asymmetric & symmetric clustering
- Two modes of operation (User & Kernel modes)? Why and how Dual-modes are used?
    - Dual modes are used to prevent the user from directly accessing the hardware and causing errors.
- Why timer is used during a process's execution
    - The timer is used to prevent a infinite loop and hogging of resources.
- Why is Cache coherency in multiprocessors environments important?
    - Cache coherency allows all the CPUs in a multiprocessor environment so that all cache is updated at the same time.
- Computing environments

Ch. 2 (Operating-System Structures) Study Guide

• What are operating system services?

```
- Services that allow the user to interact with the computer in an easy fashion due to the system services that take care of the functions that are wished to be carried out
```

• Why and how system calls are used & the role of system call interface?

```
- System calls are used to interact with the services that the OS provides so certain actions can be carried out i.e. Communication between applications/devices. The role of the system call interface is that each system call has a specific number or index specified to them which contains a specific service. Whenever a system call takes place, it will look at the index and then point to the correct service.
```

• Methods of passing parameters to the operating system during system calls

```
- The three different methods are : pass by register, pass by register via table/block due to parameters being too large or lack of registers, and stack (pop/push)
```

• Communication models

```
- The two communication models are shared memory and message passing.
    - Shared Memory - communication that is done on the same memory but requires special privileges and proper setup as processes have to make sure that they cannot overwrite one another's memory space.
```

• Operating system design goals (user goals and system goals)

```
- User goal is different from system goal as users want the system to be safe, reliable, and fast.
- System goal on the other hand is for developers or contributers who want the system to be easy to maintain, upgradable, etc.
```

• Policy & Mechanism

```
- Policy is a matter of what will be done
- Mechanism is a matter of how said thing will be done
```

• Microkernels – advantages and disadvantages

```
- Easy to extend for more features
- Easier to port across hardware
- Reliable
- Safer
- Much system-function overhead as there is overhead with the user.
```

• OS structures: DOS

```
- Single layered structure where only one process can be loaded at a time. If there is an error at any point in time, it will be loaded after the program had terminated completely.
```

Layered

```
- Hardware is considered layer 0, user applications are of layer 'N'. The
distinction of this is that any higher layer can use the layers below them.
- The difficult thing about layered OS systems is defining these layers as that will
take careful thinking
```

Modular kernel (advantages)

```
- Object oriented
- Can speak to each other across same interface
- Each core is separated
- Can be loaded when necessary, does not need to be idle at all times.
```

• Log file

```
- File that contains information about the computer's processes, errors, etc.
```

• Core dump

```
- When an application fails a core dump is created
```

• Crash /crash dump

```
- When kernel fails
```

• GRUB

```
- Some open source bootlader program
```

• SYSGEN

```
- Creating specific configurations per computer, used for operating system
generation
• Bootstrap loader
```

Chapter 3 Study Guide: • Difference between a program and a process

```
- Program is a passive entity and process is an active entity. Program is considered
passive, because it is just existing on the disk until it is executed. After being
executed, it becomes an active entity since it is executing the instructions that
was implemented to it.
• Different parts of a process (i.e., text, data, stack, heap, program counter).
- Text - place where code is implemented
- Data - Global variables
- Stack - Contains temporary data such as function parameters, return addresses,
local variables
- Program Counter - also known as the instruction pointer. It contains the address
location of the program whenever it is called/modified.
- Heap - memory dynamically allocated at run time
• Process states
- New (Program admitted) -> Ready (scheduler dispatch) -> Running
                - Running (I/O or event wait)-> Waiting (I/O Completion) -> Ready
                - Running (Interrupt) -> Ready
                - Running (Exit) -> Terminated
• Process scheduling/scheduler
- What is a process scheduler?
    - selects what should be executed next on CPU
- Different types of queues (Job Queue, Ready Queue, Device Queue)
```

    * Job - set of all processes in the system
    * Ready - set of all processes that are in main memory that are ready and waiting to execute
    * Device - processes waiting for I/O Device
• Scheduler - Short-term, Long-term and Medium-term schedulers
- Short term scheduler - selects process that should be executed next and allocates CPU, maybe the only scheduler on the system at some point in time.
    - Requirement - must be checking constantly (Interaction must be fast)
- Long-term scheduler - selects the processes to be brought to the ready queue.
    - Does not check as frequently as the short term scheduler (seconds/minutes)
    - Controls degree of Multiprogramming
- Medium Scheduler - uses swapping, is added when degree of multiple programming needs to decrease
• Swapping
- Swapping takes place between the disk and the memory. It goes into the disk to bring it back to memory to continue execution of the current program.
• Context Switch - method of saving and loading current state/saved state for the process
• Describe Program Control Block (PBC) it’s purpose.
- Used to save and load the process that didn't finish, and it is done with the help of the PCB. The program counter in the PCB updates register information, etc.
• When a process creates a new child process, what are the possibilities in terms of execution, address space of the new process, and termination for parent and children processes (Options)
+ Execution possibilities
    - I/O request
    - time slice expired
    - fork a child
    - Possible to fork a child and a tree is formed
    - wait for an interrupt
    All roads end up in ready queue
+ Address space
    - Child is the duplicate of parent
    - Child has a program loaded into it
    - new memory space filled in for the program in a unix environment
+ Termination options:
    - Execute last statement and then ask OS to delete itself via exit()
        * wait()
        * exit() takes places and resources that were allocated to this process are deallocated.
    - abort()
        * when child takes too much resources (due to threshold or lack of existing resources)
        * Child is no longer needed to execute task
        * Parent is terminating and OS does not allow child to still run without a parent
        • Cascading termination - When parent terminates, the processes that it created, and other other nodes that were created by children are deleted
- Zombie - child is waiting for parent
- Orphan - parent terminated without invoking wait()
• Interprocess communication (IPC) – Independent and cooperating processes
- Two models of IPC: Shared memory/message passing
    - Independent - Processes cannot affect/ are not affected by existing processes
    - Dependent - processes that are affected by existing processes, most likely due to some link, can also affect other existing processes as well
• Shared Memory and Message Passing models of IPC

- Shared memory - faster due to kernel not being involved. Larger risk as processes can overlap one another. Due to this, synchronization of processes are highly ensured.
- Message passing - is slower in comparison to shared memory, but less risky as kernel deal with the communication of the processes.
• Direct and Indirect communications
- Direct - Link between two processes
- Indirect - processes send information to the mailbox to receive and it is distributed from there as it doesn't matter who the receiver is.
• Synchronization (blocking/synchronous & non-blocking/asynchronous)
- Message passing can be blocking/nonblocking
    - Blocking
        * blocking send - user is blocked from sending message until its received by the end user
        * blocking receive - user is blocked from receiving message until the sender sends information
    - Non-blocking
        * Nonblocking send - sender can receive message and continue
        * Nonblocking receive - able to get null/valid messages
• Client-server communication
- Communication ports establish communication between devices, in this case it is the client and the server. Client and server use the port to send messages/listen for replies
• Sockets
- End point of communication
- IP and port used to differentiate network services on a host.
• Remote procedure calls (RPC)
- abstraction for procedure calls for processes and networking systems
• Stubs
- client side proxy for actual procedure on the server
• Marshalling
- packaging of parameters on the client side
• Rendezvous
- Term used when process if receiving and sending is blocking
• Pipes (Ordinary & Named) for process communication
- Ordinary - cannot be accessed from outside the process that created it. Typically, a parent process creates a pipe and uses it to communicate with a child process that it created.
    - Specific to parent and child
    - Producer-consumer relationship
    - Unidirectional
- Named pipes - can be accessed without a parent-child relationship.
- More powerful than ordinary pipes
- No parent-child relationship
- Bi-directional
- Several proccesses can use named pipe for communication in comparison to the ordinary pipes which only allows (parent-child)
- Like with communication of message passing (there is direct and indirect)

Chapter 4 (Threads) – Study Guide • Thread definition

- Formal definition - A unit of CPU utilization
- Informal definition - a set of instructions that are executed to carry out a task.
• Benefits of multithreaded programming
- Responsiveness
    - Tasks can be done more efficiently as sets of instructions can be executed at once instead of one at a time
- Resource Sharing

    - Threads share resources of the process
- Economy - cheaper in comparison to process creation
- Scalability - takes advantage of multiprocess systems
• Multicore/multiprocessors systems
- Multicore also known as multiprocessors allow concurrent and parallel processing
• Concurrency v. parallelism
- Concurrency - run a block of instructions one instruction at a time
- Parallelism - run two or more instructions simultaneously
• data parallelism & task parallelism
- data parallelism - distributes subsets of same data across multiple cores, same operation on each
    - Laymen explanation - given a set of data that is the same such as block of I/O, they will be processed the same way on different cores
- task parallelism - distributes threads across cores with each thread performing a unique operation
    - Laymen explanation - loading a website
• User and Kernel threads and their management
- User threads - management of threads done by a user library
- Kernel threads - management of threads supported by the kernel
• Multithreading models (many-to-one, one-to-one, many-to-many, 2-level)-strengths and weaknesses
+ Many to one
    - Not a popular model due to many user threads being mapped to a single kernel thread. All threads come from one kernel, for multithreading to be executed successfully, there needs to be multiple kernels
+ One to one
    - User thread mapped to kernel thread
        - Creation of a user thread also creates a kernel thread
    - More concurrency than many to one
    - # of threads per process are limited due to overhead occuring
+ Many to many
    - many user threads are mapped to many kernel threads
    - OS creates number of kernel threads
+ 2 level
    - Allow user thread to be bound to a kernel thread and allow many user threads to be mapped to kernel threads at the same time
• Thread libraries
- provides API for creating/managing threads
• Pthreads
- Provided at user level or kernel level
- POST IEEE 1003.1c - thread creation and synchronization
• Implicit threading
- Growing in popularity because of numbers of threads are increasing nowadays
- Creation/Management of threads done by compilers and runtime libraries rather than on the programmer side
    • Thread pool, Open MP, Grand Central Dispatch
    - Thread Pool - Threads are created and put into a pool where the will wait until they are called to work. Allows faster response to a process who needs threads to execute tasks since these threads are just waiting. Number of threads in applications are bound to the size of the pool. Threads are then recycled and put back into the pool rather than be deleted
    - Open MP - provides parallel programming in shared memory environments
        - parallel regions are identified by blocks of code that run in parallel
    - Grand Central Dispatch - allows identification of parallel sections
        - Common in Apple technology
        - manages most of the details in threading
        - blocks are placed in dispatch queue

```
            - Assigned to available thread in thread pool when removed from queue
        - Dispatch queues: serial and concurrent
            - Serial - block removed in FIFO one by one
            - Concurrent - removed in FIFO, but multiple blocks can be removed
• Signals, signal handling
- Notification that a paritcular event occurred
- Signal handling - process signals that are generated by event/ delievered by a
process/ handled by one of two signal handlers (default and user-defined)
• Thread cancellation (Asynchronous & Deferred cancellation)
- Terminating a thread before it has finished executing its task
- Target - thread that is being targeted to terminate before it is done executing
its tasks
- Asynchronous - termiantes target head immediately
- Deferred - allows thread to be periodically checked if it should be cancelled
• Thread-Local Storage
- allows thread to have its own copy of data
- useful when user has no control of thread creation process
• Scheduler activations
-Both M:M(many to many) and two-level models require communication to maintain the
```
appropriate number of kernel threads allocated to the applicationn -Typically uses an intermidiate data structure between user and kernel threads- lightweight process(LWP) -> acts as virtual processor -Schedular activations provide upcalls- a communciation mechanism from the kernel to the upcall handler in the thread library -This communication allows an application to maintain the corrent number of kernel threads

Ch. 6 (CPU Scheduling) Study Guide

• The purpose of CPU scheduling

The purpose of CPU scheduling is to be able to maximize the amount of work done on CPU.

Utilize it to its maximum efficiency

• Short-term (CPU) Scheduler

```
- Short Term CPU Scheduler selects a process from the ready queue and allocates CPU
to it and runs the process.
    - Scheduling can occur when a process switches from running to waiting or
terminates, these processes are known as non-preemptive.
    - Scehduling can also occur when a process switches from running to ready and
waiting to ready, these processes are known as preemptive
```

• Dispatcher - Dispatch latency

```
- Dispatcher is the way of grabbing the CPU's attention and allows the CPU to be in
control of the process that was selected via CPU scheduler.
- Dispatch latency is the description of the time taking for the dispatcher to allow
the CPU to  stop a process and start another.
```

• Throughput /Turnaround/Response/Waiting time

```
- These are parameters that need to be tuned properly.
- Throughout must be maximized as it is the amount of processes that complete their
execution per min. The larger the amount, the better it is for the user.
- Turnaround time must be minimized as it is the amount of time taken to execute a
process
```

```
- Waiting time must be minimized as it is the amount of time a process has been
taking while in the ready queue.
- Response time must be minimized as it is the amount of time taken for the first
response to be produced when a request was submitted.
```

• FCFS (nonpreemptive)

```
- Also known as first come first serve.
- It schedules basing on what processes arrived first.
- Not used in many OS due to its flaws
```

• Convoy effect

```
- Describes when there are processes with shorter burst time that are behind a
process with a large burst time/execution time
```

• SJF (nonpreemptive) and its difficulty

```
- Also known as shortest job first
- Associates the burst time of each process and allows the shortest burst time job
to be executed first. It is ideal as it can give the minimum average waiting time
- Difficulty - cannot know the burst time of a future process so a prediction is
used to determine it
```

• SJF (preemptive) Shortest-remaining-time-first

```
- Constantly preempts by checking the arrival time of the incoming processes and
their burst time. Even if a process with a short burst time arrives first, it could
be subject to change based off the incoming processes.
```

• Priority Algorithm

```
- To assist in choosing what processes will be processed in what order, priority
algorithm is used. In the PCB, there is an additional number known as the priority
integer which is associated with the processes. The integer states on whether or not
if it is of "high", "mid", or "low" priority.
```

• Starvation/Aging

```
- Starvation - lower prioritized processes will fail to execute and may never be
able to.
- Aging - as time continues, the priority integer will increase and the priority
will increase as well.
```

• Round Robin

```
- It is a cpu scheduling algorithm at which there is a time quantum (a time of how
long a process will run on CPU) which is chosen by the user and any of the processes
that are put into the ready queue are executed for that amount of time.
- If the process is not finished completing within that specified time quantum, it
is put to the end of the queue and will execute later on.
```

• Size of time quantum

```
- Time quantum can be between 10ms -100ms
- If it is too long, it becomes a First come first serve scheduling algorithm
```

```
- If it is too short, there will be much context switching occuring, which will create much overhead.
```

• Multilevel Queue

```
- Separate queues for interactive and batch (background processes)
```

• Foreground & background queues

```
- These queues can have separate CPU scheduling algorithms. Interactive has a Round robin CPU scheduling algorithm, while batch has FCFS.
- The idea is to dedicate most time to interactive processes and then to the background processes. This has a possibility of starvation
- 80% time dedicated to foreground (interactive), 20% time dedicated to background
```

• Multilevel feedback queue

```
- Process can move between various queues, allows process to run on specified time quantum within any of the queues.
- Example of multilevel feedback queue
- v represents exit, and --> represents destination
- P1 -> Time quantum (10 ms) v or -> Time quantum (12 ms) v or -> FCFS v or ->
```

• Thread Scheduling (PCS, SCS)

```
- Process contention scope - user threads run on LWP and compete against other user threads to run on CPU
- System Contention Scope - kernel threads compete to run on next available CPU and competes against all threads in system
```

• Multiprocessor scheduling

• Homogeneous processors

```
- Scheduling within its own multiprocessor
```

• Asymmetric multiprocessing

```
- one processor accesses system data structrue and alleviate need for data sharing
```

• Symmetric multiprocessing (SMP)

```
- each processor is self-scheduling, all processes in a common ready queue, or each has its own private queue
```

• Processor Affinity – Hard/Soft affinity

```
- Hard affinity - process stay in the same processor after running the first time
- Soft affinity - if the process is not important, process doesn't need to return to the same CPU.
```

• NUMA (Non Uniform Memory Access)

```
-  is a computer memory design used in multiprocessing, where the memory access time depends on the memory location relative to the processor.
```

• Load balancing (Pull/Push migration)

- push migration - periodic task checks load on each processor, and if found pushes task from overloaded CPU to other CPUs
- pull migration - idle processors pulls waiting task from busy processor