Thursday, 2pm. Be there at 1:45.

No calculators.

2hr test.

**Chapter 1**

Question 1: 30 marks

-**Four components of a computer system**:  
Hardware, OS, Application programs, Users (know what each 4 are; form a hardware perspective, how do we use/ how does computer hardware work together – eg Notepad, what happens when you click save. Info saved to memory, then application (from CPU), through bus to disc controller to save text file

Know how each hardware component interact with each other, and explain how it works.

**Compute System Operation**: These have some narratives

**Storage Structure**: Each computer has memory. RAM, volatile, speed, price. Cache, hard discs (form of memory).

**Storage Device Hierarchy**: This hierarchy acts as a cache, (eg main memory works as cache for SSD). Cache in it is CPU Cache (2nd level).  
A thing acts as a cache for a bigger storage (lower level) medium.

When you run a program: how they work together.

**Computer system architecture:** Running multiple processes at a time.  
Multi-Processes, or multi-Core: asymmetric and symmetric multiprocesses

**OS does many things:**

**Process Management**

Create, suspend…

**Memory management**: gives a process allocated memory for use. Ensures that eg Dyanmmic array doesn’t override allocated memory.  
Security + protection

**Chapter 2**: 20 marks

**Operating System Services  
-**Consider how user interact with OS.   
CMD great for higher level of control  
GUI: simpler to use, don’t need to know instructions/commands, more user friendly  
-advantages, and why we want to use

**System call implementation**

How does a process work.  
Application in user mode, has no permission to access hardware.

System call goes into section in computer memory, and executes code. It looks in a table to find the system call ID. (**API – system call implementation**)

**System call parameter passing**

3 types. Registers, block of memory with address, stack

**Implementation**

When we implement OS’s, we have some level of assembly. But normally most of the body of the OS is written in C and C++. File manager, system task application in any language.

Different designs/approaches

**Layered Approach (there was one other above or below?)**

Components can only access above and below layers

**MicroKernal system structure**

**Modules**

More modern approach. Instead of .. Have component made available, which the kernal hooks into

**System Boot**

Bootstrap loader is stored in ROM. This program knows how to interact with hardware.  
1st approach: Bootstrap loader locates kernel files and loads it

2nd approach: more modern: Reads boot block of hard disc. Boot block starts a bootstrap loader, which allows to choose which OS to run.

**Chapter 3: 25 Marks**

**Process Concept**

The parts a process consists of, list and explain them.  
text: code  
stack: temp variables  
Program counter: address of next instruction line  
data:  
heap:

**Diagram of process state:**

Know diagram well.  
Ready, running and waiting states are important.  
Ready and waiting, you are not actively doing anything. Only running is doing something.  
When running, you can go to waiting or ready.  
Goes to waiting when hardware/IO used, saving files.  
While in waiting queue, hardware doing stuff, then once hardware says its done, process generates an interrupt to go from waiting to ready.

Scheduler is what picks what goes from ready to running.  
Different types of schedulers (medium and short term).

**Process Control Block (PCB)**

When process changes state (ready to running eg): a Context switch occurs. Put state in PCB, save PCB, then start next process. For that process, must first get PCB from memory, then load it before it can start running. (**This is a switch between 2 processes)**

**Message passing**Direct or Indirect. --Know the properties of these 2.  
**Indirect** **Comunication**: mailbox  
**Direct** **Communication**: directly to eachother

Direct: must know name of process

Indirect: can have many processes sending to mailbox, bidirectional.

**Communication in client server environment, different solutions**

**Sockets**

Port numbers: don’t use less than 1024

**Socket Communication**

**Remote Procedure Calls (RPCs)**

Matchmakeer: central process can use to communicate with RPC. It is a guide that tells other client process servers.  
Client server communicates with matchmaker, then matchmaker replies with port number that indended port is listen to.

Matchmaker keeps track to the (dynamic) ports each system is connected to.

**Pipe**

Simplistic link between 2 process. So that the output of one process, can be used by the other.

Q4: 10 marks

Prac lectures on conversions. (know negative numbers)

Q5:  
Function, assembly

Must know:

Negative numbers, conversions, functions, working with negative numbers. (prac side).