CSSE2310 Final Exam Crib Sheet

Tom Cranitch

Nov 6, 2019

1 Shell Commands

```
for file in *.pdf; do mv "$file" "old_$file"; done
```

grep		pattern file
	-i	ignore case
	-v	invert match
	-c	count Suppress normal output; instead print a count of matching lines for each input file.
	-L	files without match Suppress normal output; instead print name of each input file from which output would normally be suppressed.
	-l	files with match
	-m	[NUM] max count Stop reading after NUM lines
	-o	only matching Print only the matched parts of a matching line
	-s	no messages Suppress error messages for unreadable files
	-H	with filenames Print filenames for each match. **Default** when there is more then one file.
	-h	no filenames
	-n	line number
	-A	[NUM] after context Print NUM lines of trailing context after match
	-B	[NUM] before context
	-exclude	[GLOB] exclude files with match
	-r	recursive

ls		
	-a	all
	-A	almost all Ignore implied . and ..
	-d	list directory itself, not its files
	-l	long listing format permissions hardlinks user group size month date time/year name
ps		UID PID PPID C STIME TTY TIME CMD
	-e	all processes
	-f	full-format listing
sort		
	-r	reverse
	-k	[s,e] use columns s to e as the key keys are evaluated in order
uniq		
	-c	count occurrences count line
cat		
head		
	-n	number of lines. If n < 0 print upto the last n lines
tail		
	-n	number of lines
chmod		
rm		
	-r	recursive
ln		target newname
	-s	symbolic
cut		option file
	-d	delimiter
	-f	only these fields (e.g. 1,2). First filed is 1.
	-s	only lines with the delimiter
wc		
	-c	bytes
	-m	chars
	-l	lines
	-L	max line length
	-w	words

2 Basic C

Example function pointers: `void (*foo)(void)`; `int (*(*foo)(void))[3]` (returns pointer to array of 3 ints); `typedef void* (*ft)(char*); ft (*var)(int, int) == void* (*(*var)(int))(char*)`.

Typedefing: Function pointer as above. For a struct `typedef structint one; mystruct`.

Common funtions: `qsort(void* base, size_t nmemb, size_t size, int (*compar)(cons void*, const void*))` (compar returns `int` < 0 if if first argument is $<$ second. `char* fgets(char* s, int size, FILE* stream)` (returns s, NULL on error or EOF without reading any chars), `char* strcpy(char* dest, char* src)` (returns dest), `void* realloc(void* ptr, size_t size)` (**NOTE: must set ptr = realloc (i.e. pointer isn't changed)**), `long int strtol(char* nptr, char** endptr, int base)` (set base = 0 for automatic base, endptr can be NULL), `fopen(char* path, char* mode)`, `fdopen(int fd, char* mode)`, `[f/s]printf([FILE* stream/char* str], char* format, ...)` (space for sprintf should be malloc'd first).

3 Processes

After calling `wait(&s)`, calling the following commands on `s` will give the following information,

Command	Returns
WIFEXITED	true if process exited normally
WEXITSTATUS	the exit status of the process
WIFSIGNALED	true if processes was terminated by a signal
WTERMSIG	the signal which caused the process to terminate

To change the program running on a process, call `int execl(char* path, char* arg0, char* arg1, ...)`. Returns -1 if the operation fails, nver returns anything other than -1. To consider the contets of `PATH` use `execlp`. To pass in an array of arguments, use `execv/execvp(char* path, char** argv)`.

Example Fork/Dup Code

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>

char** split_string(const char* line, char sep) {
        int arrpos = 0;
        int linepos = -1;
        int strpos = 0;

        char next;

        char** arr = malloc(sizeof(char*));
        arr[0] = malloc(sizeof(char) * 2);

        while (linepos++, next = line[linepos], next != '\0') {
                if (next != sep) {
                        arr[arrpos] = realloc(arr[arrpos], sizeof(char) * (
                                strpos + 2));
                        arr[arrpos][strpos] = line[linepos];
                        strpos++;
                } else if (line[linepos + 1] != '\0') {
                        arr[arrpos][strpos] = '\0';
                        strpos = 0;
                        arrpos++;
                        arr = realloc(arr, sizeof(char*) * (arrpos + 1));
                        arr[arrpos] = malloc(sizeof(char));
                }
        }

        arr[arrpos][strpos] = '\0';
        arrpos++;
        arr = realloc(arr, sizeof(char*) * (arrpos + 1));
        arr[arrpos] = (char*) 0;
        return arr;
}

void run_part(const char** argv, int a, int b) {
        if (!fork()) {
                dup2(a, STDIN_FILENO);
                dup2(b, STDOUT_FILENO);
                //close(a);
                //close(b);
                execvp(argv[0], argv);
        }
}

enum ReadWrite {
        READ = 0,
        WRITE = 1
};

void run_cmd(const char* cmd, int* to, int* from) {
        int readfrom = *to;
        char** cmds = split_string(cmd, '|');
        int fds[2];

        for (int i = 0; cmds[i]; i++) {
                char** cmdarr = split_string(cmds[i], ' ');
                if (!cmds[i + 1]) {
                        run_part(cmdarr, readfrom, *from);
                } else {
                        pipe(fds);
                        run_part(cmdarr, readfrom, fds[WRITE]);
                        readfrom = fds[READ];
                }
        }

}

int main(int argc, char** argv) {
        /* Part A
        char* str = "aaa bbb ccc ddd efg hijkl";
        char** arr = split_string(str, ' ');

        char* next;
        for (int i = 0; next = arr[i], next != (char*) 0; i++) {
                printf("%s\n", next);
        }*/

        /* Part B
        char** cmd = malloc(sizeof(char*) * 2);
        cmd[0] = "ls";
        cmd[1] = "-l";

        run_part(cmd, STDIN_FILENO, STDOUT_FILENO); */

        /* Part C
        int* i1 = malloc(sizeof(int));
        int* i2 = malloc(sizeof(int));
        (*i1) = STDIN_FILENO;
        (*i2) = STDOUT_FILENO;
        run_cmd("ls -l|sort|cat", i1, i2); */

        // Part D
        if (argc != 3) {
                return 1;
        }

        int fdin[2];
        int fdout[2];
        pipe(fdin);
```

```
        pipe(fdout);
        run_cmd(argv[1], &fdin[READ], &fdout[WRITE]);

        close(fdin[READ]);
        close(fdout[WRITE]);

        FILE* in = fdopen(fdin[WRITE], "w");
        fprintf(in, "%s\n", argv[2]);
        fflush(in);

        FILE* out = fdopen(fdout[READ], "r");
        int next;
        int count = 0;
        printf("here\n");
        printf("'%c'\n", fgetc(out));
        while(next = fgetc(out), next != EOF) {
                count++;
                printf("%c", next);
        }
        printf("here\n");

        printf("(%d chars)\n", count);

}
```

4 Networking

Physical	Medium signals travel through (e.g. wire, infra-red, pigeons)
(Data-)Link	Peers can communicate directly (e.g. wifi, ethernet frames) (MAC)
Network	Exchange messages with any other host (IP)
Transport	Exchange messages with a process on a host (e.g. UDP/TCP) (Ports)
Application	

For addresses port gives the process, IP gives the computer, MAC gives the device.

C Networking

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.*

Special Addresses and Networks

All host bits zero gives the "network address" and all host bits one gives the "broadcast address". All addresses in `127.0.0.0/8` are "loopback" addresses (localhost). The following addresses are non-routable ("link local") `10.0.0.0/8`, `172.16.0.0/12`, `192.168.0.0/16`, `169.254.0.0/16`.

Example Networking Code

`ntohs` is needed as different systems may have different endianness.

```
/**
 * Starts a server, updates the depot state and displays the port on stdout.
 * For each connection it creates a new thread to handle IO on the thread.
 */
void start_server(DepotState* depotState) {
    struct addrinfo* addrInfo = 0;
    struct addrinfo hints;
    memset(&hints, 0, sizeof(struct addrinfo));
    hints.ai_family = AF_INET;  // Use IPV4
    hints.ai_socktype = SOCK_STREAM;
    hints.ai_flags = AI_PASSIVE;  // We want to listen
    getaddrinfo(LISTEN_ADDRESS, 0, &hints, &addrInfo);

    int serverFD = socket(AF_INET, SOCK_STREAM, 0);
    bind(serverFD, (struct sockaddr*) addrInfo->ai_addr,
            sizeof(struct sockaddr));

    struct sockaddr_in addressInfo;
    memset(&addressInfo, 0, sizeof(struct sockaddr_in));
    socklen_t len = sizeof(struct sockaddr_in);
    getsockname(serverFD, (struct sockaddr*)&addressInfo, &len);

    // Save and display the port
    depotState->port = ntohs(addressInfo.sin_port);
    printf("%u\n", depotState->port);
    fflush(stdout);

    listen(serverFD, NUM_CONNECTION_REQUESTS);

    // Listen for new connection, and for each one create a new thread
    int connectionFD;
    while (1) {
        connectionFD = accept(serverFD, 0, 0);

        if (connectionFD < 0) {
            continue;
        }
        pthread_t depotThread;
        pthread_create(&depotThread, NULL, &open_connection,
                (void*) &connectionInfo);
    }

    return;
}
```

```
/**
 * Connect the another depot on the port given in the message and update the
 * state.
 */
void connect_to_port(Message message, DepotState* state) {
    // +1 for \0 and +1 to round up. Divide by 10 to get the number of digits
    char* port = malloc((message.port / 10 + 2) * sizeof(char));
    sprintf(port, "%u", message.port);  // Port must be a string

    struct addrinfo* addrInfo = 0;
    struct addrinfo hints;
    memset(&hints, 0, sizeof(struct addrinfo));
    hints.ai_family = AF_INET;  // Use IPV4
    hints.ai_socktype = SOCK_STREAM;

    if (getaddrinfo(LISTEN_ADDRESS, port, &hints, &addrInfo)) {
        // Starting failed, so return silently
        freeaddrinfo(addrInfo);
        return;
    }

    int connectionFD = socket(AF_INET, SOCK_STREAM, 0);
    if (connect(connectionFD, (struct sockaddr*) addrInfo->ai_addr,
            sizeof(struct sockaddr))) {
        // Connecting failed, so return silently
        return;
    }

    ConnectionInfo* connectionInfo = malloc(sizeof(ConnectionInfo));
    connectionInfo->depot = malloc(sizeof(Depot));
    connectionInfo->depot->connectionFileDescriptor = connectionFD;
    connectionInfo->depotState = state;

    // Start a thread to handle the connection
    pthread_t depotThread;
    pthread_create(&depotThread, NULL, &open_connection, connectionInfo);
}
```

5 Threads

To create a thread use `pthread_create(pthread_t* thread, const pthread_attr_t* attr, void* (*start_routine) (void*), void* arg)`. To join a thread use `pthread_join(pthread_t thread, void** retval)`. To get the `pthread_t` of the current thread use `pthread_self(void)`.

Semaphores: `sem_init(sem_t* sem, int pshared, unigned int value) == sem_init(sem_t* sem, 0, N)`. In general set `N` to 1. For a producer/consumer task set `N` to 0 and each time the producer adds a job post, with each consumer waiting on the semaphore. To wait/post `sem_wait/sem_post(sem_t* sem)`. Ensure to link with `-pthread`.

Note that if a thread `A` creates a thread `B` and `A` calls `pthread_exit`, then it is possible for `B` to join thread `A`.

Example Threading Code

```
    sigset_t signalSet;  // Block SIGHUP and SIGSEGV
    sigemptyset(&signalSet);
    sigaddset(&signalSet, SIGHUP);
    sigaddset(&signalSet, SIGSEGV);
    pthread_sigmask(SIG_BLOCK, &signalSet, NULL);
    pthread_t sigwaitThread;
    pthread_create(&sigwaitThread, NULL, &listen_to_sighup, &depotState);
```

6 File Systems

The following formulas may be helpful,

$$\text{pointers/block} = \frac{\text{blocksize}}{\text{pointersize}}$$

$$\text{totalblocks} = \#\text{ direct pointers} + \#\text{ single indirect} \cdot \text{pointers/block} + \ldots$$

$$\text{maxsize} = \text{blocksize} \cdot \text{totalblocks}$$

If the question asks about replacment, don't forget to subtract any existing blocks.

Remember to check block size for removing file questions.

7 Programming Questions - Common Mistakes

- When creating a thread `arg` should be a pointer (i.e. don't `arg`).
- `num >= '0' && num <= '9'` not `num >= 0 && num <= 9`
- Make sure the `_t` has been set by `pthread_create` before storing