Operating systems

Linux commands

- Man -> Manual
- Crt+z -> suspends process from foreground to background
- Fg -> takes last process you sent to the background
- Bg -> runs process in the background
- Grep -> searches
- Kill -9 -> kills process must enter number
- Ps -ef -> processes currently running
- l -> pipes output of one command to input of another
- % <job number> -> enables job number as identifier

Representation of process scheduling

- Queueing diagram represents queues, resources, flows
  - Ready queue
    - Goes to cpu
      - i/0 request -> i/0 queue -> i/o
      - Time slice expired
      - Fork a child -> child executes
      - Wait for an interrupt -> interrupt occurs

schedulers

- Short term scheduler (or CPU scheduler)
  - Selects which process should be executed next and allocates CPU
    - Sometimes the only scheduler in a system
    - Short term scheduler is invoked frequently (milliseconds) must be fast
- Long term scheduler (or job scheduler)
  - Selects which processes should be brought into the ready queue
    - Long term scheduler is invoked infrequently (seconds, minutes)
    - Long term scheduler controls the degree of multiprogramming
- Processes can be described as either
  - i/o bound processes
    - Spends more time doing i/o than computations many short cpu burst
  - Cpu bound process spends more time doing computations
- Medium term scheduler
  - Can be added if degree of multiple programming needs to decrease
    - Remove process from memory, store disk, bring back in from disk to continue execution: swapping
      - Ready que <- Swipe in <- Partially executed swapped out processes -> swap out
        - Cpu -> i/o waiting queues -> i/o ->ready queue

Multitasking in mobile systems

- Due to screen real estate user interface limits ios provides for a
  - Single foreground process
    - Controlled via user interface
  - Multiple background processes
    - In memory, running, but no on the display and with limits

Context switch (processes control block)

- When CPU switches to another process the system must save the state of the old process and load the saved state for the new process via a context switch
- Context of a process represented in the PCB

- Context switch time is overhead; the system does no useful work while switching
    - The more complex the OS and the PCB the longer the context switch
- Time dependent on hardware support
    - Some hardware provides multiple sets of registers per CPU -> multiple context loaded at once

Operation on processes

- process creation
- Process termination

Process creation

- Parent process create children processes which in tuen create other process forming a tree of processes
- Generally process identified and managed via a process identifier (pid)
- Resources sharing options (memory files devices)
    - Parent and children share all resources
    - Children share subset of parents resources
    - Parent and child share no resources
- Execution options
    - Parent and children execute concurrently
    - Parent waits until child terminates
- Address space
    - Child duplicate of parent
    - Child has a program loaded into it
- UNIX example
    - Fork() system call creates new process
    - Exec() system call used after a fork() to replace the process memory space with a new program

C program forking separate process linux

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
pid_t pid;
	//fork a child process
	pid=fork();

	if (pid < 0)
	{
	// error occurred
		fprintf( stderr, “Fork Failed”);
		return 1;
	}
	else if (pid == 0)
	{
	//child process
		execlp(“/bin/ls”,”ls”, NULL);
	}
	else
	{
	//parent process
	//parent will wait for the child to complete
		wait(NULL);
		printf(“Child Complete”);
	}

	return 0;
}
```

Creating a separate process in windows api in C

```
#include <stdio.h>
#include <windows.h>

int main (VOID)
{
STARTUPINFO si;
PROCESS_INFORMATION pi;
	//allocate memory
	ZeroMemory(&si, sizeof(si));
	Si.cb = sizeof(si);
	ZeroMemory(&pi, sizeof(pi));

	//create child
```

Process termination

- Process executes last statement and then asks the operating system to delete it using the exit() system call
    - Returns status data from child to parent (via wait())
    - Process resources are deallocated by operating system
- Parent may terminate the execution of children processes using the abort() system call some reasons for doing so:
    - Child has exceeded allocated resources
    - Task assigned to child is no longer required
    - The parent is exiting and the operating systems does not allow a child to continue if its parent terminates
- Some operating systems do not allow child exists if its parent has terminated if a process terminates then all its children must also be terminated
    - Cascading termination all children grand children ect are terminated
    - The termination is initiated by the operating system
- The parent process may wait for termination of child process by using the wait () system call the call returns status information and the pid of the terminated process
    - pid=wait(&status);
    - If no parent waiting (did not invoke wait()) process is a zombie
    - If parent terminated without invoking wait process is an orphans
    - System designed to kill all orphans