**CS372 Study Guide (I reserve the right to interpret if a question in is covered by this guide. Still, only about 90% of questions are covered by this guide**. )

# Chapter 1

* Know what is an operating system.

A program that manages a computer’s hardware. Acts as intermediary between user and computer hardware.

* Be familiar with the figure on the slide 1.6

Symmetric Multiprocessing Arch

* Know the steps to start up a computer.

Bootstrap program is loaded at power up or reboot. Loaded from ROM or EPROM (Firmware). Initializes all aspects of system. Loads operating system kernel and starts execution.

* Understand the concept of interrupt: What is an interrupt, What does it do, Interrupt handling.

An interrupt is an event signaled by hardware or software to the CPU (System Bus). CPU stops what its doing and immediately starts executing at a fixed location usually start address for service routine is located. Once complete resumes interrupted computation.

* Know storage hierarchy: Caching, Device driver, DMA,…

**Registers:** memory for instructions (closest to CPU).

**Cache:** Memory used between registers and main memory (Register and main memory)

**Main Memory:** Data storage for quick access by CPU.

**Solid State Memory**: Faster than Magnetic disks. Slower than DRAM but needs no power. Nonvolatile.

**Magnetic Disk**: Hardrives.

**Optical:** Cd & DVD

**DMA:** Blocks of Data from buffer storage go directly to main memory without CPU intervention. Interrupt is generated per block opposed to per byte.

* Know computer system architecture: Asymmetric Multiprocessing, Symmetric Multiprocessing, Multicore, Cluster

**Asymmetric Multiprocessing:** Has each processor assigned to specific task.

**Symmetric Multiprocessing:** Has each processor perform all tasks.

**Multicore:** Multiple cores in one chip. Is a type of multiprocessor.

**Cluster:** Composed of two or more individual systems, nodes, joined together.

# Chapter 2

* Be able to describe the services an operating system provides: User interface, Program execution, I/O operations, File-system manipulation, Communications, Error detection, Resource allocation, Accounting, Protection and security. Figure on the slide 2.7

**User Interface**: UI

**Program Execution**: System must be able to load program into memory and run, end execution, normally or abnormally.

**I/O Operations:** Running a program which involves a file or i/o device.

**File Manipulation:** read& write files. Create, delete, search directories. List file info and permissions.

**Communications:** Processes may exchange info on same or over networks. Can be shared memory or memory passing.

**Error Detection**: OS take appropriate action for correct and consistent computing.

**Resource Allocation**: Resource sharing many users, multiple jobs running concurrently. Resource types CPU cycles, main memory, I/O, etc.

**Accounting:** Keep track of which users use how much of resources.

**Protection & Security**: Concurrent process don’t interfere with another. Protection involves that all access to system resources is controlled. Security involves requiring outsiders user auth extends to defending external I/O devices from invalid access attempts.

* Know system calls: What are system calls, How is an system call implemented (Figure on the slide 2.17), Parameter passing, Three common API to system call.

System Calls are an interface to services made available by operating system.

System Call Implementation: Usually a number associated with system call. Moves from user mode to kernel mode. Parameter passed to registers , sometimes stored, or pushed, popped, on stack.

Three common API Win32, POSIX, Java.

* Know the difference between “policy” and “mechanism”.

**Policy:** What will be done

**Mechanism:** How it will be done.

* Know the various operating system structures: Monolithic, Layered, Microkernel, Loadable.

**Monolithic**:

**Layered**: OS broken into layers. Layer 0 is hardware layer. Highest layer is Layer N is user interface.

**Microkernel**: Removes all non-essential components from kernel and implements them as system and user level programs.

**Loadable**: kernel has set of core components and links in additional modules , either at boot or run time.

* Know some common operating systems’ structures: Linux, Windows, Mac OS X.

**Mac**: use hybrid structure but is layered.

**Linux**: Monolithic but also modular

**Windows**: Monolithic but also microkernel in behavior.

# Chapter 3

* Know process concepts: What is a process, Process components in memory.

**Process**: Is a program in execution.

* Understand the transmission of process states. Figure on the slide 3.8.
* Know PCB and at least 5 components of PCB.

**PCD:** Process control block also task control block.

**5 Components**: Process State, Program Counter, CPU registers, CPU Scheduling Information, Memory Management Information, Accounting Information, and I/O Status Information.

* Understand the queues in process scheduling, Figure on the slide 3.15.

Processes are put into a Job Queue along withal processes in system. Process in main memory waiting for execution are kept in a list know as a ready queue.

* Know the differences of various process schedulers: short-term, long-term, medium-term.

**Short-term:** Selects which process to execute and allocates CPU.

**Long-term:** Selects which processes should be brought to ready queue.

**Medium-term:** added if degree of multiple programming needs to decrease.

* Understand context switch.

**Context Switch:** The process of state save of a current process and the state restore of another process**.**

* Know the basic APIs: fork(), exec(), exit(), abort(), wait(), getPid(), and their corresponding windows APIs.

**Fork():** creates new process. WindowsCreateProcess**()**

**Exit():** exit process. Windows ExitProcess()

**Exec():** Overlays the processes space with new program. No windows

**Abort():** Like exit() but cannot ot close files that are open, delete temporary files and may not flush stream buffer.

**Wait():** blocks the calling process until one of its child processes exits or a signal is received**.**

**getPid**(): returns the process ID of the parent of the calling process.

* Understand interprocess communication: shared memory, message passing

Interprocess communication is a mechanism for cooperating process to exchange data and information. Consists of two models Shared Memory and Message Passing. Shared Memory a region of memory is set aside for cooperating processes. Message Passing takes place between cooperating processes. Less conflict. Blocking or Non Blocking

* Communications in Client-Server Systems.

Three strategies for client-server communication: socket-pair of processes communicate over a network socket, Remote Procedure Calls- sends a request message to a known remote *server* to execute a specified procedure with supplied parameters. The remote server sends a response to the client, and the application continues its process, and Pipes-

A unidirectional data channel. Data written to the write end of the pipe is buffered by the operating system until it is read from the read end of the pipe.

# Chapter 4

* Understand thread concepts: What is a thread, Why is it light weight.

A thread is a basic CPU utilization, it contains thread ID, program counter, register set, and a stack.

A thread is light weight since it shares same code, processes data and other resources with other threads that belong to same process

* Know the differences between parallelism and concurrency.

**Parallelism:** parallelization of computer code across **multiple** processors in parallel computing environments.

**Concurrency:** run several programs or several parts of a program in parallel

* Know the three multithreading models: Many-to-One, One-to-One, Many-to-Many (two level). Know at least one example system in each model.

**Many to One:** Has one kernel thread

**One-to-One:** maps each user thread to a kernel thread

**Many-to-Many:** multiplexes many user-level threads to smaller or equal kernel threads

* Know the three primary thread libraries and the basic operations: pthread\_create(), pthread\_join(),pthread\_exit(), and their corresponding operations in windows library and java library if there are.

**Three thread libraries: POSIX** Pthreads, Windows and Java.

**Pthread\_create():** creates separate thread

**Pthread\_join():** suspend execution of the calling thread until the target thread terminates

**Pthread\_exit():** terminates the calling thread, making its exit status available to any waiting threads.

* Understand threading issues: semantics of fork(), signal handling, thread cancellation, thread local storage, scheduler activations.

**Signal Handling:** Signal is generated, signal is delivered, signal is the handled one of two ways either by default handler or user-defined handler.

**Thread cancellation:** termination of thread before completion.

**Thread Local Storage:** copy of data if needed by thread.

**Scheduler Activations:** scheme of communication between user thread library and kernel.

* Windows threads and Linux threads

**Windows:** uses one-to-one mapping, has thread id, register set, user stack, private storage each process may contain one or more thresds.

**Linux:** Can use clone system call. Does not distinguish between processes and threads. Also referred to as **Task** in Linux.

# Chapter 5

* Know basic concepts: Process execution cycle, short-term scheduler, preemptive scheduling, non-preemtive scheduling, dispatch latency.

**Short term Scheduler:** selects processes in ready queue and allocates CPU to one of them.

**Preemptive:** Maybe interrupted and moved to ready queue

**Nonpreemptive:** Stays on CPU.

**Dispatch Latency:** Time it takes to stop one process and start another

* Know scheduling criteria: CPU utilization, Throughput, Turnaround time, Waiting time, Response time
* Know how to calculate a schedule under above criteria
* Understand scheduling algorithms: FCFS, SJF, Priority, RR, Multilevel (feedback) Queue
* Know how to draw a Gantt Chart of a schedule under above scheduling algorithms
* Understand starvation and its solution
* Know concepts: asymmetric multiprocessing, symmetric multiprocessing (SMP)
* Understand processor affinity, load balancing, push migration, pull migration

# Chapter 6

* Know basic concepts: race condition, atomic operation, busy waiting, spinlock,
* Understand process synchronization, why it is necessary
* Understand what is critical section problem.
* Know the three requirements for the solution of critical section problem, can prove whether a solution satisfies the three CS requirements
* Understand Peterson’s solution and hardware solution, and know why they are not used
* Understand Mutexs, Semaphore, and Monitor, know the differences between them
* Know deadlock, starvation, and priority inversion problem in semaphore usage
* Understand the three classical synchronization problems: Bounded-Buffer Problem, Readers and Writers Problem, and Dining-Philosopher Problem
* Can fill blanks if a partial solution of any above problem is given.