Jay Patel
CS 352 Exam1 Notes

Topics:
Overview
Process
Thread
Scheduling

Overview

1. Interrupts:
    a. What are interrupts used for?
    Answer: An event signal that forces the CPU to stop and resume execution from a fixed location
    b. How does it work?
    Answer:
    - **Completes current instruction**
    - **Transfer execution to Interrupt Service Routine (ISR)**
    - **ISR usually disables interrupts and processes current one**
    - **When ISR completes, resume interrupted computation.**

    c. Types
    Answer: System call, times, disk and program exception
    d. Examples
    Answer: Device controller informs CPU that it has finished an I/O operation by causing an interrupt the CPU can move data between memory and controller's local buffer.
2. Dual Mode execution
    a. What are privileges instructions?
    Answer: operations that can only be executed in kernel mode
    Example: file management, process control, communication, device management, and information maintenance.
    b. What is kernel mode?
    Answer: When the user requests a service from the OS (via a system call) the OS switches to kernel mode to complete the task. A protected mode for the OS only. Mode Bit = 0

c. What is user mode?

Answer: When the computer system is executing on behalf of the user it is in the user mode.

d. When mode switch is needed?

Answer: when the user makes a request of switching OS

3. How to protect memory?

Answer:

Limit registers: Contains the size of a user programs range.

Base registers: The registers containing the smallest legal address of a user programs range.

4. System call

a. Why system calls are need?

Answer: Because we have dual mode execution, it will allow the user to make request for the OS to perform functions that are privilege to OS

b. How are system calls implemented?

Answer:

<u>Kernel mode:</u> It associates a number with each system calls. Maintains a table according to these numbers, each entry of the table points to the systems service code.

<u>User mode:</u> System call interface is provided by the run time support system.

c. Examples of system calls?

Answer: Fork, clone, wait, exit

5. Major components of OS

Answer:

- **Manage execution resources**
- **Manage memory space**
- **Manage permanent storage space**
- **Manage I/O devices**
- **Manage program**
- **Support communications between programs and between computes**

Process

1. Structure of process: user space and kernel space

Answer:

User space: Text section, Stack, Heap, Data section.

Kernel space: Process control block.

2. Process creation: how fork() works

Answer: Creates the child process and returns 0 to child process and parents returns the child process ID.

3. Process termination: exit(), kill()

Answer:

Exit() Process executes last statement and asks the OS to delete it (exit) and then the processes resources are de-allocated.

Kill() Parent may terminate execution of children processes

4. Inter process communication mechanism
    a. **Two basic mode:**
    b. **Pipe: Similar to reading from/ writing to a file. Pipes are limited to processes with a common ancestor and are non-permanent**
    c. **Shared memory: Allows communication between process through a shared region of memory.**
    d. **Signal:**
5. Relation between pipes and standard I/O?

Answer: A pipe is a natural structure for connecting the standard input of one program to the standard output of another.

Thread

1. Internal structure of multi thread process

Answer:

2. Kernel thread: how clone() works

Answer: Linux Kernel thread creation is done with the clone() system call.

3. User thread: pthread library (basic functions)

Answer: User threads common in UNIX OS

Pthread_join: A way to accomplish sync between threads.

Blocks the caller until the specified thread terminates

1st arg: The ID of the thread to be waited

2nd arg: The address of the variable to receive the thread exist status. Usually set to NULL or int

p_exit: Terminates execution of the calling thread.
Pthread_self: returns the pthread handle (ID) of the calling thread
Pthread_create: creates a new thread
1st arg: the address of the created thread
2nd arg: the struct containing attributes of the created thread.
3rd arg: The pointer to the function that is the code for the thread.
4th arg: Pointer to the arguments for the function.

4. Mapping from user thread to kernel threads (deas)

Answer:

Two level models: A sub type of Many to many allows a user thread to be bound to a kernel thread.
One-to-one: Each user level thread maps to a kernel thread. Upon a thread currently running on a CPU yields or bocks, CPU can be switches to another thread.
Many to Many: Allows many user level threads to be mapped to many kernel threads. Upon a currently running thread yields another thread may be mapped to the kernel thread. Upon a currently running user thread blocks other thread may run on the CPU.
Many to one: Many user level threads with in the same process are mapped to single kernel thread. The multiple threads "time-sharing" the single kernel threads.

Scheduling

1. Internal data structure to support scheduling

Answer:

a) **Process control block**: A data structure used to store the context of a process which is:
   - Value of program counter register
   - Value of stack pointer register
   - Value of other registers
   - Process state, memory info, I/O

b) **Ready queue**: The queue of the PCBs of all processes residing in main memory, ready and waiting to execute.
c) **Job Queue**: The queue of the PCS of all the processes in the system
d) **Device queue**: The queue of PCBs of processes waiting for an I/O device.

2. Concept of contexts

Answer: A context is the state of a process when it is running.

3. Context switch

Answer: When the CPU has to switch from one process to a new one it saves the old state and loads the state of the new one via context switch.

4. Basic scheduling algorithm: FCFS: **First served is process is served in the order its requested. However, is sub optional in that big process can increase the average wait time of all the process through the commoy effect.**
5. SJF: **The process with that shortest execution time is served first and is optimal for shortest waiting time. Downside is you cannot know how long a process will take to execute before its run.**
    a. Preemptive: **Current will stop. High priority will run.**
    b. Non-preemptive: **Current will run, high priority will stop.**
6. Priority: **Process are served by the CPU based off an assigned priority.**
    a. Starvation: **In a priority scheduling system, low priority processes may never execute.**
    b. Aging: **Priority scheduling system priority of a process will increase over time.**
7. RR: **Ready process are served in time slices (often 1 ms) if the process takes, more than the slice it is added to the end of the queue. If the quantum is too it degrades to FCFS, too small and there is too much overhead from context switch.**
8. Time Quantum for RR: **A slice of CPU time user in RR scheduling. The time quantum must be not for large and not too small. Too large and RR degrades to FIFO, too small and context switch overhead is high.**
9. Multi-level queue scheduling

Answer: Ready queue is portioned into multiple queues for ground(interactive) and background (batch)Each queue has its own scheduling algorithm.

    a. **Foreground: RR**
    b. **Background: FCFS**

10. Multi-processor scheduling
    a. Process affinity: **Process has affinity for processor on which it is currently running.**
    b. Hybrid: **A process migrates only among a certain processor set.**

c. Hard affinity: **A process does not migrate between different processors.**
d. Soft Affinity: **attempting to keep a process running on the same processor, but not guaranteeing that it will do so.**

11. Quantitative analysis of performance:
    a. waiting time: **Amount of time a process has been waiting on the ready queue.**
    b. turnaround time: **Amount of time to execute a particular process (from creation to completion)**
    c. Throughput: **A CPU scheduling criteria (process/time-unit)**