# CSC 139 Chp 3 Continued

## 3.2 Process Scheduling

multiprogramming - have a process running at all times to maximize Cpu utilization

timesharing - switch a cpu core among processes frequently so that users can interact with each program while its running

process scheduler - selects an avilable process to execute on a core (One Cpu core one process)

degree of multiprogaming - Number of processes in memory currently

I/o bound & Cpu bound - i/o bound - spend more time doing i/o, cpu bound - uses more time doing computations

ready queue - proces is ready to execute on cpu core, stored as a linked List. Header points to first pcb in List & each pcb points to Next pcb in ready queue

wait queue - process that are waiting for an event to occur

cpu Scheduler - select the process that are in ready queue and allocate a cpu coe

swapping - a process can be "swaped out" from memory to disk, where its current state is saved then swapped back in later from disk to memory

interrupt - causes os to change a cpu core from current task & run a kernel routine

context switch - Switching a cpu core to another process. Do a save state then state rest

## 3.3 Operations on Process

pid or (Process Identification Number) - The way an Os identifies a process, int number, can be used as index to acess attributes of a process in Kernal

Process tree - parent process creates its own process and they create own process

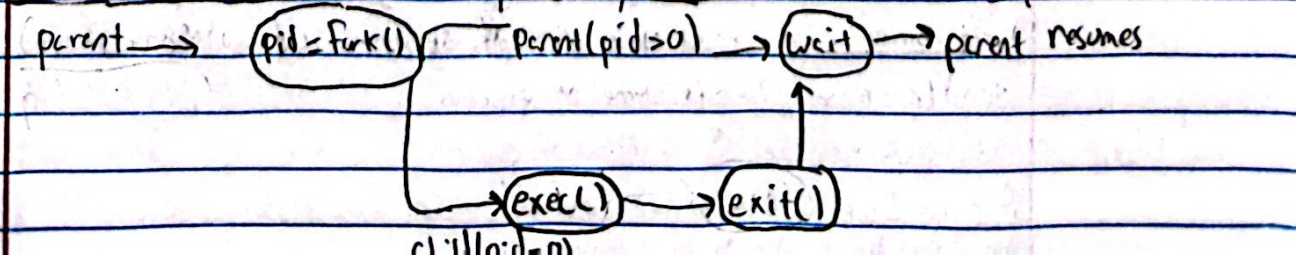
systemd in Linux pid of 1 is the root parent process, first process created on boot

SSh - Secure shell - Network protocol allows you to securley connect to a remote computer over an unsecured Network, Way to access remote machines, transfer files & execute commands remotely

ps -el - List of process on Linux, pstree - displays tree of all process

Parent / child process resource sharing - child process may ask Os for resources or the parent process could allocate its resources to child to avoid child overloading syst

fork() system call - creates a process, exec() - used to execute a process in Linux



CSC Chap 3 Continued

Create Process - similar to Fork() in Linux used to create process in Windows. CreateProcess requires Loading loading a program into the memory address space of child process, expects at least 10 parameters

Process Termination - done by using exit(), returns a status value to parent via wait() system call, all resources are deallocated & reclaimed by OS

Parent process can terminate child process, if uses to many resources, task assigned is no longer required, parent is exiting so a child can't exist without parent ← cascading termination.

wait() - accepts a parameter from child (exit status) also returns pid so parent knew which child terminated

Zombie - terminated process but parent hasn't called wait()

orphan - terminated process but parent doesn't call wait() and parent ends, so init (parent to orphans) calls wait() periodically, systemd can do same thing

3.4 Interprocess Communication

independent / cooperating process - independent does NOT share data with other process, cooperating if it can be affected of affect another process, pros of cooperating process: information Sharing, computation speedup - break a task into subtask to do faster, modularity

IPC interprocess communication done in

shared memory - process used shared memory to read & write data

message passing - message exchange between 2 cooperating process

3.5 IPC in shared memory

Shared memory in dept process have to agree to let each other to access each others memory. They have to insure they are not writing to same location at same time

3.6 IPC in Message Passing Systems

message passing more in dept - allow processes to communicate & synchronize actions without sharing memory address, useful in distributed environments

direct communication - must name name the recipient or sender

send(p, message) p is name of process

receive(q, message) Q is Name of process

A coms Link is established between pair of processes that want to communicate

# Chap 3 Continued

Link is between only 2 processes, & only 1 link between pair of process
Asymmetry direct communication - only sender names recipient
Send(Lebron, message) receive(id, message) id holds name of process it communicated with
indirect Communications - messages are sent to or recieved from a mail box or port
mailboxes - each mailbox has unique id, 2 process can communicate through shared inbox
send(A, message) recieve(A, messag) A = mailbox name
Process Mailbox ownership - owner - can only recieve messages to mailbox, user can only send messages to mailbox
OS owned mailbox - creates mailbox, send & recieve messages, delete mailbox
Message Passing Blocking / Non blocking
send Blocking / synchronus - cant send until message is recieved by other mailbox
send Nonblocking / asynchronus - sends message goes about its bidness
Blocking / synchronus recieve - reciever blocks till message is ready
Nonblocking / asynchronus recieve - reciever gets a valid message of Null "empty message"
rendezvous - when send() & recieve are blocking
temp queue - For message holding, Zero capacity - Sender blocks till message recieved
Bounded capacity - length N, can store N messages if full sender blocks till space
Unbounded Capacity - queue length is potentially infinite
posix - portable Operating System interface
Mach Message Passing - task ≈ process mailboxes = ports mach is FIFO
message passing in Windows - ALPC Advanced Local Procedure call - used for Communications between 2 processes on same machine
pipe - allows 2 process to communicate, 4 pipe cons, is it bidirectional, if its 2 way communication can data travel one way at a time or 2 way at a time, does a relationship need to exist (parent child), can the pipe communicate over a network or same machine only
Ordinary Pipe - producer writes to one end of pipe, Consumer reads from other end only one way communication, A parent can create a pipe to communicate with child windows are called anonymous pipes
Named Pipe - bidirectional & no parent/child relationship, exist After communication process have ended. They are FIFO system, but only half duplex, data can only travel one way at time, on same machine

Chapter 3 Continued

3.8

Communication in Client server Systems

Socket - end point for communication. 2 process communicating over a Network uses 2 sockets, socket is identified by ipAddress + port number

ssh server port 22, FTP server port 21, web or HTTP port 80

Well Know ports - All ports below 1024, used for standard services

Socket doesn't use well known port so a port > 1024

TCP reliable

UDP No

loopback address to refer to self 127.0.0.1

Marshalling - converting data or data structures to a format that can be easily stored later

Service - runs in background while executing long running operations or performing work for remote process