**Operating System Overview**

* What exactly is the kernel?
  + A kernel is the central part of an operating system. It manages the operations of the computer and the hardware, most notably memory and CPU time. Three basic approaches: Monolithic kernels; Microkernels; and Hybrid kernels. It also runs with the existing process. It also shares a memory-resident operating system code managed by processes.

**C Programming**

* What is a pointer?
  + A pointer is an example of a data type. It indirectly points to any value and stores an address.
* What is Pass by Value?
  + Pass by Value means you are making a copy in memory of the actual parameter's value that is passed in, a copy of the contents of the actual parameter. Furthermore, if the original variable is changed, then it will not affect the copied variable.
* How do you dynamically allocate memory?
  + By using the two key dynamic memory functions: malloc() and free(). The malloc() function takes a single parameter, which is the size of the requested memory area in bytes. By malloc(), it allows us to access more memories and allocate it. It returns a pointer to the allocated memory.

**Assembly and Machine Representation**

* What are registers and how many registers do you have in x86-64?
  + Register: it holds any information about the state of the program. It is comparable to variables that we assigned in C programming to store the values. There are 16 registers in the x86-64 system.
    - What are some of those registers reserved for?
      * The registers are reserved for:
        + Keep the stack track
        + The first program argument in a function
        + The second argument in a function
        + The third argument in a function
        + The program counter
      * The lists above are all dependent on the instructions of the programmer used. They are useful when we're making any system calls.
* What is syscall?
  + It transfers control to the operating system which then performs the requested service. Then control returns to the program.
* How do the Stack and the Heap work?
  + Stack is for static memory allocation. It’s a place in the computer’s memory that stores any temporary variables that the function creates.
  + Heap is for long lived data and is used for dynamic memory allocation. The heap is also being malloced and must be freed to avoid memory leaks.
* What are opcodes?
  + Opcodes are single instruction codes. They are executed by the CPU.
* What do the following instructions do mov, cmp, jmp, and add, etc.
  + MOV: moves data around
  + CMP: conditional code where it compares two operands
  + JMP: performs unconditional jump
  + ADD: adds two operands together

**Compiler and Linkers**

* What does a compiler do?
  + It's responsible to translate code from one language to another
    - Why do we use them?
      * We use them for programs that translate source code from a high-level programming language to a lower level language to create an executable program.
    - What is the frontend responsible for? The backend?
      * The front end's main responsibility is to build an intermediate representation from which code (typically assembly) can be generated. Whereas, the back end is to generate code (10010101...) on our machine.
* Know about Compile-Time and Link-Time Interpositioning.
  + What is the purpose of compile-time and link-time interpositioning and what could it be used for.
    - Compile time: the duration that it takes for the program codes to be executed/translated to machine type code.
    - Link time: it refers to the period of time, during the creation of a computer program, in which a linker is being applied to that program. Link time occurs after compile time and before runtime.
    - Interpositioning is overriding the original code and making changes to it and is used for monitoring and profiling as well as error checking. It also can be used for counting the function calls, tracing malloc(), and characterize for call sites.
* What is a linker?
  + It combines two or more files by taking any files that are generated by the compiler and making it into one executable program.
    - Why do we use them?
      * We use them for space efficiency, time efficiency and modularity.

\*Describe the linking process

* It starts by doing a symbol resolution then it will do the relocation of the symbols.

**Processor and Processes**

* What is a context switch? What information gets saved?
  + It is the process of storing the state of a process or thread, so that it can be restored and resume execution at a later point. This allows multiple processes to share a single CPU, and is an essential feature of a multitasking operating system. The registered information gets saved.
* What are signals?
  + Signals are ways to tell a process that an event has occurred.
* How do you write a signal handler? What goes on when a signal is found?
  + It's written with an int parameter and return type void. When it is found, the handler can either do nothing, unblock, block, or terminate the process.
* What does a fork() do? Why is it a strange function?
  + Fork() is a system call that creates an identical process. It will create copies of the stack, heap, data, and text. It's a strange function because it returns twice instead of two values.
* What does execve(...) do?
  + It executes a program that is specified by the filename.

**The Memory Hierarchy**

* What is pipelining?
  + It is the process of accumulating instruction from the processor through a pipeline. It allows storing and executing instructions in an orderly process. It is also known as pipeline processing. Pipelining is a technique where multiple instructions are overlapped during execution. The process also allows for more instruction to be executed in a short period of time.
* Why is branching bad for performance?
  + Branching will cause the pipeline after the branch to be discarded. The branching is bad for performance because it would cause the CPU to stall when there is a decision that needs to be made.
* What is caching?
  + It is the process of storing copies of files in a cache, or temporary storage location, so that they can be accessed more quickly. Technically, a cache is any temporary storage location for copies of files or data, but usually the term is used in reference to Internet technologies.
* What is a directly mapped cache?
  + It is when each block of main memory is mapped into one cache line.

**Virtual Memory**

* What is Virtual Memory?
  + VM is a feature of an operating system that enables a computer to be able to compensate for shortages of physical memory by transferring pages of data from random access memory to disk storage. This process is done temporarily and is designed to work as a combination of RAM and space on the hard disk.

**Concurrency**

* What is a thread? What is a process? What is the difference?
  + A thread is the simple unit(s) that is allocating the process time.
  + A process is an executing of the program.
  + The difference is that threads share data and code, whereas processes do not share any of that. When comparing the cost – it costs more to manage the processes rather than manage threads.
    - What is fork(), wait(), a child process, zombie processes, orphan processes?
      * Fork(): it makes a copy
      * Wait(): it waits until that its receive a signal or there is an exit of a child process
      * A child process: it's creates by the parents process
      * Zombie process: processes that has finished being executed
      * Orphan process: any processes where its parents already been terminated
* How do you spawn a thread, and what does it mean to join a thread?.
  + By doing pthread\_create to spawn a thread. By joining a thread, it will make the main thread wait until all the threads are executed.
* Talk about the problems with concurrency and sharing memory.
  + There are several problems with concurrency, such as:
    - Data race: It occurs when there are multiple processes all want to share one resource.
    - Deadlock: It occurs when there are improper resources being allocated.
    - Starvation: When there is a situation in which a runnable process is overlooked indefinitely by the scheduler; although it is able to proceed, it is never chosen.
    - Memory is the resource that is causing these problems with concurrency.
      * Example:
        + Lab 4 is a great example of a starvation.