<https://compas.cs.stonybrook.edu/~nhonarmand/courses/fa17/cse306/schedule.html>

^ class almost exactly like ours, better slides

# Exam 1 Review Sheet

Table of Contents

[Exam 1 Review Sheet 1](#_Toc526980765)

[01-Intro 2](#_Toc526980766)

[02-Virtualization CPU 3](#_Toc526980767)

[03-Scheduling 4](#_Toc526980768)

[04-Virtualizing Memory 7](#_Toc526980769)

[05-VM Paging 10](#_Toc526980770)

[06-VM TLB 11](#_Toc526980771)

[07-VM smaller page tables 11](#_Toc526980772)

[08-VM beyond physical 11](#_Toc526980773)

[09-Threads 11](#_Toc526980774)

[Notes From Review Day 11](#_Toc526980775)

## 01-Intro

**Operating System:** interface between hardware and application \* Software that makes hardware useful for applications.   
  
**Operating System** provides:

* **Abstraction**
  + Makes different devices look the same
  + Higher level functionality
* Abstraction in what form???
  + CPU: Processes
  + Memory: Address spaces
  + Disk: Files
* Resource Management: Share resources well
  + Advantages:
    - Efficient Use of resource
    - Fair Use of resource
    - Protect one application from another
* Challenges: Policy & Mechanism

**Three Pieces**

* **Virtualization**: Make every process think it has the CPU to itself
* **Concurrency**: OS must handle simultaneous processes/events. Easier if they are independent. Trickier if they interact
* **Persistency**: information is accessed permanently. Provide abstraction so that the application doesn’t need to know where the data is stored. Performance and Handle Failures.

**Design Goals:**

* **Efficiency: (low level view)** A Hardware management library
* **Better Usability: (high level view)** Physical machine to an abstract one**.**
* **Protection**: Processes are separate from another

**Other Design Goals**

* **Reliability**
* **Security**
* **Mobility**

## 02-Virtualization CPU

Process is a program in execution.   
Program is **STATIC**. Process is **DYNAMIC.**

## 03-Scheduling

**Definitions and Acronyms**

Time slice – length of time process has the CPU

longer = worse response time & few context switches

shorter = better response time & more context switches

Interactive Jobs – need I/O operations and care about response time

Batch Jobs – only care about turnaround time

**Formulas**

Turnaround Time = Completion Time – Arrival Time

Response Time = Time of First Run – Arrival Time

**Concepts**

Sharing the CPU

Why?

How?

Must have a mechanism which is the dispatcher, and a policy which is the scheduler.

FIFO – first in first out, also called FCFS, first come first served

SJF – shortest job first

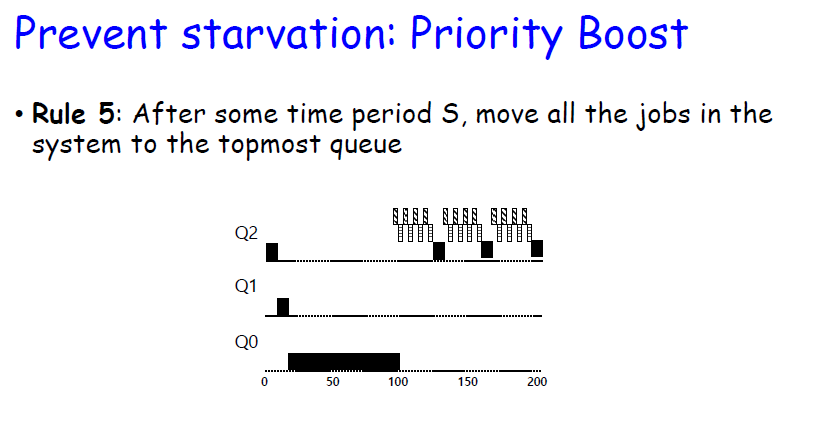
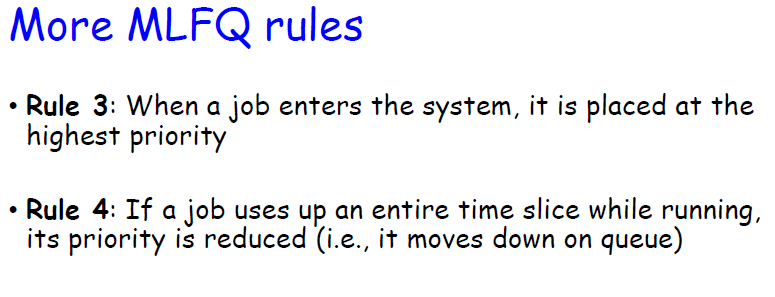
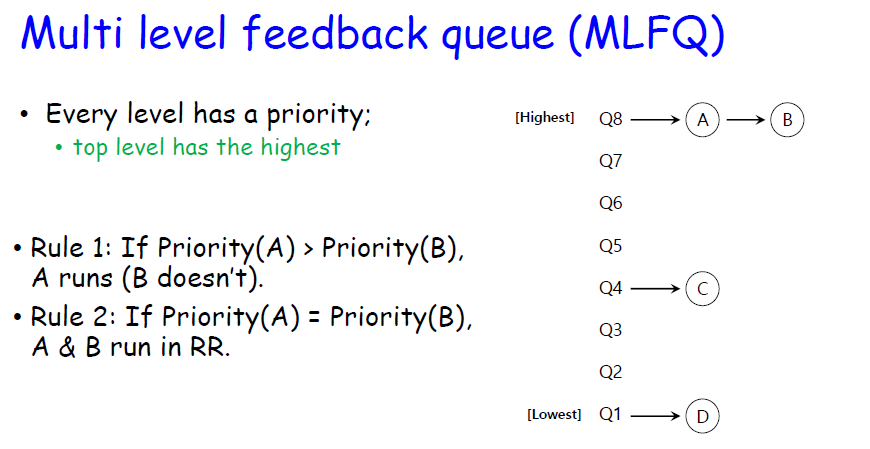
STCF – shortest time to completion first

RR – round robin

MLFQ – multi level feedback queue, top level has highest priority

Lottery - each process gets tickets, whoever wins lottery runs

|  |  |  |
| --- | --- | --- |
|  | pros | cons |
| FIFO | easy to implement | short jobs MIGHT have to wait a long time  bad response time |
| SJF | good turnaround of short jobs | short job that arrives AFTER a long job causes starvation  bad response time |
| STCF | good turnaround of differing arrival times | jobs that use I/O cause starvation  bad response time |
| RR | good response time  doesn't care about run time | does not have a mechanism to adjust to the process at hand |
| MLFQ | mimics SJF  good response time for interactive | many interactive jobs will starve the long jobs  It's possible game the scheduler by giving up CPU before time slice is done |
| Lottery | fair sharing  simple to implement |  |



## 04-Virtualizing Memory

Summary:

Anytime we must share a resource on the computer between processes, we have to virtualize it so the process has the illusion that the resource is only available to itself. The hardware and OS work together to virtualize memory, each doing what it’s best at. Hardware for simple tasks that must be fast and OS for keeping track of who’s memory space contains what. The MMU translates base register + bounds to map a virtual address to a physical one so the translations are fast. OS only gets involved during context switches and errors.

After these concepts, we have to figure out how to handle segmentation \*hint paging solves this\*

**Definitions and Acronyms:**

Fragmentation: ineffective use of memory

Internal Fragmentation: Process does not use all the space allocated to it.

External Fragmentation: There is adequate total space for a process BUT it is scattered in chucks throughout the virtual address space.

Paging: memory contains commonly accessed values, loaded for use immediately

Allocation: “to distribute”, sets limits for program where virtual memory is mapped to physical memory

Segmentation: divide each process into separate boxes of memory needs, stack head data code, each can grow or shrink independently of the others

MMU: Memory Management Unit, hardware implementation that maps a logical address to a physical address by ADDING the base register to the logical address

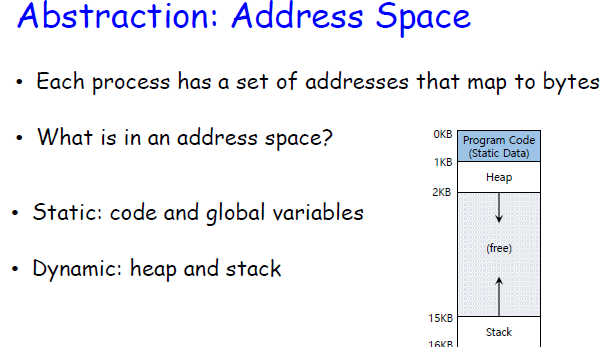
Dynamic Relocation: moving a currently executing process to a new location

**Formulas**

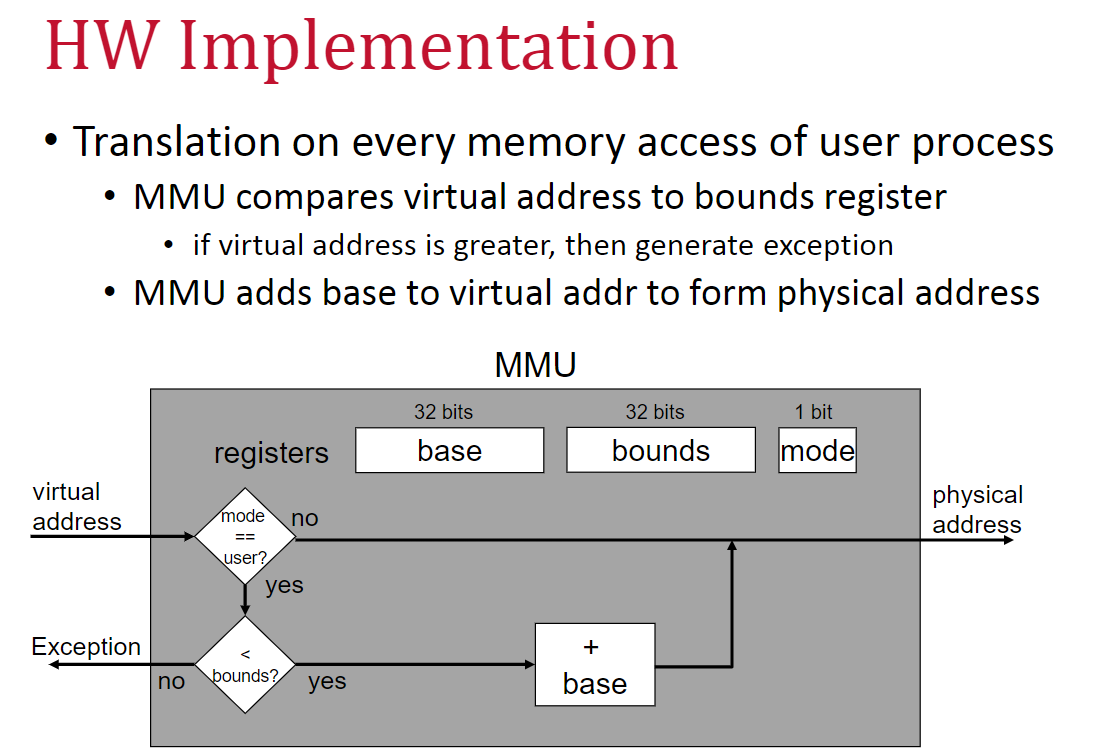
* **Segmentation Address Translation:**  *Physical Address = Offset + Base*

**Concepts**

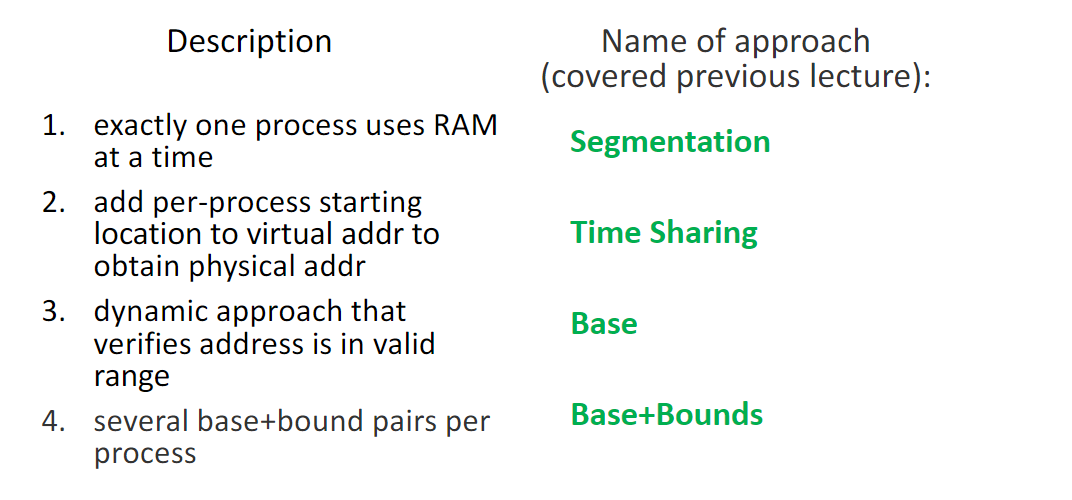
* Abstraction of Memory with the Address Space
  + View image below to answer bullet questions
  + What goes in the program code?
    - Main() and all other programs
  + What goes in heap?
    - Pointers to objects
  + What goes in (Static Data)?
    - Variables for programs, like counter variable x for program Main()



* **Context Switching with base-and-bounds**
  + Base and Bounds are a part of the process’ context
  + Store the base-and-bound to old process control block (PCB)
  + Restore them from the new process’ PCB.
  + View image below for visualization of concept



* **Base and Bounds**
  + **Advantages:** Easy to implement with Hardware. Dynamic Relocation. Protection
  + **Disadvantages:** Processes require contiguous memory, Internal Fragmentation, External Fragmentation.
  + Questions
  + Does every process have its own unique value in its base register?
    - Yes, the base register is the start of the address space for the process
  + Who modifies the base register for the process?
    - OS
  + Who decides when to relocate a process?
    - OS
  + Who generates the virtual address?
    - The process itself
  + Who maps the virtual address to physical?
    - Hardware, with the MMU
* **Solution to Base and Bounds: SEGMENTATION** 
  + Address space for a process is divided into *segments*.
  + Every segment contains an element of address space such as heap, stack, code, data.
    - Segments are variable in size.
    - Independently placed in memory.
    - Each segment is protected
  + Advantages: Segment Sharing. Easier to relocate a segment versus the entire process/program.
  + Disadvantages: Fragmentation isn’t solved. Segment’s variable size can cause them to be large.



1. ---> **Time Sharing**
2. ---> **Base**
3. ---> **Base+Bounds**
4. ---> **Segmentation**

## 05-VM Paging

**Definitions and Acronyms:**

Frame: fixed size block of physical memory

Page: frame sized portion of virtual memory

Paging: memory contains commonly accessed values, loaded for use immediately

VPN: Virtual Page Number

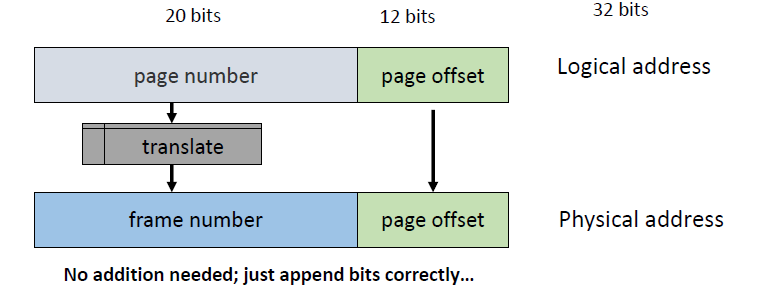
PPN: Virtual Page Number

**Formulas**

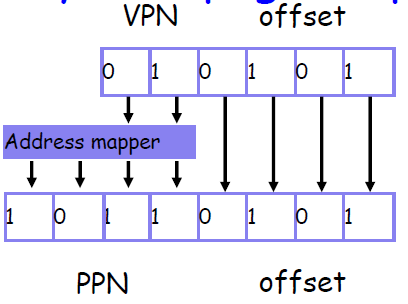
* **Segmentation Address Translation:**  *Physical Address = Offset + Base*
* **Frame Size** *= a power of 2, between 512 and 16 Mbytes*
* Offset *= ln (Page Size in Bytes) / ln (2)*
* Page size in Bytes *= 2 (offset)*
* Bits for VPN *= Virtual Address Size – Offset*
* Number of VPN Entries *= 2(bits for VPN)*
* Page Table Size *= Number of VPN Entries \* VPN Entry Size*

**Concepts**

* **Paging: Virtual Address to Physical Address**
  + Uses translation and offset to switch between virtual and physical addresses
  + Virtual Page size to offset
    - Assuming page size 512 Bytes
    - Both these formulas work
      * 512 Bytes *= 2 (offset)* => 9
      * *ln (*512 Bytes *) / ln (2)* => 9



* **Mapping: Virtual Address to Physical Address**
  + **The virtual address and physical address DO NOT need the same number of bits**
  + **The address mapper can expand the virtual address to reach whatever physical destination is required**



* Page Table
  + Storage of pages
    - Hardware finds it in base register CR3 on xv6
  + Context Switch:
    - Save old page table base register into PBC of descheduled processes, so when it’s reloaded later, it can pick up from where it left off.
    - Change contents of page table base register to the newly scheduled process
  + PTE:
    - A page table entry is simply a series of bits
    - Valid bit, protection bit, present bit, reference bit, dirty bit
    - Agreement between OS and hardware on what each bit means
* Summary
  + Paging
    - **Advantages**
      * Any page can be loaded straight into a memory frame
      * Fast allocation
      * Fast free with bitmap to see what’s open
    - **Disadvantages**
      * Wasted memory with larger pages (internal fragmentation)
      * Overhead of page table loaded into memory
        + Conversely, the MMU only needs the Page Table Base Register

## 06-VM TLB

Summary:

This slide deck is about solving the problem of page table ACCESSES by caching common values in a Translation Lookaside Buffer. The buffer contains entries that map from the current process's virtual memory, this is a CRITICAL POINT. During a context switch, it is possible that information pointed to by the TLB will be unloaded from virtual memory. So, the options are to flush this data out entirely, or lock it somehow. Obviously flushing the cache defeats the purpose of having one. So, to maintain the full cache entries are locked with another bit, called the ASID. This allows the full cache to be maintained over context switches of many processes, while locking the cache to processes that do not have access to the address space.

**Definitions and Acronyms**

Spatial Locality – future reference to nearby addresses

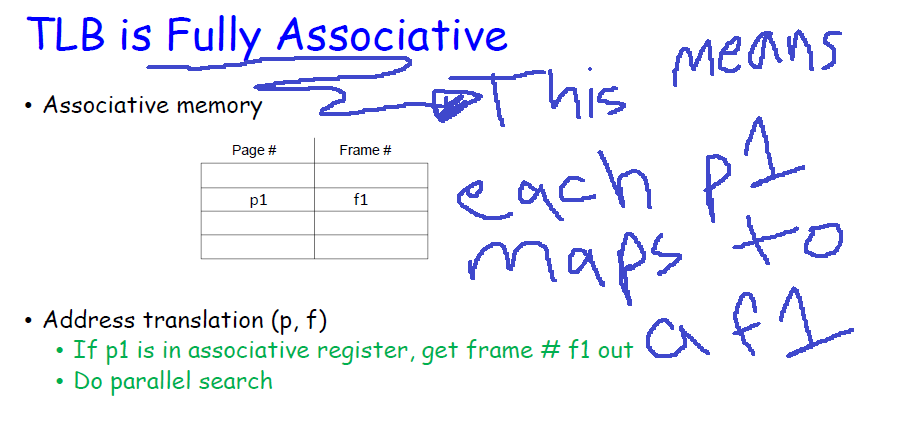
Temporal Locality – future reference to same data

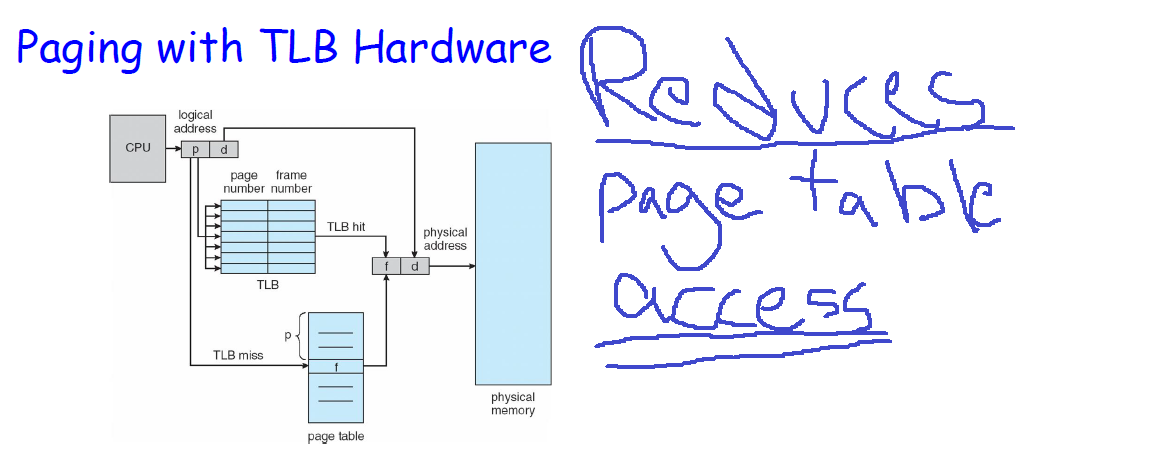
**Formulas**

**Miss rate = miss / lookups**

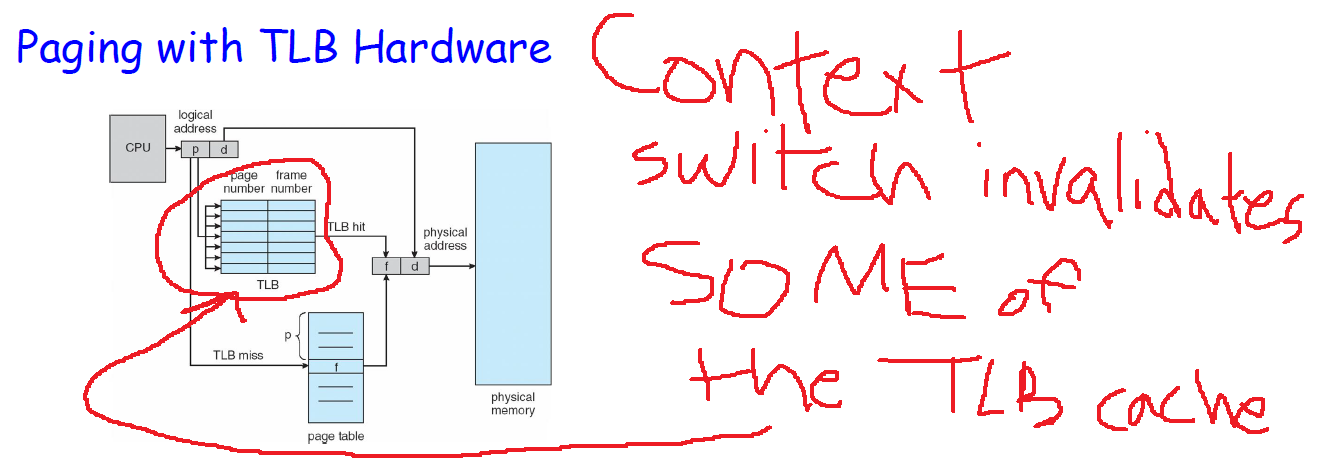
**Concepts**

* Fully associative cache
  + The nature of the table also cuts out the translation step of the MMU
  + P1 maps straight to F1, no translating needed





* Handling a Miss:
  + HW
    - Knows base address of Page Table (CR3)
    - PT structure is fixed and agreed on between HW and OS
    - Walk page table and fetch entry
  + OS
    - Interrupt
    - Software Managed TLB Fetch
* Replacement Policies
  + Random
  + LRU
* Context Switch:
  + Share the TLB across a context switch with the Address Space Identifier (ASID)
  + Process is only allowed to use a TLB entry IF the ASID allows the process to do so
  + Any ASID that blocks a process from using the TLB results in a miss, since that data is invalid to the process



## 07-VM smaller page tables

**Definitions and Acronyms**

**Formulas**

**Concepts**

## 08-VM beyond physical

**Definitions and Acronyms**

**Formulas**

**Concepts**

## 09-Threads

**Definitions and Acronyms**

**Formulas**

**Concepts**

## Notes From Review Day

Exam Review

Big picture

Virtualization

CPU

Memory

How fork works

Process creation

process deletion

waiting for child

Schedulers

How different one’s work

Comparison chart

turnaround

response time

calculation times

Paging

how paging work

translating virtual page to actual page

relationships between variables

if page size increases, what happens to page table

A modern OS virtualizes CPU by time sharing

A process is a program in execution

A thread is

You must change from user mode to kernel mode to execute syscall

More context switches, more overhead

Virtual page is an imaginary copy of the physical frame

Offset formula

Know what threads share and have of their own

Lexicographic - alphabetical

5 States in proc.h

EMBRYO, READY, RUNNING, BLOCKED, ZOMBIE

Know how to draw Gantt charts

Know definitions for scheduling

Know formulas for scheduling

Part 1

T/F

Part 2

program output

Part 3

virtual memory

Part 4

paging