rm0

```
#define _POSIX_C_SOURCE 200809L
#include <unistd.h>
//-----------------------------------------------------------------------------
// RM0
//-----------------------------------------------------------------------------
void rm0(const char* file) {
    //Pre: el archivo existe y es regular.
   unlink(file);
}
//-----------------------------------------------------------------------------
// MAIN
//-----------------------------------------------------------------------------
int main(int argc, char* argv[]) {
    rm0(argv[1]);
    return 0;
}
//-----------------------------------------------------------------------------
```

cat0

```
#define _POSIX_C_SOURCE 200809L
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include "functions.h"
//-----------------------------------------------------------------------------
// CAT 0
//-----------------------------------------------------------------------------
int cat0(const char* pathName) {
    //pre: solo se pasa un archivo, este archivo existe
    // y se tienen permisos de lectura.
    char buffer[256];
    int fdOut = fileno(stdout);
    int fd = open(pathName, O_RDONLY);
    ssize_t read = 1, written = 1;
    while (read != 0 && written != 0) {
        read = readArchive(fd, buffer, 256);
        written = writeArchive(fdOut, buffer, (size_t) read);
        if (written == -1 || read == -1) return 1;
    }
    if (close(fd) == -1) return 1;
    return 0;
}
//---------------------------------------------------------------------------------
// MAIN
//---------------------------------------------------------------------------------
int main(int argc, char* argv[]) {
    return cat0(argv[1]);
}
//---------------------------------------------------------------------------------
```

touch0

```
#define _POSIX_C_SOURCE 200809L
#include <unistd.h>
#include <fcntl.h>
//---------------------------------------------------------------------------------
// TOUCH0
//---------------------------------------------------------------------------------
int touch0(const char* file) {
    //Pre: si el archivo existe, es un archivo regular.
    int fd = open(file, O_WRONLY);
    if (fd == -1) {
        fd = open(file, O_CREAT, S_IRWXU);
    }
    if (close(fd) == -1) return 1;
    return 0;
}
//---------------------------------------------------------------------------------
// MAIN
//---------------------------------------------------------------------------------
int main(int argc, char* argv[]) {
    return touch0(argv[1]);
}
//---------------------------------------------------------------------------------
```

stat0

```
#define _POSIX_C_SOURCE 200809L
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <stdio.h>
#include <errno.h>
```

```
//--------------------------------------------------------------------------------
// STAT0
//--------------------------------------------------------------------------------
int stat0(const char* file) {
    //Pre: el archivo existe, y es un directorio o un archivo regular.
    //Pos: $./stat0 README.md
    //Size: 1318
    //File: README.md
    //Type: regular file
    struct stat buffer;
    int fd = open(file, O_RDONLY);
    int value = stat(file, &buffer);
    if (value == -1) {
        printf("ERROR at function stat0: %d\n", errno);
        return value;
    }
    // Total size, in bytes
    off_t st_size = buffer.st_size;
    // Protection
    /*
        S_IFMT     0170000   bit mask for the file type bit field

        S_IFSOCK   0140000   socket
        S_IFLNK    0120000   symbolic link
        S_IFREG    0100000   regular file
        S_IFBLK    0060000   block device
        S_IFDIR    0040000   directory
        S_IFCHR    0020000   character device
        S_IFIFO    0010000   FIFO

     */
    char* mode = "";
    switch (buffer.st_mode & S_IFMT) {
        case S_IFSOCK:
            mode = "socket";
            break;
        case S_IFLNK:
            mode = "symbolic link";
            break;
        case S_IFREG:
            mode = "regular file";
            break;
        case S_IFBLK:
            mode = "block device";
            break;
        case S_IFDIR:
            mode = "directory";
            break;
        case S_IFCHR:
            mode = "character device";
            break;
        case S_IFIFO:
            mode = "FIFO";
            break;
        default:
            break;
    }
    printf("Size: %d\nFile: %s\nType: %s\n", (int) st_size, file, mode);
    if (close(fd) == -1) return 1;
    return 0;
}
//--------------------------------------------------------------------------------
// MAIN
//--------------------------------------------------------------------------------
int main(int argc, char* argv[]) {
    return stat0(argv[1]);
}
//--------------------------------------------------------------------------------
```

rm1

```
#define _POSIX_C_SOURCE 200809L
#include <unistd.h>
#include <stdio.h>
#include <string.h>
//--------------------------------------------------------------------------------
// RM1
//--------------------------------------------------------------------------------
int rm1(const char* file) {
    //Pre: el archivo existe, y es un directorio o un archivo regular.
    if (unlink(file) == -1) {
        int bytes = 0;
        char msg[256];
        size_t size = strlen("cannot remove ") + strlen(file) + 1;
        snprintf(msg+bytes, size,"cannot remove %s", file);
        perror(msg);
        return 1;
    }
    return 0;
}
```

```
//----------------------------------------------------------------------------
// MAIN
//----------------------------------------------------------------------------
int main(int argc, char* argv[]) {
    return rm1(argv[1]);
}
//----------------------------------------------------------------------------
```

ln0

```
#define _POSIX_C_SOURCE 200809L
#include <unistd.h>
#include <stdio.h>
//----------------------------------------------------------------------------
// LN0
//----------------------------------------------------------------------------
int ln0(const char *target, const char *linkPath) {
    // Pre: no existe un archivo con el nombre del enlace.
    int value =  symlink(target, linkPath);
    if (value == -1) {
        perror("");
        return 1;
    }
    return 0;
}
//----------------------------------------------------------------------------
// MAIN
//----------------------------------------------------------------------------
int main(int argc, char* argv[]) {
    return ln0(argv[1], argv[2]);
}
//----------------------------------------------------------------------------
```

mv0

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
//----------------------------------------------------------------------------
// MV0
//----------------------------------------------------------------------------
int mv0(const char *file, const char *copy) {
    // Pre: el archivo destino no existe.
    char newPath[256], oldPath[256];

    size_t newSize = strlen("./") + strlen(copy) + 1;
    snprintf(newPath, newSize,"./%s", copy);

    size_t oldSize = strlen("./") + strlen(file) + 1;
    snprintf(oldPath, oldSize,"./%s", file);

    int value = rename(oldPath, newPath);
    if (value == -1) {
        perror("ERROR with function rename()");
        return 1;
    }
    return 0;
}
//--------------------------------------------------------------------------------
// MAIN
//--------------------------------------------------------------------------------
int main(int argc, char* argv[]) {
    return mv0(argv[1], argv[2]);
}
//--------------------------------------------------------------------------------
```

cp0

```
#define _POSIX_C_SOURCE 200809L
#include <fcntl.h>
#include <unistd.h>
#include <sys/stat.h>
#include "functions.h"
//--------------------------------------------------------------------------------
// CP0
//--------------------------------------------------------------------------------
int cp0(const char* file, const char* copy) {
    // Pre: el archivo de origen existe y es regular.
    // El archivo destino no existe.
    int fdFile = open(file, O_RDONLY);
    int mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;  // 0644
    int fdCopy = open(copy, O_WRONLY|O_CREAT, mode);
    ssize_t read = 1, written = 1;
    char buffer[256];
    while (read != 0 && written != 0) {
        read = readArchive(fdFile, buffer, 256);
        written = writeArchive(fdCopy, buffer, (size_t) read);
        if (written == -1 || read == -1) return 1;
    }
    if (close(fdFile) == -1 || close(fdCopy) == -1) return 1;
    return 0;
}
//--------------------------------------------------------------------------------
// MAIN
//--------------------------------------------------------------------------------
int main(int argc, char* argv[]) {
    return cp0(argv[1], argv[2]);
}
//--------------------------------------------------------------------------------
```

touch1

```
#define _POSIX_C_SOURCE 200809L
#include <unistd.h>
#include <fcntl.h>
#include <utime.h>
#include <stdio.h>

//--------------------------------------------------------------------------------
// TOUCH1
//--------------------------------------------------------------------------------
int touch1(const char* file) {
    //Pre: si el archivo existe, es un archivo regular.
    int fd = open(file, O_WRONLY);
    if (fd == -1) {
        fd = open(file, O_CREAT, S_IRWXU);
    }
    else {
        int value = utime(file, NULL);
        if (value == -1) {
            perror("ERROR at function touch0 with utime");
            return 1;
        }
    }
    if (close(fd) == -1) return 1;
    return 0;
}
//--------------------------------------------------------------------------------
// MAIN
//--------------------------------------------------------------------------------
```

```
int main(int argc, char* argv[]) {
    return touch1(argv[1]);
}
//--------------------------------------------------------------------------------
```

ln1

```
#define _POSIX_C_SOURCE 200809L
#include <unistd.h>
#include <stdio.h>
//--------------------------------------------------------------------------------
// LN1
//--------------------------------------------------------------------------------
int ln1(const char *target, const char *linkPath) {
    // Pre: no existe un archivo con el nombre del enlace.
    int value =  link(target, linkPath);
    if (value == -1) {
        perror("");
        return 1;
    }
    return 0;
}
//--------------------------------------------------------------------------------
// MAIN
//--------------------------------------------------------------------------------
int main(int argc, char* argv[]) {
    return ln1(argv[1], argv[2]);
}
//--------------------------------------------------------------------------------
```

tee0

```
#define _POSIX_C_SOURCE 200809L
#include <unistd.h>
#include <stdio.h>
#include <fcntl.h>
#include "functions.h"
//--------------------------------------------------------------------------------
// TEE0
//--------------------------------------------------------------------------------
int tee0(const char* file) {
    // Pre: el archivo o bien no existe, o bien es un archivo regular.
    char buffer[256];
    ssize_t read = 1, writtenSTD = 1, written = 1;
    int fdIn = fileno(stdin);
    int fdOut = fileno(stdout);
    int fdFile = open(file, O_WRONLY);
    if (fdFile == -1) {
        fdFile = open(file, O_CREAT, S_IRWXU);
        fdFile = open(file, O_WRONLY);
    }
    while (read != 0 && written != 0 && writtenSTD != 0) {
        read = readArchive(fdIn, buffer, 256);
        written = writeArchive(fdFile, buffer, (size_t) read);
        writtenSTD = writeArchive(fdOut, buffer, (size_t) read);
        if (read == -1 || written == -1 || writtenSTD == -1) {
            perror("Error at function tee0");
            return 1;
        }
    }
    if (close(fdIn) == -1 || close(fdFile) == -1 || close(fdOut) == -1) {
        return 1;
    }
    return 0;
}
//--------------------------------------------------------------------------------
// MAIN
//--------------------------------------------------------------------------------
int main(int argc, char* argv[]) {
    return tee0(argv[1]);
}
//--------------------------------------------------------------------------------
```

ls0

```
#define _POSIX_C_SOURCE 200809L
#include <dirent.h>
#include <stdio.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <errno.h>
#include "functions.h"
//--------------------------------------------------------------------------------
// WRITE ARCHIVES
//--------------------------------------------------------------------------------
```

```
int writeFunctions(const char* dir, const char* father) {
    struct stat buffer;
    int value = stat(dir, &buffer);
    if (value == -1) {
        perror("ERROR with function stat()");
        return 1;
    }
    if ((buffer.st_mode & S_IFMT) != S_IFREG) return 0;
    size_t size = strlen(dir);
    writeArchive(STDOUT_FILENO, (void*) dir, size);
    writeArchive(STDOUT_FILENO, "\n", 2);
    return 0;
}
//--------------------------------------------------------------------------------
// LS0
//--------------------------------------------------------------------------------
int ls0() {
    return walk("./", writeFunctions);
}
//--------------------------------------------------------------------------------
// MAIN
//--------------------------------------------------------------------------------
int main(int argc, char* argv[]) {
    return ls0();
}
//--------------------------------------------------------------------------------
```

cp1

```
#define _POSIX_C_SOURCE 200809L
#include <fcntl.h>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <string.h>
#include <stdio.h>
//--------------------------------------------------------------------------------
// FILE SIZE
//--------------------------------------------------------------------------------
size_t fileSize(const char* fileName) {
    struct stat st;
    stat(fileName, &st);
    return (size_t) st.st_size;
}
```

```
//-----------------------------------------------------------------------------
// CP1
//-----------------------------------------------------------------------------
int cp1(const char* file, const char* copy) {
    // Pre: el archivo de origen existe y es regular.
    // El archivo destino no existe.
    int fdFile = open(file, O_RDONLY);
    int mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;  // 0644
    int fdCopy = open(copy, O_RDWR | O_CREAT, mode);
    if (fdCopy == -1) {
        perror("ERROR: file copy at function open");
        return 1;
    }

    size_t length = fileSize(file);
    int value = truncate(copy, length);
    if (value == -1) {
        perror("ERROR: truncate size of file copy");
        return 1;
    }
    if (length == 0) {
        perror("ERROR: File is empty, nothing to do");
        return 1;
    }
    void *mappedAreaFile;
    void *mappedAreaCopy;

    mappedAreaFile = mmap(0, length, PROT_READ, MAP_SHARED, fdFile, 0);
    if (mappedAreaFile == MAP_FAILED) {
        perror("Error mmapping the input file");
        return 1;
    }

    mappedAreaCopy = mmap(0, length, PROT_WRITE, MAP_SHARED, fdCopy, 0);
    if (mappedAreaCopy == MAP_FAILED) {
        perror("Error mmapping the output file");
        return 1;
    }

    memcpy(mappedAreaCopy, mappedAreaFile, length);

    if (close(fdFile) == -1 || close(fdCopy) == -1) return 1;
    return 0;
}
//-----------------------------------------------------------------------------
// MAIN
```

```
//------------------------------------------------------------------------------
int main(int argc, char* argv[]) {
    return cp1(argv[1], argv[2]);
}
//------------------------------------------------------------------------------
```

ps0

```
#include <dirent.h>
#include <stdio.h>
#include <errno.h>
#include <ctype.h>
#include <stdbool.h>
#include <string.h>
#include <fcntl.h>
#include "functions.h"
//------------------------------------------------------------------------------
// CON CAT DIR
//------------------------------------------------------------------------------
void concatDir(char* path, const char* dir, const char* father) {
    size_t size = strlen(dir) + strlen(father) + strlen("/") + 1;
    snprintf(path, size, "%s/%s", father, dir);
}
//------------------------------------------------------------------------------
// GET FATHER
//------------------------------------------------------------------------------
void getFather(char* father, const char* path) {
    char s = '0';
    size_t pos = strlen(path) - 1;
    size_t size = strlen(path) - 1;
    while (s != '/') {
        father[size - pos] = path[pos];
        pos--;
        s = path[pos];
    }
    father[size - pos] = '\0';
}
//------------------------------------------------------------------------------
// PRINT PROC
//------------------------------------------------------------------------------
int printProc(const char* file, const char* father) {
    if (strncmp(file, "comm", 4) != 0) return 0;
    char path[256];
    concatDir(path, file, father);
    int fd = open(path, O_RDONLY);
    if (fd == -1) {
        perror("ERROR at opening file in function printProc()");
        return 1;
    }
    char proc[256];
    ssize_t read = readArchive(fd, proc, 256);
    proc[read] = '\0';
    if (read == -1) return 1;
    char msg[256];
    char pid[5];
    getFather(pid, father);
    size_t size = strlen(pid) + (read) + strlen(" ") + 1;
    snprintf(msg, size, "%s %s\n", pid, proc);
    writeArchive(STDOUT_FILENO, msg, size);
    if (close(fd) == -1) return 1;
    return 0;
}
//--------------------------------------------------------------------------------
// IS PID
//--------------------------------------------------------------------------------
int isPid(const char* pid, const char* father) {
    size_t size = strlen(pid);
    for (size_t i = 0; i < size; i++) {
        if (!isdigit(pid[i])) return 0;
    }
    char path[256];
    concatDir(path, pid, father);
    return walk(path, printProc);
}
//--------------------------------------------------------------------------------
// PS0
//--------------------------------------------------------------------------------
int ps0() {
    return walk("/proc", isPid);
}
//--------------------------------------------------------------------------------
// MAIN
//--------------------------------------------------------------------------------
int main(int argc, char* argv[]) {
    return ps0();
}
//--------------------------------------------------------------------------------
```

functions.h

```
#ifndef LAB1_SYSCALLS_H
#define LAB1_SYSCALLS_H
//-----------------------------------------------------------------------------
// INCLUDES
//-----------------------------------------------------------------------------
#include <unistd.h>
#include <dirent.h>
#define ERROR 1
#define SUCCESS 0
#define BUF_LEN 256
//-----------------------------------------------------------------------------
// READ ARCHIVE
//-----------------------------------------------------------------------------
ssize_t readArchive(int fd, void *buf, size_t bytes);
//-----------------------------------------------------------------------------
// WRITE ARCHIVE
//-----------------------------------------------------------------------------
ssize_t writeArchive(int fd, void *buf, size_t bytes);
//-----------------------------------------------------------------------------
// IS ERROR (PUNTEROS)
//-----------------------------------------------------------------------------
void perr(const char *format, ...);
//-----------------------------------------------------------------------------
// NEXT
//-----------------------------------------------------------------------------
size_t next(struct dirent** direntStructure, DIR* directoryStream);
//-----------------------------------------------------------------------------
// WALK
//-----------------------------------------------------------------------------
int walk(const char *dir, int (*f)(const char* a, const char* father));
//-----------------------------------------------------------------------------
#endif // LAB1_SYSCALLS_H
```

functions.c

```
#include "functions.h"
#include <errno.h>
#include <stdio.h>
#include <stdarg.h>
//-----------------------------------------------------------------------------
// READ ARCHIVE
//-----------------------------------------------------------------------------
```

```
ssize_t readArchive(int fd, void *buf, size_t bytes) {
    size_t bytesRead = 0;
    ssize_t value;
    while (bytesRead < bytes) {
        value = read(fd, buf+bytesRead, bytes - bytesRead);
        if (value == -1) {
            perror("ERROR at function readArchive");
            return value;
        }
        if (value == 0) {
            return bytesRead;
        }
        bytesRead += value;
    }
    return bytesRead;
}
//--------------------------------------------------------------------------------
// WRITE ARCHIVE
//--------------------------------------------------------------------------------
ssize_t writeArchive(int fd, void *buf, size_t bytes) {
    size_t bytesWritten = 0;
    ssize_t value;
    while (bytesWritten < bytes) {
        value = write(fd, buf+bytesWritten, bytes - bytesWritten);
        if (value == -1) {
            perror("ERROR at function writeArchive");
            return value;
        }
        if (value == 0) {
            return bytesWritten;
        }
        bytesWritten += value;
    }
    return bytesWritten;
}
//--------------------------------------------------------------------------------
// IS ERROR
//--------------------------------------------------------------------------------
void perr(const char *format, ...) {
    va_list args;
    va_start(args, format);
    char msgError[BUF_LEN];
    vsnprintf(msgError, BUF_LEN, format, args);
    va_end(args);
    perror(msgError);
}
```

```
//-------------------------------------------------------------------------------
// NEXT
//-------------------------------------------------------------------------------
size_t next(struct dirent** direntStructure, DIR* directoryStream) {
    errno = 0;
    (*direntStructure) = readdir(directoryStream);
    if (*direntStructure == NULL && errno != 0) {
        perr("ERROR in function readdir()");
        return 1;
    }
    return 0;
}
//-------------------------------------------------------------------------------
// WALK
//-------------------------------------------------------------------------------
int walk(const char *dir, int (*f)(const char* a, const char* father)) {
    DIR* directoryStream = opendir(dir);
    if (directoryStream == NULL) {
        perr("ERROR with dir: %s in function opendir()", dir);
        return 1;
    }
    struct dirent* direntStructure;
    if(next(&direntStructure, directoryStream) == 1) return 1;
    while (direntStructure != NULL) {
        if (f(direntStructure->d_name, dir) == 1) return 1;
        if(next(&direntStructure, directoryStream) == 1) return 1;
    }
    if (closedir(directoryStream) == -1) {
        perror("ERROR WITH closedir");
        return 1;
    }
    return 0;
}
//-------------------------------------------------------------------------------
```