**CSE381 Midterm Exam Study Guide**

**1 Textbook Coverage**

1. Chapter 2

Section 2.1, 2.2, 2.3, 2.4, 2.5, 2.6 and 2.7.

* The concept of process, process control block, process elements, the difference between process and program

A **process** is the execution of an application program with restricted rights; the process is the abstraction for protected execution provided by the operating system kernel.

A **PCB** is a data structure that stores all the information the operating system needs about a particular process: e.g., where it is stored in memory, where its executable image is on disk, which user asked it to start executing, and what privileges the process has.

* User mode, kernel mode, privileged instruction, memory protection

* The concept of interrupt, polling, system call, processor exception, the reasons to cause mode transfer

An **interrupt** is an asynchronous signal to the processor that some external event has occurred to require its handling.

A **system call** is any procedure provided by kernel that can be called from user level.

A **processor exception** is Hardware event caused by user program execution

User -> Kernel: Interrupts, Exceptions, System calls

Kernel -> User: New process, return from interrupt/exception/system call, Process context switch, User-level upcall

* Interrupt vector table, interrupt handler, interrupt handler, two stacks per process

**Interrupt vector Table** set up by OS kernel; pointers to code to run on different events

An **interrupt handler** is the term used for the procedure called by the kernel on an interrupt

**Interrupt masking** prevents interrupts from being delivered at inopportune times.

* Mode transfer implementation (section 2.5)

You need to understand every details regarding mode transfer implementation described by textbook.

1. Chapter 3

This chapter introduces a set of important system calls in the linux/unix operating system. You need to understand the function of these system calls and how to use these system calls to solve problems. Pay close attention the parameters in these system calls.

Section 3.1, 3.2, 3.3 and 3.4

* You need to know how to create a process in Unix with fork( ).

Steps to implement UNIX fork

Create and initialize the process control block (PCB) in the kernel

Create a new address space

Initialize the address space with a copy of the entire contents of the address space of the parent

Inherit the execution context of the parent (e.g., any open files)

Inform the scheduler that the new process is ready to run

The relationship between forked process and parent process.

The Unix execv( ), wait( ), kill( ), signal( )

int kill(int pid, int signal)

a system call that send a signal to a process, pid. If pid is greater than zero, the signal is sent to the process whose process ID is equal to pid. If pid is 0, the signal is sent to all processes, except system processes.

int raise(int sig)

sends the signal sig to the executing program. raise() actually uses kill() to send the signal to the executing program: kill(getpid(), sig);

* Basic idea of Unix I/O interface

Uniformity

All operations on all files, devices use the same set of system calls: open, close, read, write

Open before use

Open returns a handle (file descriptor) for use in later calls on the file

OS checks permission and sets up internal bookkeeping

Byte-oriented

All devices, even those that transfer fixed-size blocks of data, are accessed with byte arrays

Kernel-buffered read/write

Stream data stored in kernel buffer and returned to app on demand.

Process to call read( ) blocks if data is not available

Process to call write( ) blocks if writing data too fast than device can receive.

Explicit close

To garbage collect the open file descriptor

* + Open( ), read( ), write( ),
  + Concept of file descriptor used in the read( ) and write( )
* Unix pipe system call pipe( )

Understand the concept of I/O redirection (using dup2(). Remember to store standard output before calling dup2)

**2 Question Types**

The midterm could include:

1. Short answer question

One example of short answer question is to write the output of a program.

1. Fill in blanks questions

One example of fill in blanks question is to finish the missed statement of a program.

1. Ture or false questions
2. Analysis/explain questions

**3 How to prepare it**

1. In order to save time to read textbook, read slides along with notes you wrote down in class. However, please read textbook if you don’t understand a specific detail.
2. Review/redo homework, project and lab.

**4 Midterm time**

March 1st (Wednesday) one hour