Networks:

① Get Subnet Mask / CIDR ← 32 - 26 = Host Bits

→ 10.1.5.0/26 = 255.255.255.192

② N.A = IP AND Subnet Mask.

→ Check N.A. Does not conflict.

③ B.A. = OR !(Subnet Mask).

→ Check B.A. Does not conflict.

| UNIX Filesystem definitions | | | | | | |
|---|---|---|---|---|---|---|
| index-node | for example, 'etc/services' on moss has index number 67109041 on its file system | | | | | |
| Dir filesystem | Each directory contains 14 block pointers. First 12 point to data blocks. | | | | | |
| 13th block | Indirection block. Points to block of 1024 pointers to 1024 more data blocks | | | | | |
| 14th block | Two indirections. Points to block of pointers to indirect blocks | | | | | |
| **Maximum Possible File Size.** | | | | | | |
| 1. Direct Pointer Size. | = # of Direct Pointers * Block Size. | | | | | |
| 2. Pointers/Block | = Block Size / Block Pointer Size. | | | | | |
| 3. Total Data | = Direct Pointer Size + (# of Indirect Pointers * (Pointers * Block Size)) | | | | | |
| | + (# of Double Indirect Pointers * (Pointers * Pointers * Block Size)) + ... | | | | | |
| Block accesses | | | | | | |
| Direct Block contains pointers to [0, # of direct pointers]. | | | | | Access = 1 | |
| Indirect Block contains pointers to [# of direct pointers + 1, Pointers/Block - 1]. | | | | | Access = 2 | |
| Second Indirect Block contains pointers to [Pointers/Block + 1, Pointers/Block^2 - 1]. | | | | | Access = 3 | |
| **inode benefits** | | | | | | |
| Smaller file blocks are better because they allow the tree to be wider, rather than deeper. Theoeritically granting faster access | | | | | | |
| Inodes are able to store metadata for a file | | | | | | |
| Inodes allow file names to be modified while open, prevents applications from 'hijacking' a file. | | | | | | |
| **inode disadvantages** | | | | | | |
| inode information is kept separately from data, access of data often requires a long seek when file is initially accessed | | | | | | |
| inodes of files in a common directory not kept together, leading to low performance when searching directories | | | | | | |
| data blocks of a file are not stored together, leading to poor performance when accessing files sequentially. | | | | | | |

**t protocol suite**

**pplication layer**
BGP · DHCP · DNS · FTP · HTTP · IMAP · LDAP · MGCP · NNTP · NTP · POP · ONC/RPC · RTP · RTSP · RIP · SIP · SMTP · SNMP · **SSH** · Telnet · TLS/SSL · XMPP · *more...*

**Transport layer**
TCP · UDP · DCCP · SCTP · RSVP · *more...*

**Internet layer**
IP (IPv4 · IPv6) · ICMP · ICMPv6 · ECN · IGMP · IPsec · *more...*

**Link layer**
ARP · NDP · OSPF · Tunnels (L2TP) · PPP · MAC (Ethernet · DSL · ISDN · FDDI) · *more...*

8KB pages (8192Bytes) Offset = Virtual Address % Page size

two level page table.

| Page | Frame |
|---|---|
| 0 | 21 |
| ... | ... |
| 23 | 99 |
| 24 | 100 |
| 25 | 19 |
| 26 | 102 |
| 27 | 100 |
| 28 | 101 |

Virtual Page # = virtual address / Page size

← Find Physical Page, index via virtual address.

Physical Address = (Physical Page × Page Size) + Offset.

Virtual Page # = 197493 / 8192 = 24 Offset: 197493 % 8192 = 885

Physical Address = (100 × 8192) + 885 = 820 085

| ls -l columns | Links - 2 = Subdirectories | | | | | |
|---|---|---|---|---|---|---|
| Permissions | Number of Links | Owner | Group | Size (bytes) | Datetime | File/Directory |
| drwxrwxr-x | 3 | s4354663 | students | 3 | Oct 3 14:43 | CSSE2310Ass4 |
| **ls -lih columns** | | | | | | |
| Index Node | FileType/Permissions | Number of Link | Owner | Group | Size (KB/Bytes) | Datetime | File/Directory |
| 10134095 | drwxrwxr-x | 3 | s4354663 | students | 376K | Oct 3 14:43 | ass4_spec.pdf |
| **other ls flags** | | | | | | |
| -h | -i | -a | -d | | | |
| human readable | print index node | all. | list directories, not their content | | | |
| **ln <existing file> <new name>** | If the new name file already exists, the call will fail | | | | | |
| Keep an eye on index node values. Files with the same inode will be hard linked to the same place. | | | | | | |
| fjord -> python3 | means that python3 is somewhere else, and fjord is linked to it. | | | | | |
| Removing links to a file (i.e. removing fjord) does not affect the file. | | | | | | |
| Removing a file that has links will break the links (i.e. removing python3). | | | | | | |
| File Types: - file, l link file, d directory, p pipe, c character special device, b block special device | | | | | | |
| To exec subprograms in links, (i.e. execute python3.sh from fjord) we can fjord/python3.sh IFF python3/python3.sh exists | | | | | | |
| Hard links can keep the file that they reference from being deleted | | | | | | |
| Dangling pointer problem can occur with mismanaged symbolic links (i.e. A --> file1. B --> A. rm A. B --> nothing). | | | | | | |
| **permissions format** -rwxrwxrwx | Read write & execute for | Owner, group and others. | | | | |
| chmod works on octal values. | x = 1, w = 2, r = 3 | chmod u+rwx | chmod g-rw | chmod 664 file --> | -rw-rw-r-- | |
| **hard links** | | | | | | |
| Every directory is a hard link | | | | | | |
| Original can be moved or deleted without breaking other hard links to the same inode | | | | | | |
| Only possible within the same file system | | | | | | |
| Permissions must be the same as those on the original. (Permisisons are stored in the inode). | | | | | | |
| Can only be made to files | | | | | | |
| **soft links** | | | | | | |
| Simply records that point to another file path. | | | | | | |
| Will break if the original is moved or deleted. | | | | | | |
| Can point to a file in a different file system and can point to a directory | | | | | | |

TLB - Start with V.A. & Page

1. Can we use cache?

1.1. If so, + 1 Read, END.

2. Follow L1, L2 & P.A + 3 Reads

3. Cache Page x → Frame y.

```c
//2016 A
int a = 5;
int b = 13;
int c;
c = (b^a^b)|b;  // ((XOR b a) XOR b) OR B
printf("%d", c);
// a:   0101
// b:   1101
// XOR  1000
// b:   1101
// XOR  0101
// OR   1101
// FIN  1101 --> 13 = c

//2016 B
int a = 26;
int b = 9;
int c = 6;
int d = a & b | c; // (26 AND 9) OR 6
printf("%d", d);
// a:   0001 1010
// b:   0000 1001
// AND  0000 1000
// OR c:0000 0110
// FIN  0000 1110 --> 14 = d

//2016 C
int a = 7;
int b = 4;
int c = a+b/a*2.5;  //int * float = int
printf("%d", c);
// c = 7 + 4/7 * 2.5
// c = 7 + 0 * 2.5
// c = 7

//2016 D
int a=1, b=10, c=5;
if ((a--) && (b=5)) { // assignments in condi
    c--;
}
printf("%d %d %d", a, b, c);
// a-- will return 1 before decrementing
// b=5 will always be true
// thus c will be deducted.
// a = 0, b = 5, c = 4

//2016 E
int a = 17;
int b = 4;
printf("%d", a-b/a+b);
// a - b/a + a
// 17 - 4/17 + 4 = 21

//2016 F
float a = 3.0;
int b = 2;
a /= a+b;
printf("%f", a);
// a /= a+b
// a+b = 5
// a /= 5 --> 3.0 / 5
// a = 0.600000

//2016 G
int a = 17%4;
int b = 1;
switch (a) {
    case 0: b=b+2; break;
    case 1: a=2; break;
    case 2: b=b+5;
    case 3: a=a+5;
}
printf("%d %d", a, b);
// a = 1, b = 1
```

```c
// case 1 sets a to 2.
// a = 2, b = 1

//2016 H
int i = 5;
int t = 0;
for (int i = 0; i < 9; ++i) { // new i
    t = t+i;
}
printf("%d %d", i, t);
// enters for loop 8 times.
// each time inside the loop i is:
// 1 2 3 4  5  6  7  8      i
// 1 3 6 10 15 21 28 36     t
// note that i still exists in outer scope
// 5 36

//2016 I
int i = 5;
int t = 0;
for (i = 0; i < 10; ++i) {  // old i
    t = t+i;
}
printf("%d %d", i, t);
// enters the for loop 9 times.
// each time inside the loop i is:
// 1 2 3 4  5  6  7  8  9    i
// 1 3 6 10 15 21 28 36 45   t
// 10 45

//2016 J
char* w = "fred";
char* x = "hw";
char* h = &(x[1]); // pointer to w
w = w + h[1] + 1;
printf("%s", w);
//  w = w (pointer to w[0])
//  h[1] = end of h = '\0'
//  so w = w + 1.
//  now w points to "red".

//2015 A
int a = 13;
int b = 20;
b = b + a * 2;
printf("%d", b);
// a * 2 comes first
// 20 + 26 = 46

//2015 B
int a = 15;
int b = 9;
b += a - b;
printf("%d %d", a, b);
// 9 += 6
// 15 15

//2015 C
int a = 7;
int b = 4;
switch (a & b) {
    case 1: a++; break;
    case 2: b=2; break;
    case 3: a--; break;
    case 4: a==3;
    case 5: ++a; break;
    default: b--;
}
printf("%d %d", a, b);
// a & b =  0111
//          0100
//          0100 -> 4
// In case 4, nothing happens
// In case 5 a++
```

```c
// 8 4 FIN

//2015 D
int a = 0;
for (int j=0; j<12; ++j) {
    if (j % 3) {
        continue;
    }
    a++;
}
printf("%d %d", a, j);
// Won't compile.  printf
// cannot access the j variable.

//2015 E
int a = 17;
printf("%d\n", a++ - 3);
// Postincrement.  Will inc
// after printf is called.
// so 14

//2015 F
int a[] = {1, 4, 9, 16, 25};
int* b = &a[1];
printf("%d", b[1]);
// B starts at 4.
// index 1 is 9.
// 9

//2015 G
int a = 7;
while (a>0) {
    a--;
    if ((a%2==1) || (--a)) {
        continue;
    }
    a--;
}
printf("%d", a);
// Enter loop at:
// 7 5 3 1
// Modificaton:
// 5 3 1 -1
// FIN -1

//2015 H
int a = 5;
int b = 3;
float c=a/b;
printf("%f", c);
// int / int = int.
// 1.000000

//2015 I
int a = 7;
int b = 2;
float c = 1.5+a/b*1.5;
printf("%f\n", c);
// c = 1.5 + 7/2 * 1.5
// c = 1.5 + 3 * 1.5
// c = 6.000000

//2015 J
int a = 5;
int b = 3;
int c = 12;
c = c^(b&c);
printf("%d", c);
// B 0011
// C 1100
// & 0000
// ^ 1100
//F: 1100
//      12
```

Memory Required for a Page Table

memory = number of page tables * page size

Example

32-bit virtual addresses

4kB pages

PT entries are 4 bytes

2 Level PT

12280 bytes starting at 1024

30719 bytes starting at 125825024

Entries per page table = page size / PT entry size = 4096 / 4 = 1024

Starting Page = virtual address / page size = 1024 / 4096 = 0.

Ending Page = Starting at x + bytes = 123904 --> 123904 / Page Size = 30

Where is this on the L2 Page Table?

Starting Page = Starting at x / Page Size = 1024 / 4096 = 0 --> Starts at page 0

Ending Page = (Starting at x + x Bytes) / Page Size = (1028 + 122880) / 4096 = 30 --> Ends at page 30

Pages 0 to 30 will lie in the first L2 page table. (L2 page table entry contains 1024 pages)

All up?

Always need your root (L1) table. How many L2's are used?

Total Size = Page Size + (L2 tables * Page Size)

| Address block (CIDR) | Range | Number of addresses | Scope | Purpose |
|---|---|---|---|---|
| 0.0.0.0/8 | 0.0.0.0 – 0.255.255.255 | 16,777,216 | Software | Used for broadcast messages to the current ("this")[1] |
| 10.0.0.0/8 | 10.0.0.0 – 10.255.255.255 | 16,777,216 | Private network | Used for local communications within a private network[2] |
| 100.64.0.0/10 | 100.64.0.0 – 100.127.255.255 | 4,194,304 | Private network | Used for communications between a service provider and its subscribers when using a carrier-grade NAT[3] |
| 127.0.0.0/8 | 127.0.0.0 – 127.255.255.255 | 16,777,216 | Host | Used for loopback addresses to the local host[4] |
| 169.254.0.0/16 | 169.254.0.0 – 169.254.255.255 | 65,536 | Subnet | Used for link-local addresses between two hosts on a single link when no IP address is otherwise specified, such as would have normally been retrieved from a DHCP server[5] |
| 172.16.0.0/12 | 172.16.0.0 – 172.31.255.255 | 1,048,576 | Private network | Used for local communications within a private network[2] |
| 192.0.0.0/24 | 192.0.0.0 – 192.0.0.255 | 256 | Private network | Used for the IANA IPv4 Special Purpose Address Registry[6] |
| 192.0.2.0/24 | 192.0.2.0 – 192.0.2.255 | 256 | Documentation | Assigned as "TEST-NET-1" for use in documentation and examples. It should not be used publicly.[7] |
| 192.88.99.0/24 | 192.88.99.0 – 192.88.99.255 | 256 | Internet | Used by 6to4 anycast relays (deprecated)[8] |
| 192.168.0.0/16 | 192.168.0.0 – 192.168.255.255 | 65,536 | Private network | Used for local communications within a private network[2] |

**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 exan

**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.

```
char** var[];
        Declare var as array of pointer to a pointer to a characters.
int* a, var;
        Var is an integer, since a is the pointer to an int.
int var(int);
        Var is a function which takes 1 integer and returns an integer.
int (*var)(int);
        Var is a pointer to a function which takes 1 integer and returns an integer.
void (*var)();
        Var is a pointer to a function which takes anything and returns nothing
double *(*var)(double);
        Var is a pointer to a function which takes 1 double and returns a pointer to a
        double
long (*var)(int, int);
        Var is a pointer to a function which takes two integer arguments and returns a long
        integer.
void (*var)(void*, int);
        Var is a pointer to a function which takes 2 arguments, 1 is an integer and the
        other is pointer to anything or a generic pointer and returns nothing
int *(*(*var)(int*))(void);
        Var is a pointer to a function which takes 1 argument which a pointer to an integer
        and it returns a pointer to a function which takes no arguments and returns a
        pointer to an integer.
void *(*var)(void *(*)(void*), void*(*)(void*),void*);
        Var is a pointer to a function which returns a generic pointer which takes three
        arguments, the first and the second are both a pointer to a function which takes a
        generic pointer and returns a generic pointer and the third one is a generic
        pointer.
int (*(*var[3])())(void (*)());
        Var is an array of size 3 which contains function pointers which takes anything and
        returns a pointer to a function which 1 argument which is a pointer to a function
        which takes anything and returns nothing, and returns an integer.
char *(*var)(void*(*)(int), int(*)(void))
        Var is a pointer to a function which returns a pointer to a character and takes two
        arguments. The first is a pointer to a function which takes an integer arg and
        returns a generic pointer. The second is a pointer which takes no arguments and
        returns an integer
int (*var) ();
        Var is a pointer to a function which returns an integer and takes anything.
int* (*var)(void)
        Var is a pointer to a function which returns a pointer to an integer and takes no
        arguments.
int* (*var)(int, int(*)(int))
        Var is a pointer to a function which returns a pointer to an integer which takes
        two arguments. The first is an integer, the second is a pointer to a function which
        returns an integer and takes one argument which is an integer.
typedef int (*funct)(int, int);
funct (*var)(funct, funct);
        Var is a pointer to a function, this function should return a function pointer to a
        function which takes two ints and returns an int. var should take two parameters
        each of which are the same type as its return value. (Hint use a typedef).
```

```
char* substr_format(const char* big, const char* srch, int bigIndex) {
    char* format = calloc(80, sizeof(char));
    int srchlen = strlen(srch);
    char* tempBig = big + bigIndex;
    strncat(format, tempBig, srchlen);
    strcat(format, "-");
    strcat(format, srch);
    return format;
}
```

```
char* substr_format(const char* big, const char* srch, int bigIndex) {
    char* format = calloc(80, sizeof(char));
    int srchlen = strlen(srch);
    char* tempBig = big + bigIndex;
    strncat(format, tempBig, srchlen);
    strcat(format, "-");
    strcat(format, srch);
    return format;
}
```

```
int countsubstr(const char* big, const char* srch, int tCount) {
    int srchlen = strlen(srch);
    int i = 0;
    void* sum;
    while (big[i] != '\0') {
        for (int i = 0; i < tCount; i++) {
            // Format the data into something you can communicate to a thread.
            char* subStr = substr_format(big, srch, i);
            pthread_t tid;
            pthread_create(&tid, NULL, substr_examiner, subStr);
            // Wait for the thread to return.
            pthread_join(tid, &sum);
        }
        i += srchlen;
    }
    fprintf(stdout, "%d\n", *(int*) sum);
    return sum;
}
```

```
int main() {
    // First, we're going to open a file
    int file = open("myfile.txt", O_APPEND | O_WRONLY);
    if(file < 0) {
        return 1;
    }
    // Now we redirect standard output to the file using dup2
    if(dup2(file,1) < 0) {
        return 1;
    }
    // Now stdout has been redirected, we can write to
    // the file
    fprintf(stdout, "This goes into the file\n");
    fflush(stdout);
    close(file);
    return 0;
}
```

```
typedef Struct {
    int* count;
    pthread_mutex_t countMutex;

    int start;
    int end;
    int* values;
} ThreadInfo;

void threadSum (void* arg) {
    ThreadInfo* tI = (ThreadInfo*) arg;
    free(arg);
    for (int i = tI->start; i < tI->end; i++) {
        pthread_mutex_lock(&tI->countMutex);
        *tI->count += values[i];
        pthread_mutex_unlock(&tI->countMutex);
    }
}

sarray(int n, int t, int* values) {
    int start = 0;
    int count = 0;
    pthread_t threadIDs[t];
    pthread_mutex_t countMutex;
    pthread_mutex_init(&countMutex);
    int increment = n/t;
    for (int i = 0; i < t;  i++) {
        ThreadInfo* tI = malloc(sizeof(ThreadInfo));
        tI->start = start;
        tI->end  = start + increment;
        if (i == t - 1) {
            tI->end = n + 1;
        }
        tI->values = values;
        tI->countMutex = countMutex;
        tI->count = &count;
        pthread_create(&threadIDs[i], NULL, threadSum, tI);
        pthread_detatch(threadIDs[i]);
        start += increment
    }
    return count;
}
```

| Command | Example |
|---|---|
| find . -name testfile.txt | Find a file called testfile.txt in current and sub-directories. |
| find /home -name '*.jpg | Find all .jpg files in the /home and sub-directories. |
| find . -type f -empty | Find an empty file within the current directory. |
| find /home -user exampleuser -mtime 7 -iname ".db" | Find all .db files (ignoring text case) modified in the last 7 days by a user named exampleuser. |
| find -type f \| grep -Ez '[^/]{9}$' | Find all filenames in the cwd which are longer than 8 characters |
| head -5 elephant \| tail -1 | Show the 5th line of the file elephant |
| tail -2 elephant \| head -1 | Show the last 2nd last line of elephant |
| awk -F',' '{print $1}' ferret | First column of the file ferret (comma separated) |
| compgen -c | Will list all the commands you could run |
| echo $PATH | Directories which will be searched for commands to be run |
| PATH=$PATH:~/srd/bin | /srd/bin to the list of directories to search for commands |
| PATH=~/srd/bin:$PATH | /srd/bin will be added at the first directory to search for commands |
| ps -U $USER \| grep [b]ash | Show all instances of bash which you are running on the system |
| ps -C "vim" | Show all instances of vim on the system |
| awk -F',' '{print $1}' elephant \| grep 'mouse' \| wc -l | Count how many times 'mouse' appears in the first column of the file elephant |
| find $PWD -maxdepth 1 -type f -name '??????*' \| wc -l | Count how many times files in the current directory have names containing at least 5 characters |
| grep 'Clara > River' file1 | Output all lines of the file file1 which contains the text 'Clara > River' |
| grep 'dinosaur' file1 >> london | Find all the lines in the file file1 which contain the word dinosaur and store them in london. |
| cp prog.c new.c | Copies prog.c into new.c (if new.c exists, it is overwritten, if not, it is created). |
| rm A*.xyz | Remove files starting with A and ending with .xyz |
| grep '^goblin' test.txt | Will display all the lines starting with goblin in test.txt |
| grep 'goblin$' test.txt | Will display all the lines ending with goblin in test.txt |
| grep 'song\|river\|terrible' f1 f2 f3 | For files f1,f2,f3 show all lines from any of them which contain all the words "song", "river" and "terrible". |
| awk -F' ' '{ print $1, $3, $4 }' nums \| sort -k 3,3 | Split nums by a space delimiter into cols. Take the 1st, 2nd and 4th column and output sorted by the 4th col |
| mv file1 file2 | Rename file1 to file2. |
| cp ?????? /tmp | Copy all files with 6 letters in the name into tmp. |
| find -name 's?*d' -delete | Finds all files in the current directory starting with s and ending with d except sd, then deletes them |
| sort -k 2,2 -r data.txt | Reverse sort the data in data.txt by the second column of data |

| | | | | |
|---|---|---|---|---|
| pthread_create(pthread_t* thread, const pthread_attr_t* attr, void* (*start_routine) (void*), void* arg); | | | | |
| Example: pthread_create(&threadId, NULL, function_to_call, param_to_pass) | | | | |
| Note that the threadId doesn't really matter.  You can only pass one parameter to the function. | | | | |
| pthread_detach(pthread_t thread); | | | | |
| Example: pthread_detach(threadId) | | | | |
| When a thread is detached, it will execute in isolation, and when it is finished, it give resources back to the main thread | | | | |
| pthread_join(pthread_t threadId, void** retval); | | | | |
| Example: pthread_join(threadId, &status); | | | | |
| Join can be used to wait for a thread to return a value.  Just need to make sure that the 'status' is a void* and we pass the &status into the function | | | | |
| pid_t fork(); | On success, pid of 0 is given to child, pid of child returned to parent. | | | |
| int pipe(int pipefd[2]); | On success, pipefd[0] becomes the read end of the pipe, pipefd[1] becomes the write end. | | | |
| When combining forks and pipes.  The child must close the read/write end and the parent must close the opposite. | | | | |
| You can use the wait(NULL) call to wait for your children to terminate, to ensure no zombies are created. | | | | |
| int execl(char* program, char* program, char* file, void* shouldBeNull) | | | | |
| Example: execl("ls", "ls", "-lih", (char*) NULL); | | | | |
| char* fgets(char* str, int n, FILE* stream); | int sscanf(const char* str, const char* format, ... );  Returns successful scans. | | | |
| int select(int nfds, fd_set* readfds, fd_set* writefds, fd_set* exceptfds, struct timeval* timeout); | | | | |
| Example: fd_set rfds; struct timeval tv; tv.tv_sec = 5; | select(1, &rfds, NULL, NULL, &tv); | Returns the fd with input to read. | | |
| char* strcat(char* dest, const char* src); | char* strtok(char* str, const char* delim); | char* strstr(const char* haystack, const char* needle) | | |
| int strtol(const char* str, char** endptr, int base); | size_t strspn(const char* bigStr, const char* spanStr); | | | |

| | | |
|---|---|---|
| Forking in for loops | 2^n - 1 | where n is the amount of times you iterate through the for loop |
| Wait() system call explanation | | |
| Calls to wait = | Total processes (including main) / 2 | |
| Wait(int* param) | Param records the exit status of the child | |
| Zombies = | Calls to wait - 1 | |