Lab Unix

Dylan Alvarez, 98225, https://github.com/dylanalvarez/lab-unix-sistemas-operativos
1º cuatrimestre, 2018

Parte 1

rm0 ☆

```c
#define _POSIX_C_SOURCE 200809L

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

const char *file_name(const char *pathname) {
    int file_name_position = 0;
    for (int i = 0; pathname[i]; i++) {
        if (pathname[i] == '/') { file_name_position = i + 1; }
    }
    return pathname + file_name_position;
}

void rm0(const char *pathname) {
    if (unlink(pathname) == -1) {
        fprintf(stderr, "rm: cannot remove '%s': ", file_name(pathname));
        perror("");
        return;
    }
}

int main(int argc, char *argv[]) {
    if (argc < 2) { return -1; }
    rm0(argv[1]);
    return argc & 0;
}
```

cat0 ☆

```c
#define _POSIX_C_SOURCE 200809L
#define BUFFER_SIZE 100

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

void cat0(const char *pathname) {
    int file_descriptor = open(pathname, 0);
    if (file_descriptor == -1) {
        perror("");
        return;
    }

    ssize_t read_count = -1;
    char buffer[BUFFER_SIZE];

    while (read_count) {
        read_count = read(file_descriptor, buffer, BUFFER_SIZE);
        if (read_count < 0) {
            perror("");
            if (close(file_descriptor)) { perror(""); }
            return;
        }

        ssize_t written_count = 0;
        while (written_count < read_count) {
            written_count = write(STDOUT_FILENO,
                                  buffer + written_count,
```

```
                                (size_t) (read_count - written_count));
            if (written_count < 0) {
                perror("");
                if (close(file_descriptor)) { perror(""); }
                return;
            }
        }
    }

    if (close(file_descriptor)) { perror(""); }
}

int main(int argc, char *argv[]) {
    if (argc < 2) { return -1; }
    cat0(argv[1]);
    return argc & 0;
}
```

touch0 ☆

```
#define _POSIX_C_SOURCE 200809L

#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <utime.h>

void touch0(const char *pathname) {
    int file_descriptor = open(pathname, O_CREAT, S_IRWXU);
    if (file_descriptor == -1) {
        perror("");
        return;
    }
    if (close(file_descriptor)) { perror(""); }
    utime(pathname, NULL);
}

int main(int argc, char *argv[]) {
    if (argc < 2) { return -1; }
    touch0(argv[1]);
    return argc & 0;
}
```

stat0 ☆☆

```
#define _POSIX_C_SOURCE 200809L

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

const char *file_name(const char *pathname) {
    int file_name_position = 0;
    for (int i = 0; pathname[i]; i++) {
        if (pathname[i] == '/') { file_name_position = i + 1; }
    }
    return pathname + file_name_position;
}

void stat0(const char *pathname) {
    struct stat info;
    if (stat(pathname, &info) == -1) {
        perror("");
        return;
    }
    printf("Size: %d\nFile: %s\nType: %s",
           (int) info.st_size,
           file_name(pathname),
           S_ISREG(info.st_mode) ? "regular file" : "directory");
}

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

```
    stat0(argv[1]);
    return argc & 0;
}
```

rm1 ★

(Resuelto en rm0)

Parte 2

ln0 ☆

```
#define _POSIX_C_SOURCE 200809L

#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>

void ln0(const char *original, const char *link) {
    if (symlink(original, link) == -1) { perror(""); }
}

int main(int argc, char *argv[]) {
    if (argc < 3) { return -1; }
    ln0(argv[1], argv[2]);
    return argc & 0;
}
```

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

Se crea un enlace roto, es decir que apunta a un nombre de archivo no existente.

mv0 ☆

```
#define _POSIX_C_SOURCE 200809L

#include <stdio.h>

void mv0(const char *old_name, const char *new_name) {
    if (rename(old_name, new_name) == -1) { perror(""); }
}

int main(int argc, char *argv[]) {
    if (argc < 3) { return -1; }
    mv0(argv[1], argv[2]);
    return argc & 0;
}
```

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

Sí, de hecho mover es renombrar el archivo (considerando como nombre de archivo a la ruta completa)

cp0 ☆☆

```
#define _POSIX_C_SOURCE 200809L
#define BUFFER_SIZE 100

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


void cp0(const char *original, const char *copy) {
    int original_fd = open(original, 0);
    if (original_fd == -1) {
        perror("");
        return;
```

```
    }

    int copy_fd = open(copy, O_CREAT | O_WRONLY | O_TRUNC, S_IRWXU);
    if (copy_fd == -1) {
        perror("");
        if (close(original_fd)) { perror(""); }
        return;
    }

    ssize_t read_count = -1;
    char buffer[BUFFER_SIZE];

    while (read_count) {
        read_count = read(original_fd, buffer, BUFFER_SIZE);
        if (read_count < 0) {
            perror("");
            if (close(original_fd)) { perror(""); }
            if (close(copy_fd)) { perror(""); }
            return;
        }

        ssize_t written_count = 0;
        while (written_count < read_count) {
            written_count = write(copy_fd,
                                  buffer + written_count,
                                  (size_t) (read_count - written_count));
            if (written_count < 0) {
                perror("");
                if (close(original_fd)) { perror(""); }
                if (close(copy_fd)) { perror(""); }
                return;
            }
        }
    }

    if (close(original_fd)) { perror(""); }
    if (close(copy_fd)) { perror(""); }
}

int main(int argc, char *argv[]) {
    if (argc < 3) { return -1; }
    struct stat info1;
    stat(argv[1], &info1);
    struct stat info2;
    stat(argv[2], &info2);
    if (info1.st_dev == info2.st_dev && info1.st_ino == info2.st_ino) {
        fprintf(stderr, "cp: '%s' and '%s' are the same file",
                argv[1], argv[2]);
    }
    cp0(argv[1], argv[2]);
    return argc & 0;
}
```

touch1 ★

(Resuelto en touch0)

ln1 ★

```
#define _POSIX_C_SOURCE 200809L

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

void ln1(const char *original, const char *link_path) {
    if (link(original, link_path) == -1) { perror(""); }
}

int main(int argc, char *argv[]) {
    if (argc < 3) { return -1; }
    ln1(argv[1], argv[2]);
    return argc & 0;
}
```

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

Un hard link crea nueva metadata que apunta al mismo inode, por lo que ambos archivos son independientes y de igual jerarquía, solo que comparten los mismos datos. Un soft link es un link a otro nombre en el file system.

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

Los datos del archivo se van a eliminar cuando nadie los referencie, por lo que lo único que se elimina es la metadata del archivo original. Una vez que se elimine el segundo archivo, se va a eliminar la metadata del mismo, y al no haber quien apunte al inodo (datos), se van a eliminar también.

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

El enlace no puede abrirse porque apunta a un nombre que no existe, y los datos se pierden porque no hay quien apunte al inodo.

Parte 3

tee0 ☆☆

```c
#define _POSIX_C_SOURCE 200809L
#define BUFFER_SIZE 100

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

void tee0(const char *pathname) {
    int output = open(pathname, O_CREAT | O_WRONLY | O_TRUNC, S_IRWXU);
    if (output == -1) {
        perror("");
        return;
    }

    ssize_t read_count = -1;
    char buffer[BUFFER_SIZE];

    while (read_count) {
        read_count = read(STDIN_FILENO, buffer, BUFFER_SIZE);
        if (read_count < 0) {
            perror("");
            if (close(output)) { perror(""); }
            return;
        }

        ssize_t file_written_count = 0;
        while (file_written_count < read_count) {
            file_written_count = write(output,
                                       buffer + file_written_count,
                                       (size_t) (read_count -
                                                 file_written_count));
            if (file_written_count < 0) {
                perror("");
                if (close(output)) { perror(""); }
                return;
            }
        }

        ssize_t stdout_written_count = 0;
        while (stdout_written_count < read_count) {
            stdout_written_count = write(STDOUT_FILENO,
                                         buffer + stdout_written_count,
                                         (size_t) (read_count -
                                                   stdout_written_count));
            if (stdout_written_count < 0) {
                perror("");
                if (close(output)) { perror(""); }
                return;
            }
        }
    }
```

```
    if (close(output)) { perror(""); }
}

int main(int argc, char *argv[]) {
    if (argc < 2) { return -1; }
    tee0(argv[1]);
    return argc & 0;
}
```

ls0 ☆☆☆

```
#define _POSIX_C_SOURCE 200809L

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

void ls0() {
    DIR *directory = opendir(".");
    if (directory == NULL) { perror(""); }
    for (struct dirent *child = readdir(directory);
         child != NULL;
         child = readdir(directory)) {
        printf("%s\n", child->d_name);
    }
    if (closedir(directory) == -1) { perror(""); }
}

int main() {
    ls0();
    return 0;
}
```

cp1 ☆☆☆

```
#define _POSIX_C_SOURCE 200809L

#include <sys/types.h>
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <string.h>


size_t file_size(int file_descriptor) {
    struct stat original_info;
    if (fstat(file_descriptor, &original_info) == -1) {
        return 0;
    }
    return (size_t) original_info.st_size;
}

long int min(long int a, long int b) {
    return a - b > 0 ? b : a;
}

void cp1(const char *original, const char *copy) {
    int original_fd = open(original, 0);
    if (original_fd == -1) {
        perror("");
        return;
    }

    int copy_fd = open(copy, O_CREAT | O_RDWR | O_TRUNC, S_IRWXU);
    if (copy_fd == -1) {
        perror("");
        if (close(original_fd)) { perror(""); }
        return;
    }

    size_t size = file_size(original_fd);
```

```
    if (size == 0 || ftruncate(copy_fd, size) == -1) {
        perror("");
        if (close(original_fd)) { perror(""); }
        if (close(copy_fd)) { perror(""); }
    }

    long int iterations = 0;
    long int page_size = sysconf(_SC_PAGE_SIZE);

    while ((size_t) (iterations * page_size) < size) {
        long int bytes_to_transfer = min(page_size,
                                         size - (iterations * page_size));

        char *source = mmap(0, (size_t) bytes_to_transfer,
                            PROT_READ, MAP_SHARED,
                            original_fd, iterations * page_size);
        if (source == MAP_FAILED) {
            perror("");
            if (close(original_fd)) { perror(""); }
            if (close(copy_fd)) { perror(""); }
            return;
        }

        char *destination = mmap(0, (size_t) bytes_to_transfer,
                                 PROT_READ | PROT_WRITE, MAP_SHARED,
                                 copy_fd, iterations * page_size);
        if (destination == MAP_FAILED) {
            perror("");
            if (close(original_fd)) { perror(""); }
            if (close(copy_fd)) { perror(""); }
            return;
        }

        memcpy(destination, source, (size_t) bytes_to_transfer);
        iterations += 1;
    }

    if (close(original_fd)) { perror(""); }
    if (close(copy_fd)) { perror(""); }
    return;
}

int main(int argc, char *argv[]) {
    if (argc < 3) { return -1; }
    struct stat info1;
    stat(argv[1], &info1);
    struct stat info2;
    stat(argv[2], &info2);
    if (info1.st_dev == 0) {
        fprintf(stderr, "cp: '%s' doesn't exist", argv[1]);
        return 0;
    }
    if (info1.st_dev == info2.st_dev && info1.st_ino == info2.st_ino) {
        fprintf(stderr, "cp: '%s' and '%s' are the same file",
                argv[1], argv[2]);
        return 0;
    }
    cp1(argv[1], argv[2]);
    return argc & 0;
}
```

ps0 ★★

```
#define _POSIX_C_SOURCE 200809L
#define BUFFER_SIZE 100

#include <sys/types.h>
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <dirent.h>
#include <ctype.h>
#include <stdbool.h>
#include <string.h>

void cat0(const char *pathname) {
```

```
    int file_descriptor = open(pathname, 0);
    if (file_descriptor == -1) {
        perror("");
        return;
    }

    ssize_t read_count = -1;
    char buffer[BUFFER_SIZE];

    while (read_count) {
        read_count = read(file_descriptor, buffer, BUFFER_SIZE);
        if (read_count < 0) {
            perror("");
            if (close(file_descriptor)) { perror(""); }
            return;
        }

        ssize_t written_count = 0;
        while (written_count < read_count) {
            written_count = write(STDOUT_FILENO,
                                  buffer + written_count,
                                  (size_t) (read_count - written_count));
            if (written_count < 0) {
                perror("");
                if (close(file_descriptor)) { perror(""); }
                return;
            }
        }
    }

    if (close(file_descriptor)) { perror(""); }
}

bool is_number(char *name) {
    bool result = true;
    int position = 0;
    while (name[position] != 0) {
        result = result && isdigit(name[position]);
        position++;
    }
    return result;
}

void ps0() {
    DIR *directory = opendir("/proc");
    if (directory == NULL) { perror(""); }
    for (struct dirent *child = readdir(directory);
         child != NULL;
         child = readdir(directory)) {
        if (is_number(child->d_name)) {
            ssize_t name_length = strlen(child->d_name);
            ssize_t written_count = 0;
            while (written_count < name_length) {
                written_count = write(STDOUT_FILENO,
                                      child->d_name + written_count,
                                      (size_t) (name_length - written_count));
            }
            write(STDOUT_FILENO, " ", 1);
            char path[BUFFER_SIZE];
            strcpy(path, "/proc/");
            strcat(path, child->d_name);
            strcat(path, "/comm");
            cat0(path);
        }
    }
    if (closedir(directory) == -1) { perror(""); }
}

int main() {
    ps0();
    return 0;
}
```