CSSE2310 Computer Systems Principles and Programming

Question Tips

- Unused addr: $2^{\text{host bits}} - 2$ — number of addresses.
- Thread diagrams are always valid.
- socket, bind, listen, accept
- socket, connect
- LOOK AT PRE AND POST INCREMENT!!!!
- TRANSLATION LOOKASIDE BUFFER
- TCP: send message back **always**.
- TLB: Memory cache for pages and page to frame references.
- One level: if in the TLB 0, otherwise 1 page read
- Two level: if in the TLB 0, otherwise, 2 pages read
- Number of memory access: 1 access if in the TLB, 3 if two-level,2 if one-level

Unix

Examples (courtesy of Ainsley Nand on Attic)

- Select the first and third column delimited by ' ' and sort by the first
  cat file.txt | cut -d' ' -c1,3 | sort -k1
- Get only the 12th line of a given file
  cat file.txt | head -12 | tail -1
- Get each line with cat and dog in a given file
  cat file.txt | grep "cat" | grep "dog"
- Find each line which contains "CSSE1001" and does not contain the word "boring"
  cat file.txt | grep "CSSE1001" | grep -v "boring"
- Find the number of times "rowing" occurs in a given file
  cat ainsley.txt | grep -o "rowing" | wc -l
- For all files f1,f2,f3 show all lines containing "song", "river" and "terrible"
  cat f1 f2 f3 | grep song | grep river | grep terrible
- For all files g1,g2,g3 show all lines not containing "song", "river" and "awful"
  cat g1 g2 g3 | grep -v song | grep -v river | grep -v awful OR grep -ve grep -ve song -ve awful f1 f2 f3
- Find all lines in file1 containing the word "dinosaur" and store in file called london
  cat file1 | grep "dinosaur" > london OR grep dinosaur file1 > london
- Show all lines in file1 starting with W
  grep ^W file1
- Show all lines in file2 ending with S
  grep S$ file2
- Modify path
  export PATH = $PATH:newpath
- Show second last line of file
  cat file | tail -n 2 | head -n 1
- Show fifth line of file
  cat file | head -n 5 | tail -n 1
- Show the third line and later (hide the first 2 lines)
  cat file | tail -n +3
- Show all but the last 10 lines (hide the last 10 lines)
  cat file | head -n -10

Unix commands:

- grep print lines matching pattern
  - -v invert-match: displays lines not containing the string
  - -o only show part of line that matches pattern
  - -i case insensitive
  - -R search all directories
  - $ fish$, for example, looks for the word fish as the last word of a line
  - ^ fish^, for example, looks for the word fish as the first word of a line
  - . any character in regex except newlines
  - * 0 or more of previous expression in regex
- gcc create executable
  - -o cretaes an executable file from a c file, the order needs to be: `gcc -o <executable-name> <name-of-file>`
  - -c compiles a c file and creates an .o file of the same name
  - -g include debugging symbols
- ls list directory contents
  - -l use a long listing format
  - -a shows all hidden directories, do not ignore entries starting with .
  - -d list directories themselves not their content
  - -i print the index number of each file
- ps process status
  - -e show all processes (-> to see every process on the system using standard syntax)
  - -u user's processes, if wishing to specify a user, specify -u
  - -f do full format listing, adds additional columns
- sort write sorted concatenation to stdout
  - -r reverse result of comparisons
  - -k sort via key
- uniq report or omit repeated lines
  - -c count number of occurrences
- cat concatenate files to print on stdout
- head output first part of file to stdout (def. 10)
  - -n output first N number of lines (with leading - print all but last N)
  - -c print first N bytes of each file (leading - as above)
  - -q quiet, never print headers giving file name
- tail output last part of file to stdout (def. 10)
  - -n output last N number of lines
- cut remove sections from each line of flines
  - -f select only these fields; print any line that contains no delimiter character, unless -s is specified. Can specify multiple fields with a comma (e.g. -f1,3,4). Can specify ranges with dashes. N- Nth onwards, -M up to M, N-M n to m
  - -s do not print lines not containing delimiters
  - -d specify delimiter (e.g. -d':')
- wc print newline, word and byte counts for each file
  - -l number of newlines
  - -c byte count
  - -m character count
- diff compare files line by line
  - -q report only when files differ
  - -s report identical files
- svn subversion
  - commit send changes from working copy to repository
  - add put files in directory under version control. Added to repository in next commit
  - remove remove files and directories from VC. Scheduled for deletion upon next commit and removed from working copy.
  - move move and/or name something in working copy or repository
  - update bring changes from repository into working copy
  - info display information about a local or remote item
  - log show log messages for a set revision(s) and/or path(s)
  - status print the status of working copy and files and directories
  - diff display differences between two revisions or paths
- chmod change file modes (permissions)
  - u/g/o/a user/group/others/all
  - +/- add remove
  - r/w/x read/write/execute
  - -c only report when a change is made
  - -R change files and directories recursively
  - -v output a diagnostic for eveyr file processed
  - -f suppress most error messages
- rm remove
  - -r remove directories and their contents recursively, i.e. delete everything in the specified subdirectories
  - -f force -> ignore nonexistent files and arguments, never prompt
  - -d remove empty directories
  - -v explain what is being done (verbose)
  - * remove all
- mkdir make directories
  - -m set file mode like chmod
  - -p no error if existing parent, make parent directories as needed
- rmdir remove empty directories
  - -p remove directory and its ancestors e.g. 'rmdir -p a/b/c' becomes 'rmdir a/b/c a/b a'
- cp copy files
  - -r copy directories recursively
- scp secure copy
  - -c selects the cipher to use for encrypting data, this option is directly passed to ssh(1)
- mv move files / rename
  - -f force; do not prompt before overwriting
  - -i interactive; prompt before overwrite
- vim programmers text editor
- pico simple text editor
- less allows backward and forward movement
- ln makes links between files (source is first, destination is second)
  - -s for symbolic link
  - -p for physical/hard link

Networks

Five-layer TCP/IP stack:

1. **Physical layer** Medium through which signals travel through.
2. **(Data-)link layer** where peers can communicate directly via messages (MAC addresses)
3. **Network layer** exchange messages with any other host on the "internet", uses IP protocol, IPv4 addresses are 32bit dotted quads, sends messages via the link layer.
4. **Transport layer** UDP user datagram protocol (datagrams)/TCP transmission control protocol (segments), sends messages using the network layer, addresses are ports (16 bit int, restricted below 1024)
5. **Application layer** "Everything else" / web, ssh, games, SMB. addresses include URL/URI but can be anything.

We have four main addresses:

- Application specific addresses
- Port (differentiate between processes)
- IP (which computer is this process on)
- MAC (which device is this direct message to)

IP addresses and MAC addresses technically don't identify devices. The layer isolation is not completely enforced.

What is the transport layer for? The network layer deals with packets. We'd like streams of bytes and reliable delivery. TCP is good for this.

- Establish connection
- Making a connection requires messages to travel there and back
- Connections are bi-directional

UDP deals with discrete messages and no guarantees of delivery or acknowledgment. So streaming video, games, congested networks (many small operations) UDP is appropriate for.

$$\text{Bandwidth} \neq \text{Latency}$$

Where do ports come from?

- Ports: you choose
- IP: internal (admin chooses) or external (DHCP)
- MAC: default address, can be changed

IPv4 has 32 bit addresses. IPv6 is 128bit. We use IPv4. **Server** a process which waits for requests from clients. **Client** a process which submits requests to a server.

TCP: there is always a client and a server. There is no difference between what a client can do and what a server can do. Connections are bi-directional.

Client steps:

1. Find out the address of the machine you wish to connect to
2. Make a socket (fd)
3. `connect()` to the server
4. Wrap socket descriptor for nicer IO (`dup()` before calling `fdopen()`)

Server steps:

1. Make a socket
2. (Optional) set parameters
3. `bind()` the socket to a port
4. Set the socket to `listen()` for connections
5. Call `accept()` to allow a connection (use the new fd to interact with the client)

`ntohs` converts a 16 bit value from network representation to the machines normal ordering.

Note that `accept()` is a blocking call, so fork or create a pthread or *use a non-blocking call but don't actually tho*.

IP headers have a packet length 2^{16} bytes incl. header, protocol and TTL (reduced each time the packet reaches an interface, packet is dropped if TTL reaches 0).

`ping` sends a message to a device, sends a copy back and calculates the travel time.

IPv4 structure has 32 bits, divided into network and host parts eg

```
130.102 | 72.9
10000010.01100100 | 01001000.00001001
```

Increasing the number of bits in the network parts means a "smaller" network. When sending a message, the network layer needs to make a decision:

1. Send direct to the destination (find the MAC of the destination)
2. Send via another mahcine (find the MAC of the intermediary)

An organisation's network can be divided into **subnets**. A host can directly communicate with everything on the same subnet. Broadcasts will reach all hosts in the subnet. Network $\leftrightarrow$ subnet. We can describe the subnet in two ways, CIDR notation or subnet mask.

130.102.0.0/16 means set all host bits to 0 and the value after / is how many bits are in the network part.

130.102.12.0/24 is subnet of all addresses starting with 130.102.12 for example. Note that the /x of CIDR notation does not need to fall on a byte boundary.

Each subnet will have two addresses reserved: all host bits = zero (minimum host address) and all host bits = one (maximum host address). So subnet `A.B.C.D/x` has $32 - x$ host bits and $2^{32-x} - 2$ usable host addresses (31 is a special case).

A **netmask** is a bit pattern which will map any IP address to the corresponding network address.

- Set all network bits to 1
- Set all host bits to 0.

For example, `/24` with mask `255.255.255.0`.

- 130.102.24.17 → 130.102.24.0
- 130.102.24.250 → 130.102.24.0
- 130.102.21.16 → 130.102.21.0

Needs to be a contiguous string of 1s. Example: `130.102.160.0/20` is
`130.102.10100000.00000000` network bits
`255.255.11110000.00000000` netmask
`255.255.240.0`

- 130.102.163.19 → 130.102.160.0 — yes
- 130.102.171.99 → 130.102.160.0 — yes
- 130.102.176.14 → 130.102.176.0 — yes

What is the broadcast address for use by 117.98.141.19 netmask = 255.254.0.0? Netmask tells us that the network is 117.98.0.0/15 or
01110101.01100010.00000000.00000000/15
Setting the $32 - 15 = 17$ least significant bits to 1 gives
01110101.01100010.00000000.00000000/15
01110101.01100011.11111111.11111111/15 = 117.99.255.255

Give the CIDR form and netmask for the largest network which includes 100.89.19.80, 100.89.19.82 and does not include 100.89.19.97 `yes 100.89.19.80 01100100...00010011.01010000`
`yes 100.89.19.82 01100100...00010011.01010010`
`no 100.89.19.97 01100100...00010011.01100001`
So 100.89.18.80/26 is as big as possible without including 97. The netmask is 255.255.255.224

Special networks

- 10.0.0.0/8
- 172.16.0.0/12
- 192.168.0.0/16
- 169.254.0.0/16 (for auto config)

All addresses in 127.0.0.0/8 are loopback addresses ($2^{24} - 2$ addresses)

NAT Host X wants to connect with address G (sends {src-ip=X, src-port=sp, dest-ip=G, dest-port=80}). Packet arrives at G. G tries to reply with {src-ip=G, src-port=80, dest-ip=X, dest-port=sp} but reply doesn't go anywhere because nobody knows where X is.

NAT is network address translation.

1. $X \rightarrow \ldots \rightarrow R \rightarrow \ldots \rightarrow G$
   {src-ip=X, src-port=sp, dest-ip=G, dest-port=80}
2. Packet arrives at R
3. R modifies address information
   {src-ip=R, src-port=np, dest-ip=G, dest-port=80}
4. ...
5. G receives packet and replies
   {src-ip=G, src-port=80, dest-ip=R, dest-port=np}
6. ...
7. R receives packet and modifies info
   {src-ip=G, src-port=80, dest-ip=X, dest-port=sp}
8. X receives the message

NAT only works because R remembers that port np corresponds to port sp on X. R does not need to be directly connected to X or G.

DNS IP packets need to use IP addresses. Map names to IP addresses. Hosts file is still used but useless.

DNS is domain name service. Each "domain" will have at least two servers which know the name to address mapping for that domain. Have a collection of root nameservers. The root servers know the information for the nameservers for the TLDs. Those servers each know the nameservers for subdomains. DNS is essentially a distributed phonebook.

Queries are UDP messages. Servers can operate iteratively or recursively. DNS responses have TTL (more stable mappings will have longer TTL). Load balancing: different requests for the same name could get different answers. Give answers which are close to the query source. DNS domains are independent of networks.

HTTP is HyperText transfer protocol. Runs on top of TCP (so layer 5). We send:

- `GET /` (i want a page, this is the name)
- `HTTP/1.1` (the protocol we are using)
- A newline

We receive:

- `HTTP/1.1 400 Bad Request` (response is in this protocol, status code, readable version of the status)
- Headers (Content-Type: text/html (what sort of file), Content-Length: 173 (how big is the file))
- Blank line
- HTML content telling us it was a bad request

Note ACK means acknowledgment, so go BACK!!!!

C Standard Library

```
#include <stdio.h>
int printf  (const char *format, ...);
int fprintf (FILE *stream, const char *format
      ↪ , ...);
int sprint  (char *str, const char *format,
      ↪ ...);
int snprintf(char *str, size_t size, const
      ↪ char
            *format, ...);
(returns -1 on error)
int scanf(const char *format, ...);
int fscanf(FILE *stream, const char *format,
      ↪ ...);
int sscanf(const char *str, const char *
      ↪ format, ...);
(returns EOF or number of tokens matched)
FILE *fopen(const char *path, const char *
      ↪ mode);
int fclose(FILE *fp);
int fgetc(FILE *stream);
char *fgets(char *s, int size, FILE *stream);
int feof(FILE *stream);
(returns NULL, EOF, EOF and NULL on errors)
feof is only set after a failed read!
#include <stdlib.h>
void *malloc(size_t size);
void free(void *ptr);
void *calloc(size_t nmemb, size_t size);
void *realloc(void *ptr, size_t size);
(returns NULL on error)
int atoi(const char *nptr);
long int strtol(const char *nptr, char **
      ↪ endptr,
                int base);
(sets errno on error)
#include <string.h>
void *memset(void *s, int c, size_t n);
void *memcpy(void *dest, const void *src,
      ↪ size_t n);
char *strcpy(char *dest, const char *src);
char *strncpy(char *dest, const char *src,
      ↪ size_t n);
int strcmp(const char *s1, const char *s2);
```

System Calls

```
#include <unistd.h>
pid_t fork(void);
(returns 0 to child, child PID to parent, -1
      ↪ on error)
int execl(const char *path, const char *arg,
      ↪ ...);
int execlp(const char *file, const char *arg,
      ↪  ...);
(does not return on success)
int pipe(int pipefd[2]);
#include <sys/wait.h>
pid_t wait(int *stat_loc);
pid_t waitpid(pid_t pid, int *stat_loc, int
      ↪ options);
(returns PID if reaped, 0 if nothing
      ↪ available)
```

Threading

```
#include <pthread.h> (gcc -pthread)
int pthread_create(pthread_t *thread, const
    pthread_attr_t *attr,
    void *(*start_routine) (void *), void *
          ↪ arg);
int pthread_join(pthread_t thread, void **
      ↪ retval);
int pthread_mutex_init(pthread_mutex_t *
      ↪ restrict
    mutex, const pthread_mutexattr_t *
          ↪ restrict attr);
int pthread_mutex_destroy(pthread_mutex_t *
      ↪ mutex);
```

Networking

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
int socket(int domain, int type, int protocol
      ↪ );
int connect(int sockfd, const struct sockaddr
      ↪  *addr,
            socklen_t addrlen);
int bind(int sockfd, const struct sockaddr *
      ↪ addr,
           socklen_t addrlen);
int listen(int sockfd, int backlog);
int accept(int sockfd, struct sockaddr *addr,
        socklen_t *addrlen);

int getaddrinfo(const char *node, const char
      ↪ *service,
                const struct addrinfo *hints,
                struct addrinfo **res);
void freeaddrinfo(struct addrinfo *res);
```

Misc

```
void qsort(void *base, size_t nmemb, size_t
    ↪ size,
        int (*compar)(const void *, const
            ↪  void *));
```

Read line example

```
char* read_line(FILE* file) {
    char* result = malloc(sizeof(char)*40);
    int position = 0;
    int next = 0;

    while (1) {
        next = fgetc(file);
        if (next == EOF || next == '\n') {
            result[position] = '\0';
            result = reverse(result);
            return result;
        } else {
            result[position++] = (char)next;
        }
        if (position > 30) {
            result = realloc(result, sizeof(
                ↪ char)*position+1);
        }
    }
}
```

Equations

$$\text{Block Number} = \left\lfloor \frac{\text{Address}}{\text{Block Size}} \right\rfloor$$

$$\text{Offset} = \text{Address} \ \% \ \text{Block Size}$$

$$\text{No. of Block Pointers} = \frac{\text{Block Size}}{\text{Block Pointer Size}}$$

$$\text{Subdirectories} = \text{Link Count} - 2$$

2018 Question 11

```
typedef struct {
    int* matrixRow;
    int numCols;
} ThreadArgs;

void* rowsum(void* v) {
    ThreadArgs* threadArgs = (ThreadArgs*)v;
    int rowSum = 0;
    for (int i = 0; i < threadArgs->numCols;
        ↪ i++) {
        rowSum += threadArgs->matrixRow[i];
    }
    return (void*)rowSum;
}

int matrixsum(int** matrix, int rows, int
    ↪ cols) {
    int sum = 0;
    pthread_t ids[cols];
    for (int i = 0; i < rows; i++) {
        int* matrixRow = matrix[i];
        ThreadArgs threadArgs;
        threadArgs.matrixRow = matrixRow;
        threadArgs.numCols = cols;
        pthread_create(&ids[i], rowsum, &
            ↪ threadArgs);
    }
    for (int i = 0; i < rows; i++) {
        void* val;
        pthread_join(ids[i], &val);
        sum += (int)val;
    }
    return sum;
}
```