COMP 3500: A Study Guide for Midterm 1

A computer system can be divided roughly into four components: the hardware, the operating system, the application programs, and the users

Kernel - is the one program running at all times on the computer

**Kernel Mode.**

* + Executing code has complete and unrestricted access to the underlying hardware.
  + Is reserved for the lowest-level, trusted functions of OS.
  + Crashes in this mode are catastrophic.

**User Mode**

* + Executing code has no ability to *directly* access hardware
  + Code must delegate to system APIs to access hardware
  + Crashes in user mode are always recoverable

User Goals

* Convenient to use
* Easy to learn

System Goals

* Easy to implement
* Easy to maintain
* Easy to design

Both

* Flexible
* Error-free
* Efficient and Fast
* Safe
* Reliable

1. OS Overview (Ch 1.1-1.6 and Ch 2.1-2.7)
   1. What is an operating system?
      * program that manages a computer’s hardware
      * Resource manager. It hides details of how underlying machinery operates.
   2. System view of the OS
      * In user view resource utilization is not a primary goal.
      * Computer and Software
      * Resource allocation: CPU, memory, disk, and the like
      * Control programs
      * Resource sharing
   3. Goals of an OS
      * Convenience for user
      * Efficient operation of computer systems
   4. Operating System Strategies
      * Batch systems
      * Multiprogrammed batch systems
      * Time-shairng systems
   5. Monilithic kernel vs. Microkernel
      * Mircokernel
        + This method structures the operating system by removing all nonessential components from the kernel and implementing them as system and user-level programs. The result is a smaller kernel.

This method structures the

operating system by removing all nonessential components fromthe kernel and

implementing them as system and user-level programs. The result is a smaller

* + - kernel.

1. Processes (Ch 3.1-3.4)
   1. Concept
      * Process vs. program
        + Program – not running
        + Process – running
      * Process states
      * Process Control Block (PCB)
   2. Process Control
      * Creation
      * Termination
      * Process state transitions
   3. Process states
      * Two-state process model
        + Creation and termination
      * Five-state model
        + Suspended processes
        + **New** The process is being created.
        + **Running** Instructions are being executed.
        + **Waiting** The process is waiting for some event to occur (such as an I/O

completion or reception of a signal).

* + - * **Ready** The process is waiting to be assigned to a processor.
      * **Terminated** The process has finished execution.

1. Synchronization (Ch 5.1-5.8)
   1. Motivation: an example
      * The critical-section problem
   2. Synchronization hardware
      * TestAndSet
        + boolean test\_and\_set(boolean \*target) {

boolean rv = \*target;

\*target = true;

return rv;

}

* + - Swap instruction
      * int compare and swap(int \*value, int expected, int new value) {

int temp = \*value;

if (\*value == expected)

\*value = new value;

return temp;

}

* 1. Semaphores: synchronization tool
     + Three operations
     + Definition of semaphore primitives
     + Mutual exclusion using semaphores
     + Solving synchronization problems using semaphores
  2. Monitors: Concept
     + monitor type is an abstract data type (ADT) that includes a set of programmer defined operations that are provided with mutual exclusion within the monitor
  3. A case study: Cats-Mice Problem

1. Projects: OS/161
   1. Thread questions
      * What happens to a thread when it exits (i.e., calls thread\_exit() )? What about when it sleeps?
        + When a thread exits, it ensures the stack isn’t damaged, destroys

the virtual memory space, decrements the counter of whatever vnode it may be poitning at, puts itself into S\_ZOMB,

and preps itself to panic incase it runs before it dies When it sleeps, it makes sure it’s not in an interrupt

handler, yields control to the next thread, enters the S\_SLEEP

state, and only starts taking control once more when wakeup() is called on its address.

* + - What function(s) handle(s) a context switch?
      * mi\_switch and md\_switch
    - How many thread states are there? What are they?
      * four - S\_RUN, S\_READY, S\_SLEEP, and S\_ZOMB
    - What does it mean to turn interrupts off? How is this accomplished? Why is it important to turn off interrupts in the thread subsystem code?
      * When you turn off interrupts they cannot be triggered. They are turned off using the function splhigh and back on again using spl0 Turning off interrupts for thread operations is necessary to ensure that these operations complete successfully and aren’t broken mid-execution.
    - What happens when a thread wakes up another thread? How does a sleeping thread get to run again?
      * It removes it from the sleeping queue and adds it to the running queue. It calls mi\_switch.
  1. Scheduler questions
     + How does that function pick the next thread?
       - struct thread \* scheduler(void);// The function
       - round-robin run queue that schedules each thread in the queue without priorities
     + What role does the hardware timer play in scheduling?
       - interrupt handler for the hardware timer calls hardclock; thread\_yield
  2. Synchronization Questions
     + What is the purpose of the argument passed to thread\_sleep()?
       - thread\_sleep causes the program to pause until the semaphores are greater than 0; thread\_wakeup() causes the processes to wakeup

the argument that is pass in is the address.

* + - Why does the lock API in OS/161 provide lock\_do\_i\_hold(), but not lock\_get\_holder()?
      * locks have to be released by the same thread that acquired them