Sistemas Operativos (75.08): Lab UNIX

Matias Rozanec (97404)
rozanecm@gmail.com

Abril 2018

Índice

Parte I
Consigna

Lab Unix

El objetivo de este lab es tener un primer contacto con la interfaz del kernel mediante syscalls tradicionales de Unix, en particular las relacionadas al manejo de archivos.

El lab consiste en la implementación de versiones simplificadas de las herramientas Unix más comunes provistas en cualquier distribución: *ls*, *cat*, *rm*, *cp*, *mv*, etc.

Enlaces recomendados:

- el régimen de cursada, donde se explica la notación /
- las instrucciones de entrega en papel.

Syscalls UNIX

Se debe implementar cada ejercicio usando las syscalls Unix apropiadas, evitando el uso de las "funciones de alto nivel" que proporciona la biblioteca estándar de C. Así, por ejemplo, para la apertura de archivos se debe usar la syscall `open(2)`, y no la función `fopen(3)`.

Sí se permite el uso de funciones de la biblioteca estándar para trabajar con strings y para mostrar información por pantalla. Así, por ejemplo, para escribir en la consola se puede usar `printf(3)` en lugar de `write(2)`.

Tanto en el caso de syscalls, como funciones, se puede consultar su documentación mediante el comando `man`. Esto es particularlmente recomendable en el caso de syscalls como `stat(2)`, que son complejas y tienen muchos flags: `man 2 stat`. En las páginas de manual también se indican los includes necesarios para cada syscall.

En cada ejercicio se indica la lista de syscalls recomendadas. Como cada ejercicio emula una herramienta estándar de Unix, se puede obtener una descripción de la funcionalidad completa también en las páginas del man (e.g. `man 1 cat`).

Esqueleto y flags de compilación

El siguiente esqueleto de un comando que acepta un único parámetro puede usarse a modo de ejemplo para cualquiera de las implementaciones:

```
#define _POSIX_C_SOURCE 200809L

#include <...>

void rm0(const char *file) {
    // ...
```

```
}

int main(int argc, char *argv[]) {
    rm0(argv[1]);
}
```

Se recomienda compilar utilizando los flags `-std=c11 -Wall -Wextra -g`.

Parte 1

rm0

rm (*remove*) es la herramienta unix que permite eliminar archivos y directorios.

El uso estándar `rm <file>` permite borrar solo archivos regulares, y arrojará error si se intenta eliminar un directorio.

Para la implementación de *rm0* solo se considerará el caso de archivos regulares.

```
$ ls
archivo1   archivo2   directorio1   rm0
$ ./rm0 archivo1
$ ls
archivo2   directorio1   rm0
```

Se pide: implementar *rm0* que elimina un archivo regular.

Pre-condición: el archivo existe y es regular.

Syscalls recomendadas: unlink.

cat0

cat (*concatenate*) es una herramienta unix que permite concatenar archivos y mostrarlos por salida estándar. En este lab se implementará una vesión simplificada de cat, que muestra en pantalla los contenidos de un único archivo.

```
$ cat ejemplo.txt
Sistemas Operativos, 1er cuatrimestre 2018
```

Se pide: Implementar *cat0* que toma un archivo regular y muestra su contenido por salida estándar.

Pre-condición: solo se pasa un archivo, este archivo existe y se tienen permisos de lectura.

Syscalls recomendadas: open, read, write, close.

touch0

touch toma como parámetro un archivo (de cualquier tipo) y permite actualizar su metadata, especialmente las fechas de último acceso (*atime*) y última modificación (*mtime*); ambos atributos pueden verse mediante el comando *stat*. Una llamada a touch sobre un archivo actualiza ambas fechas al tiempo actual.

No obstante, el uso más común del comando touch es la creación de archivos regulares: si el parámetro referencia a un archivo que no existe, se lo crea. La primera versión *touch0* sólo implementará esta funcionalidad de touch.

Se pide: Implementar *touch0* que toma como parámetro un archivo y lo crea en caso de que no exista (el archivo creado debe estar en blanco). Si el archivo ya existía, no se hace nada.

Ejemplo:

```
$ ls
touch0
$ ./touch0 un_archivo
$ ls
touch0 un_archivo
$ stat un_archivo
  File: un_archivo
  Size: 0     Blocks: 0    IO Block: 4096   regular empty file
[...]
```

Notar que el tamaño del archivo creado es 0, y stat también nos lo indica enunciando *regular empty file*.

Pre-condición: si el archivo existe, es un archivo regular.

Syscalls recomendadas: open.

stat0

stat muestra en pantalla los metadatos de un archivo, incluyendo información sobre tipo de archivo, fechas de creación y modificación, permisos, etc.

```
$ stat README.md
  File: README.md
  Size: 1318       Blocks: 8          IO Block: 4096   regular file
Device: 806h/2054d  Inode: 2753812     Links: 1
Access: (0644/-rw-r--r--) Uid: ( 1000/   juan)  Gid: ( 1000/   juan)
Access: 2018-03-14 17:36:37.497432618 -0300
Modify: 2018-03-08 23:27:15.765147109 -0300
Change: 2018-03-08 23:27:15.765147109 -0300
 Birth: -
```

La implementación de *stat0* mostrará únicamente el nombre, tipo y tamaño del archivo (en bytes).

```
$ ./stat0 README.md
Size: 1318
File: README.md
Type: regular file
```

Se pide: Implementar *stat0* que muestra el nombre, tipo y tamaño en bytes de un archivo regular o directorio.

Pre-condición: el archivo existe, y es un directorio o un archivo regular.

Syscalls recomendadas: stat. Se puede consultar también la página de manual `inode(7)`.

rm1

Mostrar cómo se usaría `errno` y `perror(3)` para obtener el siguiente comportamiento de *rm*:

```
$ ./rm1 directorio1
rm: cannot remove 'directorio1': Is a directory
```

Parte 2

ln0

ln (*link*) permite la creación de enlaces a archivos, tanto "hard links" como "soft links". Por defecto el uso de *ln* toma dos parámetros, el objetivo del link y el nombre; y crea un hard link. Puede usarse el flag `-s` (o *–symbolic*) para crear enlaces simbólicos ("soft links").

La implementación de *ln0* replicará el comportamiento de `ln -s`, permitiendo crear solamente enlaces simbólicos. Un uso de *ln0* podría ser:

```
$ ls -al
[...]
-rw-r--r-- 1 juan juan    0 Mar 21 18:39 archivo1
-rw-r--r-- 1 juan juan    0 Mar 21 18:39 ln0

$ ./ln0 archivo1 enlace

$ ls -al
-rw-r--r-- 1 juan juan    0 Mar 21 18:39 archivo1
lrwxrwxrwx 1 juan juan    8 Mar 21 18:40 enlace -> archivo1
-rw-r--r-- 1 juan juan    0 Mar 21 18:39 ln0
```

Se pide: Implementar *ln0* que permite crear enlaces simbólicos.

Pre-condición: no existe un archivo con el nombre del enlace.

Syscalls recomendadas: symlink.

Pregunta: ¿Qué ocurre si se intenta crear un enlace a un archivo que no existe?

mv0

mv (*move*) permite mover un archivo (regular o directorio) de un directorio a otro. El archivo no se mueve físicamente sino que sólo se renombra y se modifica el enlace al mismo en su directorio actual. La implementación de *mv0* tendrá la misma funcionalidad que *mv*.

Un ejemplo del uso de *mv0* podría ser:

```
$ ls
directorio1   archivo1
$ ls directorio1
$ mv archivo1 directorio1/archivo2
$ ls
directorio1
$ ls directorio1
archivo2
```

Se pide: Implementar *mv0*, que permite mover un archivo de un directorio a otro.

Pre-condición: el archivo destino no existe.

Syscalls recomendadas: rename.

Pregunta: ¿se puede usar *mv0* para renombrar archivos dentro del mismo directorio?

cp0

cp (*copy*) es el comando de Unix que permite copiar archivos. La sintaxis toma como parámetros dos archivos regulares, y copia los contenidos del primero al segundo. Si el segundo archivo no existe, es creado; y si ya existía, sus contenidos se sobreescriben.

El uso de *cp* sin flags adicionales sólo permite copiar archivos regulares, aunque puede especificarse el flag `-r` para copiar directorios de manera recursiva. La implementación de *cp0* sólo tendrá en cuenta el caso de copiar archivos regulares.

Ejemplo:

```
$ cat archivo1
Sistemas Operativos, 1er cuatrimestre 2018
$ ls
archivo1 cp0
$ ./cp0 archivo1 archivo2
$ ls
archivo1 archivo2 cp0
$ cat archivo2
Sistemas Operativos, 1er cuatrimestre 2018
```

Se pide: Implementar *cp0* que copia los contenidos de un archivo a otro. Utilizar para esta implementación las syscalls básicas de entrada y salida, esto es: open(2), read(2), write(2) y close(2).

Pre-condición: el archivo de origen existe y es regular. El archivo destino no existe.

touch1

Revisitar la implementación de *touch0* realizada en los puntos anteriores y agregarle la funcionalidad de actualización de las fechas en la metadata de un archivo en caso de que ya exista. Tal metadata puede verse mediante el comando *stat*.

Un ejemplo del uso de *touch1* sería:

```
$ stat archivo
[...]
Access: 2018-03-16 17:31:48.722017895 -0300
Modify: 2018-03-14 17:27:56.438147960 -0300
Change: 2018-03-14 17:27:56.438147960 -0300
[...]
$ ./touch0 archivo
$ stat archivo
[...]
Access: 2018-03-21 00:58:04.671902112 -0300
Modify: 2018-03-21 00:58:04.671902112 -0300
Change: 2018-03-21 00:58:04.671902112 -0300
[...]
```

Se actualizaron todas las fechas asociadas al archivo, pero los contenidos del mismo no se vieron modificados, sólo se alteró la metadata.

Implementar *touch1* que toma un archivo como parámetro. Si no existe, crea un archivo regular vacío. Si el archivo existía previamente, modifica las fechas de acceso y de modificación al tiempo actual.

Syscalls recomendadas: utime(2) y la función futimes(3).

ln1

Implementar una versión *ln1* que cree hard links en lugar de soft links. Hacer uso de la syscall *link(2)*. Luego, teniendo las dos versiones, responder las siguientes preguntas:

¿Cuál es la diferencia entre un hard link y un soft link?

Crear un hard link a un archivo, luego eliminar el archivo original ¿Qué pasa con el enlace? ¿Se perdieron los datos del archivo?

Repetir lo mismo, pero con un soft link. ¿Qué pasa ahora con el enlace? ¿Se perdieron los datos esta vez?

Explicar las diferencias.

Syscalls recomendadas: link(2), symlink(2)

Parte 3

tee0

tee (conector T) toma como parámetro un archivo, y escribe todo lo que llega por entrada estándar, tanto en la salida estándar como al archivo. Resulta muy útil cuando se quiere ver el resultado de la ejecución de un programa y a su vez guardar una copia de todo lo que escriba en un archivo.

Un ejemplo del uso de `tee` podría ser:

```
$ echo "Hola" | tee dump.txt
Hola
$ cat dump.txt
Hola
```

Por defecto `tee` crea el archivo si no lo encuentra, y lo sobreescribe (trunca) si ya existía. La implementación estándar de tee tiene muchas más opciones que pueden consultarse en el man (*tee(1)*).

Implementar *tee0* que transcribe la entrada estándar tanto en la salida estándar como en el archivo especificado.

Pre-condición: el archivo o bien no existe, o bien es un archivo regular.

ls0

ls (*list*) lista los contenidos del directorio que se le pase por parámetro. Si no se especifica ningún parámetro, ls muestra el contenido de los archivos en el directorio actual (ver *pwd(1)*).

El comando ls admite una gran variedad de flags para elegir qué información se mostrará de los archivos, con qué formato y orden. La implementación de ls0 se corresponderá con `ls -U1`, o lo que es equivalente `ls --format=single-column --sort=none`, que lista únciamente los nombres de los archivos, sin ningún ordenamiento particular y de a uno por línea.

Por ejemplo:

```
$ ls
archivo1   archivo2   archivo3   ls0
$ ./ls0
archivo2
archivo1
ls0
archivo3
```

Se pide: Implementar *ls0* que lista todos los archivos en el directorio actual, uno en cada línea. No hay que preocuparse por el orden en que se listen los archivos, con que se muestren todos es suficiente.

Funciones recomendadas: stat(2), opendir(3), readdir(3), closedir(3).

cp1

La syscall *mmap* permite mapear una región de los contenidos de un archivo a memoria, y acceder a los mismos directamente como si fuera un array de bytes. Si se utilizan los flags apropiados (`MAP_SHARED`, ver mmap(2)) los cambios en la memoria correspondiente al archivo se verán reflejados en el mismo.

Si bien el uso de las syscall de entrada y salida básicas es la implementación más común para *cp*, también es posible utilizar *mmap* para copiar archivos. La idea es crear un archivo nuevo, y mapear tanto el archivo origen como el destino a *regiones de memoria distintas*, luego copiar los datos en memoria de una región a la otra (por ejemplo, utilizando *memcpy*).

Notar que esta implementación requiere tener tanta memoria ram como el doble del tamaño del archivo a copiar. Para mitigar esto, se puede optimizar mapeando en memoria bloques de tamaño fijo de cada archivo, uno a la vez.

Implementar *cp1*, que debe tener la misma funcionalidad que *cp0* pero implementada mediante *mmap* y *memcpy*.

Syscalls recomendadas: mmap(2), memcpy(2), open(2)

ps0

ps (*proccess status*) es un comando unix que permite obtener todo tipo de información acerca de los procesos que están corriendo actualmente, disponiendo de

muchos flags que alteran la cantidad de información a mostrar. Ver *ps(1)* para la lista completa de flags.

Toda esta información se obtiene del pseudo-filesystem `/proc`, que mantiene acceso de sólo lectura a muchas estructuras de control del kernel relacionadas con procesos. En particular, los datos de cada proceso se encuentran en el subdirectorio `/proc/[pid]`, siendo *pid* el proccess ID del proceso.

Dentro de `/proc/[pid]` hay información exhaustiva sobre cada proceso. Para este ejercicio nos interesa en particular `/proc/[pid]/comm`, que guarda el nombre del programa que se usó para lanzar el proceso. Para tener una descripción exacta de qué guarda cada archivo en `/proc` y cómo está codificado, referirse a *proc(5)*.

La implementación de *ps0* (mucho más humilde), sólo listará para cada proceso su *pid* y el nombre del binario ejecutable que está corriendo. Para lograrlo hay que recorrer el directorio `/proc` y recaudar la información importante.

La salida de *ps0* equivale a ejecutar `ps -eo pid,comm`, que lista en dos columnas el process id y el comando de todos los procesos. Un ejemplo de esta salida sería:

```
$ ps -eo pid,comm
    1 systemd
    2 kthreadd
    3 ksoftirqd/0
    5 kworker/0:0H
    7 rcu_sched
    8 rcu_bh
[...]
 7531 bash
 7625 kworker/1:0
 8046 ps
```

Implementar *ps0* que debe mostrar la misma información que `ps -eo pid,comm`.

Syscalls recomendadas: opendir(3), readdir(3)

Ayudas: proc(5), isdigit(3), para corroborar que se esté accediendo al directorio de un proceso y no a algún otro archivo de `/proc`.

Challenge del challenge: dar más información del estado de un proceso a través de `/proc/[pid]/stat`, tomar de *proc(5)* el formato del archivo y ayudarse de *scanf(3)* para realizar el parseo.

Parte II
Resolución

1. Parte 1

1.1. rm0

```c
#include <unistd.h>

int main(int argc, char* argv[]){
    /* Implementar rm0 que elimina un archivo regular.
     * Pre-condicion: el archivo existe y es regular. */

    /* check num of args */
    if(argc != 2){
        write(1, "Incorrect number of args received.\n", 35);
        _exit(1);
    }
    unlink(argv[1]);
}
```

1.2. cat0

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

void my_write(void* buf, short count){
    /* On program startup, the integer file descriptors associated with the
     * streams stdin, stdout, and stderr are 0, 1, and 2, respectively.
     * The preprocessor symbols STDIN_FILENO, STDOUT_FILENO, and STDERR_FILENO
     * are defined with these values in <unistd.h>.
     * (Applying freopen(3) to one of these streams can change the file
     * descriptor number associated with the stream.)
     * */

    ssize_t num_of_written_bytes = write(STDIN_FILENO, buf, count);
    while(num_of_written_bytes < count){
        num_of_written_bytes += write(STDIN_FILENO, buf + num_of_written_bytes,
                                      count - num_of_written_bytes);
    }
}

int main(int argc, char* argv[]){
    /* Pre-condicion: solo se pasa un archivo,
     * y el archivo existe y tiene permisos de lectura */

    // get file descriptor for file in path
    int fd = open(argv[1], O_RDONLY);

    // read file
    //read by chunks of size 1KB
    size_t count = 1024;
    char* buf[count];
    short num_of_read_bytes = read(fd, buf, count);
    while(num_of_read_bytes != 0){
        my_write(buf, num_of_read_bytes);
        num_of_read_bytes = read(fd, buf, count);
    }
    close(fd);
}
```

1.3. touch0

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

int main(int argc, char* argv[]){
    if (argc != 2){
        write(1, "Incorrect number of args received.\n", 35);
        _exit(1);
    }
    /* 436 is the bit mask which gives the following permissions:
     *      USER:   read, write
     *      GROUP:  read, write
     *      OTHERS: read
```

```
     * */
    int fd = open(argv[1], O_CREAT, 436);
    close(fd);
}
```

1.4. stat0

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

ssize_t my_strlen(char* buf){
    ssize_t size = 0;
    while(buf[size] != '\0'){
        size++;
    }
    return size;
}

void my_write(void* buf, short count){
        /* On program startup, the integer file descriptors associated with the
         * streams stdin, stdout, and stderr are 0, 1, and 2, respectively.
         * The preprocessor symbols STDIN_FILENO, STDOUT_FILENO, and STDERR_FILENO
         * are defined with these values in <unistd.h>.
         * (Applying freopen(3) to one of these streams can change the file
         * descriptor number associated with the stream.)
         * */

        ssize_t num_of_written_bytes = write(STDIN_FILENO, buf, count);
        while(num_of_written_bytes < count){
            num_of_written_bytes += write(STDIN_FILENO, buf + num_of_written_bytes,
                                          count - num_of_written_bytes);
        }
}

void my_reverse_string(char* buf, int len){
    ssize_t i = 0;
    char c;
    while(i <= len/2){
        c = buf[i];
        buf[i] = buf[len-i-1];
        buf[len-i-1] = c;
        ++i;
    }
}


void uint_to_string(unsigned int src, char* dest){
    int i = 0;
    while(src > 0){
        unsigned int last_digit = src%10;
        /* 0 is 48 in ASCII. All numbers are ordered, so the corresponding ASCII
         * code of a number is the number + 48.
         * */
        unsigned int temp = last_digit + 48;
        dest[i] = (char)temp;
        src /= 10;
        ++i;
    }
    dest[i] = '\0';
    my_reverse_string(dest, i);
}

int main(int argc, char* argv[]){
    /* La implementación de stat0 mostrará únicamente el nombre,
     * tipo y tamaño del archivo (en bytes).
     *
     * Implementar stat0 que muestra el nombre, tipo y tamaño en bytes
     * de un archivo regular o directorio.
     *
     * Pre-condición: el archivo existe, y es un directorio o un archivo regular.
     * */

    /* First, show file's name */

    char* out = "File name:\t\t\t";

    my_write(out, my_strlen(out));
    my_write(argv[1], my_strlen(argv[1]));
    my_write("\n", 1);

    /* Now show file's type */

    struct stat buf;
    stat(argv[1], &buf);

    out = "File type:\t\t\t";
    my_write(out, my_strlen(out));

```

```
    if(S_ISDIR(buf.st_mode)){
        out = "Directory\n";
        my_write(out, my_strlen(out));
    }else if(S_ISREG(buf.st_mode)){
        out = "Regular file\n";
        my_write(out, my_strlen(out));
    }

    out = "File size (in bytes):\t\t";
    my_write(out, my_strlen(out));
    int size = buf.st_size;
    /* We won't need more than five chars to write the number */
    char c_size[] = "aaaaaa";
    uint_to_string(size, c_size);
    my_write(c_size, my_strlen(c_size));
    my_write("\n", 1);
}
```

1.5. rm1

```
#define _POSIX_C_SOURCE 200809L

#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <errno.h>

int main(int argc, char* argv[]){
    /* check number of args received. */
    if(argc != 2){
        write(1, "file to remove should be passed as arg.\n", 40);
    }

    struct stat buf;
    if(stat(argv[1], &buf) == -1){
        write(1, "An error occurred retrieving information from path.\n", 51);
        _exit(1);
    }

    if(buf.st_mode == S_IFDIR){
        /* print error msg */
        perror("cannot remove ");
        perror(argv[1]);
        perror(": Is a directory");

        /* exit with error code */
        _exit(1);
    }

    if(unlink(argv[1]) == -1){
        write(1, "An error occurred removing file.\n", 33);
    }
}
```

2. Parte 2

2.1. ln0

```c
#define _POSIX_C_SOURCE 200809L

#include <unistd.h>

int main(int argc, char* argv[]){
    /* Se pide: Implementar ln0 que permite crear enlaces simbólicos.
     *
     * Pre-condición: no existe un archivo con el nombre del enlace.
     *
     * Syscalls recomendadas: symlink.
     *
     * Pregunta: ¿Qué ocurre si se intenta crear un enlace a un archivo
     * que no existe?
     * Si ejecutamos el comando ll, se puede ver que el enlace fue creado.
     * Sin embargo, hay que tener en cuenta que el enlace creado apunta
     * a un archivo inexistente.
     * */

    /* check for correct number of args */
    if (argc != 3){
        write(1, "2 args must be passed.\n", 23);
        _exit(1);
    }

    symlink(argv[1], argv[2]);
}
```

2.2. mv0

```c
#include <stdio.h>

int main(int argc, char* argv[]){
    /* Se pide: Implementar mv0, que permite mover un archivo
     * de un directorio a otro.
     *
     * Pre-condición: el archivo destino no existe.
     *
     * Syscalls recomendadas: rename.
     *
     * Pregunta: ¿se puede usar mv0 para renombrar archivos
     * dentro del mismo directorio?
     * Si. Se ha probado creando un archivo y luego renombrandolo.
     * */
    rename(argv[1], argv[2]);
}
```

2.3. cp0

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

void my_write(int fd, const void *buf, size_t count){
    size_t bytes_written = 0;
    while(bytes_written < count){
        bytes_written += write(fd, buf+bytes_written, count-bytes_written);
    }
}

int main(int argc, char*argv[]){
    /* Se pide: Implementar cp0 que copia los contenidos de un archivo a otro.
     * Utilizar para esta implementación las syscalls básicas de entrada y
     * salida, esto es: open(2), read(2), write(2) y close(2).
     *
     * Pre-condición: el archivo de origen existe y es regular. El archivo
     * destino no existe.
     * */

    /* check for correct num. of args. */
    if(argc != 3){
        write(1, "ERROR! Se deben recibir dos args.!\n", 35);
        _exit(1);
    }

    /* open files */
    int fd_origin = open(argv[1], O_RDONLY);
    int fd_des = open(argv[2], O_CREAT, 436);
    /* as newly created file can't have data written on it, we must close it
     * and re open it again.
```

```
     * */
    close(fd_des);

    /* open destination file again, now as write only */
    fd_des = open(argv[2], O_WRONLY);

    /* copy */
    int chunk_size = 1024;      //read by 1Kb chunks
    char buf[chunk_size];
    int read_bytes;
    while((read_bytes = read(fd_origin, &buf, chunk_size)) > 0){
       my_write(fd_des, &buf, read_bytes);
    }

    /*close files */
    close(fd_origin);
    close(fd_des);
}
```

2.4. touch1

```
#define _POSIX_C_SOURCE 200809L

#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <errno.h>
#include <utime.h>

int main(int argc, char* argv[]){
    if (argc != 2){
        write(1, "Incorrect num. or args. received.\n", 34);
        _exit(1);
    }

    /* 436 is the bit mask which gives the following permissions:
     *      USER:   read, write
     *      GROUP:  read, write
     *      OTHERS: read
     * */
    int fd = open(argv[1], O_CREAT | O_EXCL, 436);
    if(fd == -1 && errno == EEXIST){
        /* file already exists */
        fd = open(argv[1], O_RDWR);
        if(fd == -1){
            write(1, "Path specified indicates no file, and an error "
                    "occurred creating the file.\n", 75);
            _exit(1);
        }

        if(utime(argv[1], NULL) == -1){
            write(1, "An error occurred during the operation.\n", 40);
        }
    }
    close(fd);
}
```

2.5. ln1

```
#define _POSIX_C_SOURCE 200809L

#include <unistd.h>

int main(int argc, char* argv[]){
    /* check num of args */
    if(argc != 3){
        write(1, "Incorrect number of arguments received.\n", 41);
        _exit(1);
    }

    if(link(argv[1], argv[2]) == -1){
        /* handle error */
        write(1, "An error occurred during operation.\n", 36);
        _exit(1);
    }
}
```

3. Parte 3

3.1. tee0

```c
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>

void my_write(int fd, void* buf, size_t count){
    size_t bytes_written = 0;
    while(bytes_written < count){
        bytes_written += write(fd, buf+bytes_written, count-bytes_written);
    }
}

int main(int argc, char* argv[]){
    /* check num of args */
    if(argc != 2){
        write(1, "Incorrect number or arguments.\n", 31);
        _exit(1);
    }

    /* by default create file. Then open the file with write permissions. */

    /* 436 is the bit mask which gives the following permissions:
     *          USER:   read, write
     *          GROUP:  read, write
     *          OTHERS: read
     * */
    int fd = open(argv[1], O_CREAT | O_EXCL, 436);

    /* check for errors */
    if(fd != -1){
        /* if no error occurred */
        close(fd);
    }

    fd = open(argv[1], O_WRONLY);

    /* From read(2) man page: On success, the number of bytes read is
     * returned (zero indicates end of file) */
    ssize_t bytes_read;

    /* read by 1K chunks */
    int buf_size = 1024;
    char buf[buf_size];

    /* fd = 0 indicates stdin */
    while((bytes_read = read(0, &buf, buf_size)) != 0){
        my_write(fd, &buf, bytes_read);
        my_write(1, &buf, bytes_read);
    }
    close(fd);
}
```

3.2. ls0

```c
#define _POSIX_C_SOURCE 200809L
#define _GNU_SOURCE

#include <dirent.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>     //necessary for free()
#include <sys/stat.h>
#include <errno.h>

void free_res(DIR* dir, int argc, char* path){
    closedir(dir);
    if(argc == 1){
        free(path);
    }
}

int main(int argc, char* argv[]){
    /* evaluate num of args */
    char* path;
    if(argc == 1){
        /* path is current path */
        path = get_current_dir_name();
    }else{
        path = argv[1];
    }

    /* check that given path is a dir */
    //TODO correct verification. Currently not working.
```

```
    /*
    struct stat buf;
    stat(path, &buf);
    if(S_ISDIR(buf.st_mode)){
        write(1, "Given path is not a dir.\n", 25);
        _exit(1);
    }
    */

    DIR* dir_ptr = opendir(path);
    if(dir_ptr == NULL){
        perror("Error occurred opening dir.");
        write(1, "An error occurred during command execution.\n", 44);
        /* exit with error exit code */
        _exit(1);
    }
    struct dirent* read_dir = readdir(dir_ptr);
    int current_errno = errno;
    while(read_dir != NULL){
        if(current_errno != errno){
            /* theres an error */
            write(1, "An error occurred during command execution.\n", 44);
            free_res(dir_ptr, argc, path);
            _exit(1);
        }
        printf("%s\n", read_dir->d_name);
        read_dir = readdir(dir_ptr);
        current_errno = errno;
    }
    free_res(dir_ptr, argc, path);
}
```

3.3. cp1

```
#define _POSIX_C_SOURCE 200809L

#include <sys/mman.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>     //for EXIT codes
#include <sys/fcntl.h>
#include <sys/stat.h>
#include <string.h>
#include <sys/types.h>

int main(int argc, char* argv[]){
    /* check num of args */
    if(argc != 3){
        write(1, "Incorrect number or arguments.\n", 31);
        _exit(EXIT_FAILURE);
    }

    /* open src and dst file */
    int src_fd = open(argv[1], O_RDONLY);
    int dst_fd = open(argv[2], O_CREAT | O_RDWR, 436);

    /* check errors on file opening */
    if((src_fd == -1) || (dst_fd == -1)){
        write(1, "An error occurred opening files.\n", 33);
        if(src_fd != -1)
            close(src_fd);
        if(dst_fd != -1)
            close(dst_fd);
    }

    struct stat file_stat;
    if(stat(argv[1], &file_stat) == -1){
        write(1, "An error occurred during command execution.\n", 44);
        close(src_fd);
        close(dst_fd);
        _exit(EXIT_FAILURE);
    }
    int file_len = file_stat.st_size;

    if(ftruncate(dst_fd, file_len) == -1){
        perror("");
        write(1, "An error occurred during command execution.\n", 44);
        close(src_fd);
        close(dst_fd);
        _exit(EXIT_FAILURE);
    }

    /* get memory areas for both files */
    char* src_mapped_area = mmap(0 /* If addr is NULL, then the kernel chooses
                                    the address at which to create the mapping */,
            /* length */ file_len,
            /* prot */ PROT_READ | PROT_WRITE,
            /* flags */ MAP_PRIVATE,
            /* fd */ src_fd,
```

```
56              /* offset */ 0);
57
58      char* dst_mapped_area = mmap(0 /* If addr is NULL, then the kernel chooses
59                                      the address at which to create the mapping */,
60              /* length */ file_len,
61              /* prot */ PROT_READ | PROT_WRITE,
62              /* flags */ MAP_SHARED,
63              /* fd */ dst_fd,
64              /* offset */ 0);
65
66      /* check errors on mmap */
67      if((src_mapped_area == MAP_FAILED) || (dst_mapped_area == MAP_FAILED)){
68          /* error occurred */
69          perror("error with mmap");
70          write(1, "An error occurred during command execution.\n", 44);
71          _exit(EXIT_FAILURE);
72      }
73
74      /* copy data */
75      memcpy(dst_mapped_area, src_mapped_area, file_len);
76
77      if(((munmap(src_mapped_area, file_len)) == -1) ||
78              (munmap(dst_mapped_area, file_len))){
79          perror("");
80          write(1, "An error occurred during command execution.\n", 44);
81      }
82
83      /* close files */
84      if((close(src_fd) == -1) || (close(dst_fd) == -1)){
85          perror("");
86          write(1, "An error occurred when closing files.\n", 38);
87          _exit(EXIT_FAILURE);
88      }
89  }
```

3.4. ps0

```
1   #define _POSIX_C_SOURCE 200809L
2
3   #include <stdio.h>
4   #include <unistd.h>
5   #include <stdlib.h>
6   #include <dirent.h>
7   #include <sys/types.h>
8   #include <ctype.h>
9   #include <fcntl.h>
10
11  void write_command(const char* path){
12      FILE* f = fopen(path, "r");
13      if(f == NULL){
14          perror("error opening comm: ");
15      }
16      int pid, ppid, pgrp, session, tty_nr, tpgid;
17      char comm[256];
18      unsigned int flags;
19      char state;
20      fscanf(f, "%d %s %c %d %d %d %d %d %u", &pid, comm, &state, &ppid, &pgrp,
21              &session, &tty_nr, &tpgid, &flags);
22
23      printf("%6d %20s %5c %4d %4d %8d %8d %8d %20u\n", pid, comm, state, ppid,
24              pgrp, session, tty_nr, tpgid, flags);
25
26      fclose(f);
27  }
28
29  int main(){
30      DIR* dir_ptr = opendir("/proc");
31      /* error checking */
32      if(dir_ptr == NULL){
33          perror("An error occurred during command execution\n");
34          _exit(EXIT_FAILURE);
35      }
36
37      /* print column headers */
38      printf("%6s %20s %5s %4s %4s %8s %8s %8s %20s\n", "PID", "COMMAND",
39              "STATE", "PPID", "PGRP", "SESSION", "TTY_NR", "PTGID", "FLAGS");
40
41      /* read dir */
42      struct dirent* read_dir = readdir(dir_ptr);
43      while(read_dir != NULL){
44          if(isdigit(*read_dir->d_name)){
45              /* case: dir corresponds to process */
46              char path[267];
47              sprintf(path, "/proc/%s/stat", read_dir->d_name);
48              write_command(path);
49          }
50          read_dir = readdir(dir_ptr);
51      }
52      /* close dir */
```

```
53      if(closedir(dir_ptr) == -1){
54          perror("An error occurred during command execution.\n");
55          _exit(EXIT_FAILURE);
56      }
57  }
```