Computer Systems Fundamentals

hello_syscalls.c

hello world implemented with direct syscall

```c
#include <unistd.h>

int main(void) {
    char bytes[16] = "Hello, Andrew!\n";

    // argument 1 to syscall is  system call number, 1 == write
    // remaining arguments are specific to each system call

    // write system call takes 3 arguments:
    //   1) file descriptor, 1 == stdout
    //   2) memory address of first byte to write
    //   3) number of bytes to write

    syscall(1, 1, bytes, 15); // prints Hello, Andrew! on stdout

    return 0;
}
```

cat_syscalls.c

copy stdin to stdout implemented with system calls

```c
#include <unistd.h>

int main(void) {
    while (1) {
        char bytes[4096];

        // system call number 0 == read
        // read system call takes 3 arguments:
        //   1) file descriptor, 1 == stdin
        //   2) memory address to put bytes read
        //   3) maximum number of bytes read
        // returns number of bytes actually read

        long bytes_read = syscall(0, 0, bytes, 4096);

        if (bytes_read <= 0) {
            break;
        }

        syscall(1, 1, bytes, bytes_read); // prints bytes to stdout
    }

    return 0;
}
```

cp_syscalls.c

cp <file1> <file2> implemented with syscalls and *zero* error handling

```c
#include <unistd.h>

int main(int argc, char *argv[]) {
    // system call number 2 == open
    // open system call takes 3 arguments:
    //   1) address of zero-terminated string containing pathname of file to open
    //   2) bitmap indicating whether to write, read, ... file
    //      0x41 == write to file creating if necessary
    //   3) permissions if file will be newly created
    //      0644 == readable to everyone, writeable by owner

    long read_file_descriptor = syscall(2, argv[1], 0, 0);
    long write_file_descriptor = syscall(2, argv[2], 0x41, 0644);

    while (1) {
        char bytes[4096];
        long bytes_read = syscall(0, read_file_descriptor, bytes, 4096);
        if (bytes_read <= 0) {
            break;
        }
        syscall(1, write_file_descriptor, bytes, bytes_read);
    }

    return 0;
}
```

hello_libc.c

hello world implemented with libc

```c
#include <unistd.h>

int main(void) {
    char bytes[16] = "Hello, Andrew!\n";

    // write takes 3 arguments:
    //   1) file descriptor, 1 == stdout
    //   2) memory address of first byte to write
    //   3) number of bytes to write

    write(1, bytes, 15); // prints Hello, Andrew! on stdout

    return 0;
}
```

cat_libc.c

copy stdin to stdout implemented with libc

```c
#include <unistd.h>

int main(void) {
    while (1) {
        char bytes[4096];

        // system call number 0 == read
        // read system call takes 3 arguments:
        //   1) file descriptor, 1 == stdin
        //   2) memory address to put bytes read
        //   3) maximum number of bytes read
        // returns number of bytes actually read

        ssize_t bytes_read = read(0, bytes, 4096);

        if (bytes_read <= 0) {
            break;
        }

        write(1, bytes, bytes_read); // prints bytes to stdout
    }

    return 0;
}
```

cp_libc.c

cp <file1> <file2> implemented with libc and *zero* error handling

```c
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int main(int argc, char *argv[]) {
    // open takes 3 arguments:
    //   1) address of zero-terminated string containing pathname of file to open
    //   2) bitmap indicating whether to write, read, ... file
    //   3) permissions if file will be newly created
    //      0644 == readable to everyone, writeable by owner

    int read_file_descriptor = open(argv[1], O_RDONLY);
    int write_file_descriptor = open(argv[2], O_WRONLY | O_CREAT, 0644);

    while (1) {
        char bytes[4096];
        ssize_t bytes_read = read(read_file_descriptor, bytes, 4096);
        if (bytes_read <= 0) {
            break;
        }
        write(write_file_descriptor, bytes, bytes_read);
    }

    return 0;
}
```

hello_fputc.c

hello world implemented with fputc

```c
#include <stdio.h>

int main(void) {
    char bytes[16] = "Hello, Andrew!\n";

    for (int i = 0; i < 15; i++) {
        fputc(bytes[i], stdout);
    }

    // or as we know bytes is null-terminated: bytes[15] == '\0'

    for (int i = 0; bytes[i] != '\0'; i++) {
        fputc(bytes[i], stdout);
    }

    // or if you prefer pointers

    for (char *p = &bytes[0]; *p != '\0'; p++) {
        fputc(*p, stdout);
    }

    return 0;
}
```

hello_fputs.c

hello world implemented with fputs

```c
#include <stdio.h>

int main(void) {
    char bytes[] = "Hello, Andrew!\n";

    fputs(bytes, stdout); // relies on bytes being nul-terminated

    return 0;
}
```

hello_fwrite.c

hello world implemented with fwrite

```c
#include <stdio.h>

int main(void) {
    char bytes[] = "Hello, Andrew!\n";

    fwrite(bytes, 1, 15, stdout); // prints Hello, Andrew! on stdout

    return 0;
}
```

hello_fprintf.c

hello world implemented with fwrite

```c
#include <stdio.h>

int main(void) {
    // fprintf same as fprintf except first argument
    // is the stream to write to
    fprintf(stdout, "Hello, Andrew!\n");

    return 0;
}
```

cat_fgetc.c

copy stdin to stdout implemented with fgetc

```c
#include <stdio.h>

int main(void) {
    // c can not be char (common bug)
    // fgetc returns 0..255 and EOF (usually -1)
    int c;

    // return  bytes from the stream (stdin) one at a time
    while ((c = fgetc(stdin)) != EOF) {
        fputc(c, stdout); // write the byte to standard output
    }

    return 0;
}
```

cat_fgets.c

copy stdin to stdout implemented with fgets

```c
#include <stdio.h>

int main(void) {
    // return  bytes from the stream (stdin) line at a time
    // BUFSIZ is defined in stdio.h - its an efficient value to use
    // but any value would work

    // NOTE: fgets returns a null-terminated string
    //       in other words a 0 byte marks the end of the bytes read

    // so fgets can not be used to read data containing bytes which are 0
    // also fputs takes a null-terminated string so it can not be used to write bytes which are 0
    // in other word you can't use fget/fputs for binary data e.g. jpgs

    char line[BUFSIZ];
    while (fgets(line, BUFSIZ, stdin) != NULL) {
        fputs(line, stdout);
    }

    return 0;
}
```

cat_fwrite.c

copy stdin to stdout implemented with fwrite

```c
#include <stdio.h>

int main(void) {
    while (1) {
        char bytes[4096];

        // system call number 0 == read
        // read system call takes 3 arguments:
        //   1) file descriptor, 1 == stdin
        //   2) memory address to put bytes read
        //   3) maximum number of bytes read
        // returns number of bytes actually read

        ssize_t bytes_read = fread(bytes, 1, 4096, stdin);

        if (bytes_read <= 0) {
            break;
        }

        fwrite(bytes, 1, bytes_read, stdout); // prints bytes to stdout
    }

    return 0;
}
```

cp_fgetc.c

cp <file1> <file2> implemented with fgetc

```c
#include <stdio.h>

int main(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "Usage: %s <source file> <destination file>\n", argv[0]);
        return 1;
    }

    FILE *input_stream = fopen(argv[1], "rb");
    if (input_stream == NULL) {
        perror(argv[1]);  // prints why the open failed
        return 1;
    }

    FILE *output_stream = fopen(argv[2], "wb");
    if (output_stream == NULL) {
        perror(argv[2]);
        return 1;
    }

    int c; // not char!
    while ((c = fgetc(input_stream)) != EOF) {
        fputc(c, output_stream);
    }

    // close occurs automatically on exit
    // so these lines not nee
    fclose(input_stream);
    fclose(output_stream);

    return 0;
}
```

cp_fwrite.c

cp <file1> <file2> implemented with libc and *zero* error handling

```c
#include <stdio.h>

int main(int argc, char *argv[]) {
    // open takes 3 arguments:
    //   1) address of zero-terminated string containing pathname of file to open
    //   2) bitmap indicating whether to write, read, ... file
    //   3) permissions if file will be newly created
    //      0644 == readable to everyone, writeable by owner

    // b = binary mode - not needed on Linux, OSX (POSIX) systems
    //                 - needed on Windows

    FILE *read_stream = fopen(argv[1], "rb");
    FILE *write_stream = fopen(argv[2], "wb");

    // this will be slightly faster than an a fgetc/fputc loop
    while (1) {
        char bytes[BUFSIZ];
        size_t bytes_read = fread(bytes, 1, 4096, read_stream);
        if (bytes_read <= 0) {
            break;
        }
        fwrite(bytes, 1, bytes_read, write_stream);
    }

    return 0;
}
```

create_file_fopen.c

Simple example of file creation creates file "hello.txt" containing 1 line ("Hello, Andrew!\n")

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    FILE *output_stream = fopen("hello.txt", "w");
    if (output_stream == NULL) {
        perror("hello.txt");
        return 1;
    }

    fprintf(output_stream, "Hello, Andrew!\n");

    fclose(output_stream);

    return 0;
}
```

create_append_truncate_fopen.c

```
$ dcc create_append_truncate_fopen.c
$ ./a.out
open("hello.txt", "w")           -> -rw-r--r-- 1 andrewt andrewt 0 Oct 22 19:11 hello.txt
fputs("Hello, Andrew!\n")        -> -rw-r--r-- 1 andrewt andrewt 0 Oct 22 19:11 hello.txt
fclose                           -> -rw-r--r-- 1 andrewt andrewt 15 Oct 22 19:11 hello.txt
fopen("hello.txt", "a")          -> -rw-r--r-- 1 andrewt andrewt 15 Oct 22 19:11 hello.txt
fputs("Hello again, Andrew!\n")  -> -rw-r--r-- 1 andrewt andrewt 15 Oct 22 19:11 hello.txt
fflush                           -> -rw-r--r-- 1 andrewt andrewt 36 Oct 22 19:11 hello.txt
open("hello.txt", "w")           -> -rw-r--r-- 1 andrewt andrewt 0 Oct 22 19:11 hello.txt
fputs("Good Bye Andrew!\n")      -> -rw-r--r-- 1 andrewt andrewt 0 Oct 22 19:11 hello.txt
assa:files% ./a.out
open("hello.txt", "w")           -> -rw-r--r-- 1 andrewt andrewt 0 Oct 22 19:12 hello.txt
fputs("Hello, Andrew!\n")        -> -rw-r--r-- 1 andrewt andrewt 0 Oct 22 19:12 hello.txt
fclose                           -> -rw-r--r-- 1 andrewt andrewt 15 Oct 22 19:12 hello.txt
fopen("hello.txt", "a")          -> -rw-r--r-- 1 andrewt andrewt 15 Oct 22 19:12 hello.txt
fputs("Hello again, Andrew!\n")  -> -rw-r--r-- 1 andrewt andrewt 15 Oct 22 19:12 hello.txt
fflush                           -> -rw-r--r-- 1 andrewt andrewt 36 Oct 22 19:12 hello.txt
open("hello.txt", "w")           -> -rw-r--r-- 1 andrewt andrewt 0 Oct 22 19:12 hello.txt
fputs("Good Bye Andrew!\n")      -> -rw-r--r-- 1 andrewt andrewt 0 Oct 22 19:12 hello.txt
$ ls -l hello.txt
-rw-r--r-- 1 andrewt andrewt 17 Oct 22 19:12 hello.txt
$ cat hello.txt

Good Bye Andrew!
$
```

```c
#include <stdio.h>
#include <stdlib.h>

void show_file_state(char *message);

int main(int argc, char *argv[]) {
    FILE *output_stream1 = fopen("hello.txt", "w"); // no error checking

    // hello.txt will be created if it doesn't exist already
    // if hello.txt previous existed it will now contain 0 bytes

    show_file_state("open(\"hello.txt\", \"w\")");

    fputs("Hello, Andrew!\n", output_stream1);

    // the 15 bytes in "Hello, Andrew!\n" are buffered by the stdio library
    // they haven't been written to hello.txt
    // so it will still contain 0 bytes

    show_file_state("fputs(\"Hello, Andrew!\\n\")");

    fclose(output_stream1);

    // The fclose will flush the buffered bytes to hello.txt
    // hello.txt will now contain 15 bytes

    show_file_state("fclose()");

    FILE *output_stream2 = fopen("hello.txt", "a"); // no error checking

    // because "a" was specified hello.txt will not be changed
    // it will still contain 15 bytes

    show_file_state("fopen(\"hello.txt\", \"a\")");

    fputs("Hello again, Andrew!\n", output_stream2);

    // the 21 bytes in "Hello again, Andrew!\n" are buffered by the stdio library
    // they haven't been written to hello.txt
    // so it will still contain 15 bytes

    show_file_state("fputs(\"Hello again, Andrew!\\n\")");

    fflush(output_stream2);

    // The fflush will flush ahe buffered bytes to hello.txt
    // hello.txt will now contain 36 bytes

    show_file_state("fflush()");

    FILE *output_stream3 = fopen("hello.txt", "w"); // no error checking

    // because "w" was specified hello.txt will be truncated to zero length
    // hello.txt will now contain 0 bytes

    show_file_state("open(\"hello.txt\", \"w\")");

    fputs("Good Bye Andrew!\n", output_stream3);

    // the 17 bytes in "Good Bye Andrew!\" are buffered by the stdio library
    // they haven't been written to hello.txt
    // so it will still contain 0 bytes

    show_file_state("fputs(\"Good Bye Andrew!\\n\")");

    // if exit is called or main returns stdio flushes all stream
    // this will leave hello.txt with 17 bytes
    // but if a program terminates abnormally this doesn't happen

    return 0;
}
```

```
void show_file_state(char *message) {
    printf("%-32s -> ", message);
    fflush(stdout);
    system("ls -l hello.txt");
}
```

myio_unbuffered.c

simple re-implementation of stdio functions fopen, fgetc, fputc, fclose no buffering *zero* error handling for clarity

```c
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdint.h>
#include <stdlib.h>
#include <assert.h>
#include <stdio.h>

#define MY_EOF -1

// struct to hold data for a stream
typedef struct my_file {
    int fd;
} my_file_t;

my_file_t *my_fopen(char *file, char *mode) {
    int fd = -1;
    if (mode[0] == 'r') {
        fd = open(file, O_RDONLY);
    } else if (mode[0] == 'w') {
        fd = open(file, O_WRONLY | O_CREAT, 0666);
    } else if (mode[0] == 'a') {
        fd = open(file, O_WRONLY | O_APPEND);
    }

    if (fd == -1) {
        return NULL;
    }

    my_file_t *f = malloc(sizeof *f);
    f->fd = fd;
    return f;
}

int my_fgetc(my_file_t *f) {
    uint8_t byte;
    int bytes_read = read(f->fd, &byte, 1);
    if (bytes_read == 1) {
        return byte;
    } else {
        return MY_EOF;
    }
}

int my_fputc(int c, my_file_t *f) {
    uint8_t byte = c;
    if (write(f->fd, &byte, 1) == 1) {
        return byte;
    } else {
        return MY_EOF;
    }
}

int my_fclose(my_file_t *f) {
    int result = close(f->fd);
    free(f);
    return result;
}

int main(int argc, char *argv[]) {
    my_file_t *input_stream = my_fopen(argv[1], "r");
    if (input_stream == NULL) {
        perror(argv[1]);
        return 1;
    }

    my_file_t *output_stream = my_fopen(argv[2], "w");
    if (output_stream == NULL) {
        perror(argv[2]);
        return 1;
```

```
    }

    int c;
    while ((c = my_fgetc(input_stream)) != MY_EOF) {
        my_fputc(c, output_stream);
    }

    return 0;
}
```

myio_input_buffered.c

simple re-implementation of stdio functions fopen, fgetc, fputc, fclose input buffering *zero* error handling for clarity

```c
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdint.h>
#include <stdlib.h>
#include <assert.h>
#include <stdio.h>

// how equivalents for  EOF & BUFSIZ from stdio.h
#define MY_EOF -1
#define MY_BUFSIZ 512

// struct to hold data for a stream
typedef struct my_file {
    int      fd;
    int      n_buffered_bytes;
    int      next_byte;
    uint8_t buffer[MY_BUFSIZ];
} my_file_t;

my_file_t *my_fopen(char *file, char *mode) {
    int fd = -1;
    if (mode[0] == 'r') {
        fd = open(file, O_RDONLY);
    } else if (mode[0] == 'w') {
        fd = open(file, O_WRONLY | O_CREAT, 0666);
    } else if (mode[0] == 'a') {
        fd = open(file, O_WRONLY | O_APPEND);
    }

    if (fd == -1) {
        return NULL;
    }

    my_file_t *f = malloc(sizeof *f);
    f->fd = fd;
    f->next_byte = 0;
    f->n_buffered_bytes = 0;
    return f;
}

int my_fgetc(my_file_t *f) {
    if (f->next_byte == f->n_buffered_bytes) {
        // buffer is empty so fill it with a read
        int bytes_read = read(f->fd, f->buffer, sizeof f->buffer);
        if (bytes_read <= 0) {
            return MY_EOF;
        }
        f->n_buffered_bytes = bytes_read;
        f->next_byte = 0;
    }

    // return 1 byte from the buffer
    int byte = f->buffer[f->next_byte];
    f->next_byte++;
    return byte;
}

int my_fputc(int c, my_file_t *f) {
    uint8_t byte = c;
    if (write(f->fd, &byte, 1) == 1) {
        return byte;
    } else {
        return MY_EOF;
    }
}

int my_fclose(my_file_t *f) {
    int result = close(f->fd);
    free(f);
```

```
    return result;
}

int main(int argc, char *argv[]) {
    my_file_t *input_stream = my_fopen(argv[1], "r");
    if (input_stream == NULL) {
        perror(argv[1]);
        return 1;
    }

    my_file_t *output_stream = my_fopen(argv[2], "w");
    if (output_stream == NULL) {
        perror(argv[2]);
        return 1;
    }

    int c;
    while ((c = my_fgetc(input_stream)) != MY_EOF) {
        my_fputc(c, output_stream);
    }

    my_fclose(input_stream);
    my_fclose(output_stream);

    return 0;
}
```

myio_output_buffered.c

simple re-implementation of stdio functions fopen, fgetc, fputc, fclose *zero* error handling for clarity

```c
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdint.h>
#include <stdlib.h>
#include <stdio.h>

// how equivalents for  EOF & BUFSIZ from stdio.h
#define MY_EOF -1
#define MY_BUFSIZ 512

// struct to hold data for a stream
typedef struct my_file {
    int      fd;
    int      is_output_stream;
    int      n_buffered_bytes;
    int      next_byte;
    uint8_t buffer[MY_BUFSIZ];
} my_file_t;

my_file_t *my_fopen(char *file, char *mode) {
    int fd = -1;
    if (mode[0] == 'r') {
        fd = open(file, O_RDONLY);
    } else if (mode[0] == 'w') {
        fd = open(file, O_WRONLY | O_CREAT, 0666);
    } else if (mode[0] == 'a') {
        fd = open(file, O_WRONLY | O_APPEND);
    }

    if (fd == -1) {
        return NULL;
    }

    my_file_t *f = malloc(sizeof *f);
    f->fd = fd;
    f->is_output_stream = mode[0] != 'r';
    f->next_byte = 0;
    f->n_buffered_bytes = 0;
    return f;
}

int my_fgetc(my_file_t *f) {
    if (f->next_byte == f->n_buffered_bytes) {
        // buffer is empty so fill it with a read
        int bytes_read = read(f->fd, f->buffer, sizeof f->buffer);
        if (bytes_read <= 0) {
            return MY_EOF;
        }
        f->n_buffered_bytes = bytes_read;
        f->next_byte = 0;
    }

    // return 1 byte from the buffer
    int byte = f->buffer[f->next_byte];
    f->next_byte++;
    return byte;
}

int my_fputc(int c, my_file_t *f) {
    if (f->n_buffered_bytes == sizeof f->buffer) {
        // buffer is full so empty it with a write
        write(f->fd, f->buffer, sizeof f->buffer); // no error checking
        f->n_buffered_bytes = 0;
    }

    // add byte byte to buffer to be written later
    f->buffer[f->n_buffered_bytes] = c;
    f->n_buffered_bytes++;
    return 1;
```

```
}

int my_fclose(my_file_t *f) {
    // don't keave unwritten bytes
    if (f->is_output_stream && f->n_buffered_bytes > 0) {
        write(f->fd, f->buffer, f->n_buffered_bytes); // no error checking
    }

    int result = close(f->fd);
    free(f);
    return result;
}

int main(int argc, char *argv[]) {
    my_file_t *input_stream = my_fopen(argv[1], "r");
    if (input_stream == NULL) {
        perror(argv[1]);
        return 1;
    }

    my_file_t *output_stream = my_fopen(argv[2], "w");
    if (output_stream == NULL) {
        perror(argv[2]);
        return 1;
    }

    int c;
    while ((c = my_fgetc(input_stream)) != MY_EOF) {
        my_fputc(c, output_stream);
    }

    my_fclose(input_stream);
    my_fclose(output_stream);

    return 0;
}
```

lseek.c

use lseek to access diferent bytes of a file with no error checking

the return value of thecalls to open, lseek and read should be checked to see if they worked!

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <source file>\n", argv[0]);
        return 1;
    }

    int read_file_descriptor = open(argv[1], O_RDONLY);
    char bytes[1];
    // move to a position 1 byte from end of file
    // then read 1 byte
    lseek(read_file_descriptor, -1, SEEK_END);
    read(read_file_descriptor, bytes, 1);
    printf("The last byte of the file is 0x%02x\n", bytes[0]);

    // move to a position 0 bytes from start of file
    // then read 1 byte
    lseek(read_file_descriptor, 0, SEEK_SET);
    read(read_file_descriptor, bytes, 1);
    printf("The first byte of the file is 0x%02x\n", bytes[0]);

    // move to a position 41 bytes from start of file
    // then read 1 byte
    lseek(read_file_descriptor, 41, SEEK_SET);
    read(read_file_descriptor, bytes, 1);
    printf("The 42nd byte of the file is 0x%02x\n", bytes[0]);

    // move to a position 58 bytes from current position
    // then read 1 byte
    lseek(read_file_descriptor, 58, SEEK_CUR);
    read(read_file_descriptor, bytes, 1);
    printf("The 100th byte of the file is 0x%02x\n", bytes[0]);

    return 0;
}
```

fseek.c

use fseek to access diferent bytes of a file with no error checking

the return value of the calls to fopen, fseek and fgetc should be checked to see if they worked!

```c
#include <stdio.h>

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <source file>\n", argv[0]);
        return 1;
    }

    FILE *input_stream = fopen(argv[1], "rb");

    // move to a position 1 byte from end of file
    // then read 1 byte
    fseek(input_stream, -1, SEEK_END);
    printf("The last byte of the file is 0x%02x\n", fgetc(input_stream));

    // move to a position 0 bytes from start of file
    // then read 1 byte
    fseek(input_stream, 0, SEEK_SET);
    printf("The first byte of the file is 0x%02x\n", fgetc(input_stream));

    // move to a position 41 bytes from start of file
    // then read 1 byte
    fseek(input_stream, 41, SEEK_SET);
    printf("The 42nd byte of the file is 0x%02x\n", fgetc(input_stream));

    // move to a position 58 bytes from current position
    // then read 1 byte
    fseek(input_stream, 58, SEEK_CUR);
    printf("The 100th byte of the file is 0x%02x\n", fgetc(input_stream));

    return 0;
}
```

stat.c

call stat on each command line argument as simple example of its use

```c
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>
#include <stdlib.h>

void stat_file(char *pathname);

int main(int argc, char *argv[]) {
    for (int arg = 1; arg < argc; arg++) {
        stat_file(argv[arg]);
    }
    return 0;
}

void stat_file(char *pathname) {
    struct stat s;

    printf("stat(\"%s\", &s)\n", pathname);

    if (stat(pathname, &s) != 0) {
        perror(pathname);
        exit(1);
    }

    printf(" s.st_ino =   %10ld  # Inode number\n", s.st_ino);
    printf(" s.st_mode =  %10o  # File mode \n", s.st_mode);
    printf(" s.st_nlink = %10ld  # Link count \n", (long)s.st_nlink);
    printf(" s.st_uid =   %10u  # Owner uid\n", s.st_uid);
    printf(" s.st_gid =   %10u  # Group gid\n", s.st_gid);
    printf(" s.st_size =  %10ld  # File size (bytes)\n", (long)s.st_size);

    printf(" s.st_mtime = %10ld  # Modification time (seconds since 01/01/70)\n", (long)s.st_mtime);
}
```

create_gigantic_file.c

```c
#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>
#include <fcntl.h>

int main(void) {
    int fd = open("sparse_file.txt", O_WRONLY | O_CREAT, 0644);
    write(fd, "Hello, Andrew!\n", 15);
    lseek(fd, 16L * 1000 * 1000 * 1000 * 1000, SEEK_CUR);
    write(fd, "Good Bye Andrew!\n", 17);
    close(fd);
    return 0;
}
```

fuzz.c

use fseek to change a random bit in a file the return value of the calls to fopen, fseek and fgetc should be checked to see if they worked!

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <source file>\n", argv[0]);
        return 1;
    }

    // open file for reading and writing
    FILE *f = fopen(argv[1], "r+");

    // move to end of file
    fseek(f, 0, SEEK_END);

    long n_bytes_in_file = ftell(f);

    // seed random number generator with current time
    srandom(time(NULL));

    // pick a random byte
    long target_byte = random() % n_bytes_in_file;

    // move to byte
    fseek(f, target_byte, SEEK_SET);

    // read byte
    int byte = fgetc(f);

    // pick a random bit
    int bit = random() % 7;

    // flip the bit
    int new_byte = byte ^ (1 << bit);

    // move back to write byte to same position
    fseek(f, -1, SEEK_CUR);

    // write the byte
    fputc(new_byte, f);

    fclose(f);

    printf("Changed byte %ld of %s from %02x to %02x\n",target_byte, argv[1], byte, new_byte);
    return 0;
}
```

my_cd.c

useles suse of chdir() because it only affects this process and any it runs

```c
#include <unistd.h>
#include <stdio.h>

int main(int argc, char *argv[]) {
    if (argc > 1 && chdir(argv[1]) != 0) {
        perror("chdir");
        return 1;
    }
    return 0;
}
```

getcwd.c

use repeated chdir("..") to climb to the root of the file system as a silly example of getcwd and chdir

```c
#include <unistd.h>
#include <limits.h>
#include <stdio.h>
#include <string.h>

int main(void) {
    char pathname[PATH_MAX];
    while (1) {
        if (getcwd(pathname, sizeof pathname) == NULL) {
            perror("getcwd");
            return 1;
        }
        printf("getcwd() returned %s\n", pathname);

        if (strcmp(pathname, "/") == 0) {
            return 0;
        }

        if (chdir("..") != 0) {
            perror("chdir");
            return 1;
        }
    }
    return 0;
}
```

read_directory.c

list the contenst of directories specified as command-line arguments

```c
#include <stdio.h>
#include <dirent.h>

/*
$ dcc read_directory.c
$ ./a.out .
read_directory.c
a.out
.
..
$
*/

int main(int argc, char *argv[]) {

    for (int arg = 1; arg < argc; arg++) {
        DIR *dirp = opendir(argv[arg]);
        if (dirp == NULL) {
            perror(argv[arg]);  // prints why the open failed
            return 1;
        }

        struct dirent *de;

        while ((de = readdir(dirp)) != NULL) {
            printf("%ld %s\n", de->d_ino, de->d_name);
        }

        closedir(dirp);
    }
    return 0;
}
```

[create_directory.c]

create the directories specified as command-line arguments

```c
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>

/*
$ dcc create_directory.c
$ ./a.out new_dir
$ ls -ld new_dir
drwxr-xr-x 2 z5555555 z5555555 60 Oct 29 16:28 new_dir
$
*/

int main(int argc, char *argv[]) {

    for (int arg = 1; arg < argc; arg++) {
        if (mkdir(argv[arg], 0755) != 0) {
            perror(argv[arg]);  // prints why the mkdir failed
            return 1;
        }
    }

    return 0;
}
```

[rename.c]

rename the specified file

```c
#include <stdio.h>

/*
$ dcc rename.c
$ ./a.out rename.c renamed.c
$ ls -l  renamed.c
renamed.c
$
*/

int main(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "Usage: %s <old-filename> <new-filename>\n", argv[0]);
        return 1;
    }

    if (rename(argv[1], argv[2]) != 0) {
        fprintf(stderr, "%s rename <old-filename> <new-filename> failed:", argv[0]);
        perror("");
        return 1;
    }

    return 0;
}
```

nest_directories.c

silly program which creates a 1000-deep directory hierarchy

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <limits.h>


int main(int argc, char *argv[]) {

    for (int i = 0; i < 1000;i++) {
        char dirname[256];
        snprintf(dirname, sizeof dirname, "d%d", i);

        if (mkdir(dirname, 0755) != 0) {
            perror(dirname);
            return 1;
        }
        if (chdir(dirname) != 0) {
            perror(dirname);
            return 1;
        }

        char pathname[1000000];
        if (getcwd(pathname, sizeof pathname) == NULL) {
            perror("getcwd");
            return 1;
        }
        printf("\nCurrent directory now: %s\n", pathname);
    }

    return 0;
}
```

many_links.c

silly program which create a 1000 links to file in effect there are 1001 names for the file

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <limits.h>
#include <string.h>


int main(int argc, char *argv[]) {
    char pathname[256] = "hello.txt";

    // create a target file
    FILE *f1;
    if ((f1 = fopen(pathname, "w")) == NULL) {
        perror(pathname);
        return 1;
    }
    fprintf(f1, "Hello Andrew!\n");
    fclose(f1);

    for (int i = 0; i < 1000;i++) {
        printf("Verifying '%s' contains: ", pathname);
        FILE *f2;
        if ((f2 = fopen(pathname, "r")) == NULL) {
            perror(pathname);
            return 1;
        }
        int c;
        while ((c = fgetc(f2)) != EOF) {
            fputc(c, stdout);
        }
        fclose(f2);

        char new_pathname[256];
        snprintf(new_pathname, sizeof new_pathname, "hello_%d.txt", i);

        printf("Creating a link %s -> %s\n", new_pathname, pathname);
        if (link(pathname, new_pathname) != 0) {
            perror(pathname);
            return 1;
        }

    }

    return 0;
}
```

chain_links.c

silly program which attempts to creates a long chain of symbolic links

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <limits.h>
#include <string.h>


int main(int argc, char *argv[]) {
    char pathname[256] = "hello.txt";

    // create target file
    FILE *f1;
    if ((f1 = fopen(pathname, "w")) == NULL) {
        perror(pathname);
        return 1;
    }
    fprintf(f1, "Hello Andrew!\n");
    fclose(f1);

    for (int i = 0; i < 1000;i++) {
        printf("Verifying '%s' contains: ", pathname);
        FILE *f2;
        if ((f2 = fopen(pathname, "r")) == NULL) {
            perror(pathname);
            return 1;
        }
        int c;
        while ((c = fgetc(f2)) != EOF) {
            fputc(c, stdout);
        }
        fclose(f2);

        char new_pathname[256];
        snprintf(new_pathname, sizeof new_pathname, "hello_%d.txt", i);

        printf("Creating a symbolic link %s -> %s\n", new_pathname, pathname);
        if (symlink(pathname, new_pathname) != 0) {
            perror(pathname);
            return 1;
        }

        strcpy(pathname, new_pathname);

    }

    return 0;
}
```

write_array.c

write bytes of array to file array.save

```
$ dcc write_array.c -o write_array
$ dcc read_array.c -o read_array
$ ./write_array
$ ls -l array.save
-rw-r--r-- 1 z5555555 z5555555 40 Oct 30 21:46 array.save
$ ./read_array
10 11 12 13 14 15 16 17 18 19
$
```

```c
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>

int array[10] = {10, 11, 12, 13, 14, 15, 16, 17, 18, 19};

int main(int argc, char *argv[]) {

    int fd = open("array.save", O_WRONLY|O_CREAT, 0644);
    if (fd < 0) {
        perror("array.save");
        return 1;
    }

    if (write(fd, array, sizeof array) != sizeof array) {
        perror("array.save");
        return 1;
    }
    close(fd);

    return 0;
}
```

read_array.c

read bytes of array + pointer to file array_pointer.save non-portable between platforms breaks if sizeof int changes or endian-ness changes

Handling this safely is called serialization: https://en.wikipedia.org/wiki/Serialization

```c
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>

int array[10];

int main(int argc, char *argv[]) {

    int fd = open("array.save", O_RDONLY, 0644);
    if (fd < 0) {
        perror("array.save");
        return 1;
    }

    if (read(fd, array, sizeof array) != sizeof array) {
        perror("array.save");
        return 1;
    }
    close(fd);

    // print array
    for (int i = 0; i < 10; i++) {
        printf("%d ", array[i]);
    }
    printf("\n");

    return 0;
}
```

write_pointer.c

write bytes of array + pointer to file array_pointer.save

```
$ dcc write_pointer.c -o write_pointer
$ dcc read_pointer.c -o read_pointer
$ ./write_pointer
p         = 0x410234
&array[5] = 0x410234
array[5]  = 15
*p        = 15
$ ls -l array_pointer.save
-rw-r--r-- 1 z5555555 z5555555 48 Oct 30 21:46 array.save
$ ./read_pointer
10 11 12 13 14 15 16 17 18 19
p         = 0x410234
&array[5] = 0x4163f4
array[5]  = 15
*p        = -1203175425
$
```

```c
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>

int array[10] = {10, 11, 12, 13, 14, 15, 16, 17, 18, 19};
int *p = &array[5];

int main(int argc, char *argv[]) {

    int fd = open(".save", O_WRONLY|O_CREAT, 0644);
    if (fd < 0) {
        perror("array_pointer.save");
        return 1;
    }

    if (write(fd, array, sizeof array) != sizeof array) {
        perror("array_pointer.save");
        return 1;
    }

    if (write(fd, &p, sizeof p) != sizeof p) {
        perror("array_pointer.save");
        return 1;
    }

    close(fd);

    printf("p         = %p\n", p);
    printf("&array[5] = %p\n", &array[5]);
    printf("array[5]  = %d\n", array[5]);
    printf("*p        = %d\n", *p);
    return 0;
}
```

read_pointer.c

read bytes of array + pointer to file array_pointer.save breaks even on same machine because address of array different for every execution see https://en.wikipedia.org/wiki/Address_space_layout_randomization

```c
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>

int array[10];
int *p;

int main(int argc, char *argv[]) {

    int fd = open("array_pointer.save", O_RDONLY, 0644);
    if (fd < 0) {
        perror("array_pointer.save");
        return 1;
    }

    if (read(fd, array, sizeof array) != sizeof array) {
        perror("array_pointer.save");
        return 1;
    }

    if (read(fd, &p, sizeof p) != sizeof p) {
        perror("array_pointer.save");
        return 1;
    }

    close(fd);

    // print array
    for (int i = 0; i < 10; i++) {
        printf("%d ", array[i]);
    }
    printf("\n");

    printf("p         = %p\n", p);
    printf("&array[5] = %p\n", &array[5]);
    printf("array[5]  = %d\n", array[5]);
    printf("*p        = %d\n", *p);

    return 0;
}
```