CSSE2310 Cheat Sheet

By Jenna Macdonald :^)

PAGING STEPS: Addr = Phys/V
Phys->V=F->P, V->Phys=P->F

1: Pg# = Addr/PgSize
2: Offset = Addr%PgSize
= (A/PgSize) - floor(A/PgSize)
* PgSize = Offset
3: Use PgTable to conv to page/frame (denote x)
4: Ans = x * PgSize + Offset

TLB = Translation Lookaside Buffer. Every frame->phys memory access is stored; not tables just frames, no page acces on same page if in the TLB (0Pg access) ∴

Memory Accesses

2 Level PgTable means if not in TLB, 2 mem access per page. All access to physical addr is +1, so if in TLB 1 **Mem** access, 3 otherwise (2 if single lvl)

PgTable Size

PgSize -> 2^x Bytes
x is offset length
#Pages = Addr/PgSize = 2^Z
Addr=Length of Addr (2^32 for 32bit).

Size of 1lvl

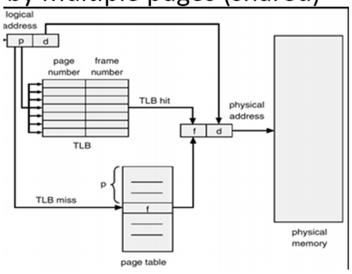
PgSize/EntrySize = 2^X
X = entries per frame
2^Z/2^X = #Frames = 2^F
Size=#Frames*PgSize

Size of 2lvl

AddrSize (32) - offset length = Y bits. Same process for #entries per frame.
Bottom lvl (2nd lvl) stores #entries per frame (X).
Top lvl has Y - X entries.
If result < X, only one top page, else numerous.
Size = PgSize * Toplvl# + PgSize * UsedBottomLvl
#Used=EndPgOfAddr-StartPg

Toplvl Must be sequential!

Seg fault on invalid/readOnly Page (SIGSEV(11))
Frames can be pointed at by multiple pages (shared)



File Systems

Files have metadata stored by inode; filename, type, location, size, etc
File systems rely on Acyclic graphs to function = no recursive links in system
System prevents this!

Directories

Used to find inodes in system
Each inode is a unique # in file system, directory maps names to inode #'s. Dir stored in file, but treated diferently to other files. Each dir has its own name space so two dirs can have the same name but a single dir can't have duped name mappings.

RWX on dir is diff; R=See inside
W=Can you add/rm files to this dir?
X=Can you access things in this dir?
Need syscalls to manipulate inodes; kernel adds/rms inodes when you ask

Linking

Use ln (-s for soft links) source linkname
Hard links mean file has two parents in acyclic graph; if one changes it, the other sees the changes. File has a reference count which is incremented for every hardlink that points to it; decremented when a hardlink is removed. If count = 0, file is marked as free memory by kernel. Hard links = share inode #.
Can tell how many dirs in a dir by using ls -id /dirpath (-2 for . and parent).

Soft links (-s) is a mapping from name -> name instead of name->inode (hardlink), Not a proper link, acts like a window to the actual file location
Remove link; nothing happens. Remove file end and soft link is now dangling ptr. Replacing the removed file with a new file with same name causes soft link left dangling to point to new file automatically. Soft links = names only.

Mounting

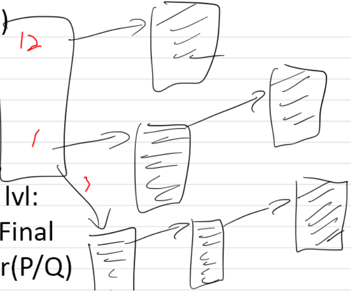
Put new filesystem in bottom of filesystem tree.
Main sys has priority, but lets you chain systems.

File perms (user = owner)

wrxwrxwrx = user, group, others, your group overrides the others (user has highest priority)

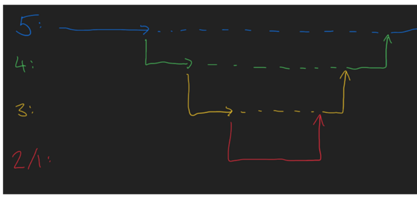
UNIX uses X/Y/Z system (picture)

X direct ptrs to data,Y indirect to blk of ptrs to data,Z dubs indirect ptr to blk of ptrs to blks of ptrs to data. Nums vary i.e 10/2/3, 12/1/1
Q = blkSize/ptrSize. MaxSize = (X*blkSize) + (Y*(Q)*blkSize)+(Z*Q^2*blkSize)
FileSize=ptr+ blkSize (varies) Reads: 1 per lvl unless cached, +1 for blk itself. Byte A in blk B, B=floor(A/blkSize). Dub lvl: B = X + Y*Q + P. Solve P. Final loc in last lvl is ptr # floor(P/Q)



Networking

Layer 1: Physical: (Wire signals, voltages, signals etc).
Layer 2: (Data) Link: MAC, wifi, ethernet, envelopes data into packets to be sent down physical
Node=Computer/Router etc, can have many interfaces to one node no intermediary between L1.
Layer 3: Network: IP etc. Receive packet, find adjacent node, send to them (if it's not addressed to you). If it is, give it to L4. Check all L2 for node to send to, pick one with ip in the right dir of the ip hierarchy. Envelopes data with IP info.
Layer 4: Transport. TCP/UDP. Introduces sockets/ports, send packets in order, adds reliability. Enables talking to specific processes.
Layer 5: Application Builds on L4. Putty, SSH, etc "Care about data only"
All layers build on last.
Envelopes add hdrs/ftrs to packets, layers add/rm these to their protocol.
IP HDR: Source/Dest Addrs, Time2Live (maxHops), size
Subnets: IP tree/hierarchy provides subnets. N bits common between IP's in subnet; 32-n "host bits" ID uniq endpt. 2^(32-n) - 2 max uniq endpts on subnet (-2 for bcAddr/nwAddr).
CIDR notation: A.B.C.D/X
A.B.C.D = nwAddr, X = n common bits between IPs "Classless Interdomain Routing". **Nmask:** bin# with X leading 1's, rest 0 i.e 0b1111000..... **bcAddr:** uniq IP to bc to all endpts. = nwAdrr up to X, then all 1's. **nwAddr:** networkAddr. & any nw IP with nmask to get. If any of nwAddr/bcAddr collide with an existing IP in subnet, -1 from X and recalc.
Math is take all IP's in nw, find where they differ (A||B||C||D) conv diff section to binary and draw line where they diff; this gives you X (Each sect = 8 + #bits common in differing sect).
NATTING (priv IP->PubIp) causes change in source IP of packet and dest IP (if endpt is private).
Each node hope is a new MAC addr == +1 MAC count. Ethernet frame is 0 IP's as ethernet is Layer 2!
No diff in TCP/UDP same layer same stuff; ignore TCP sending data to and from

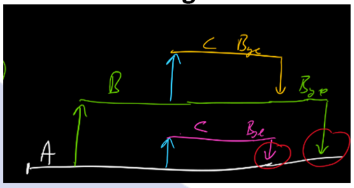


DNS is layer 5: It uses L4 to search for new addr to conv.

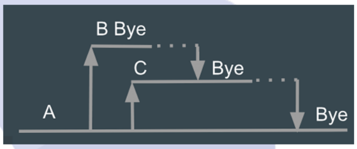
Forking Diagram Examples

Remember: Children on fork can only join to own parent
If proces has 2+ kids, order those kids join in is irrelevant

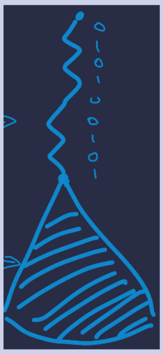
Valid Fork Diagram:



Invalid:



Forked children **can only join their direct parents; threads can join any other thread**, so almost all thread diags valid



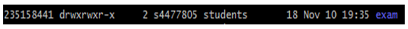
NExample:
X: 10.1
Y: 10.2
Z: 10.4
diff@B
dec->bin
X: 00000001
Y: 00000010
Z: 00000100
First 5 bits common so 8+5=13
CIDR /13
Nmask: 255.248...
NwAddr: 10.1&NM
BC: NwAddr first 13 bit then all 1 (in bin)

2^13 = 8Kb

Smaller blkSize improve *internal fragmentation* (fileSys)

Bash

ls -ali output: inode# (-i only) filePerms #hardLinks owner group fileSize timeInfo fileName

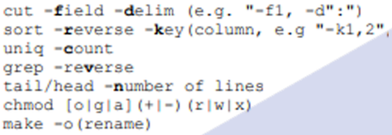


```
235158441 drwxrwxr-x    2 s4477805 students      18 Nov 10 19:35 exam
```

For Loops/Vars: Vars specified by $var var=<blah>, echo $var. $$ = pid of shell, $0-$9 = cmd args, $* = argv, $# = argc, $? = exit status, $! = pid of bg command.

for name in [list]; do if [cond] then elif [cond2] then else fi; done; cond: eq = = g/l/e = >=/<=, g/l/t = >/<, ne = !=.

Commands/Examples



```
cut -field -delim (e.g. "-f1, -d":")
sort -reverse -key(column, e.g "-k1,2",
uniq -count
grep -reverse
tail/head -number of lines
chmod [o|g|a](+|-)(r|w|x)
make -o(rename)
```

List uniq users running fitz in reverse alpha order:**ps -eF | grep “fitz” | tr -s ‘ ’ | cut -d ‘ ’ -f 1,11- | sort -r | uniq.** English: Print every process, search for fitz, condense spaces to allow easier cutting for columns 1 (sID) and 11 (command), sort in reverse order, remove duplicates.

kill -9 PID = kill non-child, kill in C sends signals

echo $PATH | cut -d ‘;‘ -f 1: Output first dir to be searched for commands.

PATH=$PATH:X = add X to PATH var

for file in *.pdf   (Copy all pdfs to new place
do   Adding old_ to front)
cp $file /dest/old_$file
done

Modify perms to deny group anything

chmod u=wrx,g=---,o=wrx <file>

ls -d ?????* | grep -v ^d = all filenames >=5

SVN

svn rm <file> = Remove from repository

svn commit = Update repository with change

svn diff <f1> <f2> = See diffs from repo copy and local copy

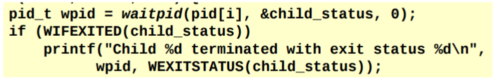
svn checkout = Get a new local copy

svn status = Check repo status

svn update = Update repo to latest version

Add svn to mv, cp etc to use in repo; svn add adds a file to the repository. Always commit after making a change for it to take effect.

Misc &> redir both. pgrep looks at proccesses 2> redirs stderr, > stdout, >> apend stdout kill(pid) kills a process. Get this from ps -u Killing PPID (parent pid) kills kids under it. Process 1 adopts zombies if parents die. Zombie = process which has been killed, but keeps using system resources. Needs to be repaed with wait by parent as can’t kill what is already dead.



```
pid_t wpid = waitpid(pid[i], &child_status, 0);
if (WIFEXITED(child_status))
    printf("Child %d terminated with exit status %d\n",
           wpid, WEXITSTATUS(child_status));
```

int child_status holds onto exit status of child, use WIFEXITED to see if exited normally, WEXITSTATUS to get specific status out.

C Thread Format

Every thread Q has the same pattern, use below (Simplest memory freeing chosen): Be aware; types of things may change, like the thing being added/found

```
typedef struct Box {
    pthread_mutex_t* lock; // No races
    int* sum; // Ptr to main threads sum
    int numLoops; // They always loop
    int* list; // Master list, ptr varies
}Box; // too much for array

/* Used by threads to do their small bit of
work; unpack data, do math, return NULL*/
void* thread_func(void* args) {
  Box* data = (Box*) args;
  int sum = 0; // local var

  for (int i = 0; i < data->numLoops; i++) {
    sum += data->list[index];
  } // May ask to use a func instead, or diff
  // Operation; replace this loop op with that

// Lock the global sum to avoid collisions
  pthread_mutex_lock(data->lock);
*(data->sum) += sum;
  pthread_mutex_unlock(data->lock);

  free(data); // Thread box is now done
  return NULL;
}

void question_funct(<whatever it takes>) {
  // Assume we’re just adding numbers
  int perThread = numElements/numThreads;
  int sum = 0, index = 0; //list is an arg

  pthread_mutex_t masterLock;
  pthread_mutex_init(&masterLock, NULL);
  // Here we make an array and init to 0 of
  // Thread ID’s for later joining
  pthread_t* threadIds = (pthread_t*)
  calloc(numThreads, sizeof(pthread_t));

  for (int i = 0; i < numThreads; i++) {
    // Make a box in mem; thread frees
    Box* data = (Box*) calloc(1, sizeof(Box));
    // How many pieces do they get?
    if (i + 1 == numThreads) { // Last itr
      data->numLoops = <rest of arr>;
      // (numEle - i * perThread);
    } else { // Above should still give interval
      data->numLoops = perThread;
    }
    // Adjust list they see
    data->list = (list + i * perThread);
    data->sum = &sum; // Master sum
    data->lock = &masterLock;
    pthread_create(&(threadIds[i]), NULL
        thread_func, (void*) data);
  }
->
// Finally, join them back
for (int i = 0; i < numThreads; i++) {
  pthread_join(threadIds[i], NULL);
}
pthread_mutex_destroy(&masterLock);
free(threadIds); // This is how every thread
// Question will go down 98% confidence
return sum;
} // Call this in a main and done!
```

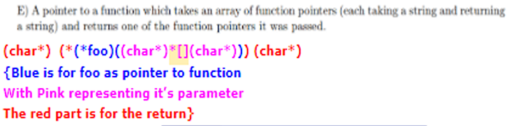
C Stuff

(unsigned) int, long, long long, double, floa char, bool. Fork: pid_t (pids). Pipe takes an int[2] as arguments, 0 = read, 1 = write. FILE* var = fdopen(fd, “mode”); // r | | w Fork returns a pid_t of child. Close ends of pipe you aren’t using with close(fd). dup2(fd you want to use, fd you replace) i.e dup2(pipe1[WRITE], STDOUT_FILENO); // Replaces stdout of process with read end

Example:

```
int pipe[2]; // Will be executing program
pipe(pipe); // Called name, with args
  pid_t pid; // Pid of child stored here
  (if (pid = fork()) == 0) { // Args are sep
    dup2(pipe[1], STDOUT_FILENO);
    dup2(pipe[2], STDIN_FILENO);
    execlp(name, name, args, (char*)NULL);
  } // Can waitpid now on pid later
```

Networking: Socket makes a phone, **Bind** gives it a number (port), **Listen** lets it be called by others, **Accept** accepts a call (a connection). Use fdopen on socket to talk to connection with fprintf etc. Disconnect by closing the socket.



E) A pointer to a function which takes an array of function pointers (each taking a string and returning a string) and returns one of the function pointers it was passed.

(char*) (*(*foo)((char*)*[](char*))) (char*)

{Blue is for foo as pointer to function

With Pink representing it’s parameter

The red part is for the return}

strncmp(str1, str2, n bytes); // Comp strs fgets(char* buff, int n, FILE* stream); // n bytes read into buff up to /n or EOF // Return NULL on eof, so if NULL exit loop sprintf(char* dest, “str%d%c”, args); strcpy(new, old); Malloc first, sizeof(old)+1 isdigit(char c), isalpha(char c), strtol needs ptr; strltol(str, &ptr, 10), ptr pts to first char after num conversion stopped. strtol strips newline (as does sscanf(str, “format”, &var) fopen == filenames, fdopen == fd’s, NULL on failure. char** arr = (char**) realloc(arr, newSize); // Use this to grow stuff dup(fd) copies to get two streams same fd execvp lets you give an array of args instead of individualy giving them. Signals affect an entire process, so threads all get signal Two pipes made to talk and receive data between processes. FD_CLOEXEC to close on exec. Close pipes after dup2. Good luck!

Pointers (Quickly): int a = 5, b = 65; int* x = &a; // POINTS@a *x= 6; // De-ptr a & change val; a == 6. b = (char) b; // b = ‘A’ because of typecasting.