75.08 Sistemas Operativos

Lab Unix - Parte 1

Nombre y Apellido: Damián Cassinotti
Padrón: 96618
Fecha de Entrega: 06/04/2018

Índice

1. Comandos obligatorios

En esta primera parte del Lab se implementarán versiones simplificadas de comandos típicos de Unix. Por cada comando pedido se indicará un resumen de lo pedido y luego el código fuente utilizado para la solución.

1.1. rm0

Para el comando rm se pidió una implementación en la cual se puedan eliminar archivos regulares. La precondición que se debe cumplir es que el archivo que se pasará por parámetro existe y es regular.
A continuacón se mostrará el código de los archivos utilizados.

```
1 #define _POSIX_C_SOURCE 200809L
2 #include <unistd.h>
3
4 int main(int argc, char *argv[]);
5 void rm0(char* file);
```

rm0.h

```
1  #include "rm0.h"
2
3  int main(int argc, char *argv[]) {
4      if (argc != 2)
5          return -1;
6      rm0(argv[1]);
7      return 0;
8  }
9
10 void rm0(char* file) {
11     unlink(file);
12 }
```

rm0.c

1.2. cat0

Para nuestra implementación del comando cat se debe mostrar por pantalla el contenido de un archivo. A diferencia de la implementación nativa, solo mostraremos un solo archivo regular (el cual, por precondicion, existe y se tienen permisos de lectura).
La solución propuesta es la siguiente:

```c
#define _POSIX_C_SOURCE 200809L
// open
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
// read
#include <unistd.h>

#define READ_BYTES 10

int main(int argc, char* argv[]);
void cat0(const char *file);
// Return read-only file descriptor
int open0(const char *file);
// read READ_BYTES bytes from descriptor and stores in buffer
int read0(int descriptor, char *buffer);
// prints on stdio cantidad from buffer;
void write0(char* buffer, int cantidad);
```

cat0.h

```c
#include "cat0.h"

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

void cat0(const char *file) {
    int descriptor = open0(file);
    char buffer[READ_BYTES];
    int leidos;
    while ((leidos = read0(descriptor, buffer)) > 0) {
        write0(buffer, leidos);
    }
    close(descriptor);
}

int open0(const char *file) {
    return open(file, O_RDONLY);
}

int read0(int descriptor, char *buffer) {
    return read(descriptor, buffer, READ_BYTES);
}
 void write0(char *buffer, int cantidad) {
    int escritos = 0;
```

```
29      while (escritos < cantidad) {
30          escritos += write(STDOUT_FILENO, buffer, cantidad);
31      }
32  }
```

cat0.c

1.3. touch0

En este caso, cat0 implementará la función más utilizada del comando touch: crear un archivo vacío. No se tendrá en cuenta la modificación de fechas para archivos existentes. Es decir, si el archivo no existe se crea, y si existe no se hace nada.

El código fuente de la solución es el siguiente:

```
1  #define _POSIX_C_SOURCE 200809L
2  // Open
3  #include <sys/types.h>
4  #include <sys/stat.h>
5  #include <fcntl.h>
6  // Close
7  #include <unistd.h>
8
9  int main(int argc, char* argv[]);
10 void touch0(const char *file);
```

touch0.h

```
1  #include "touch0.h"
2
3  int main(int argc, char* argv[]) {
4      if (argc != 2)
5          return -1;
6      touch0(argv[1]);
7      return 0;
8  }
9
10 void touch0(const char *file) {
11     int fd = open(file, O_CREAT|O_RDWR, S_IRUSR|S_IWUSR|S_IRGRP|S_IWGRP|S_IROTH);
12     close(fd);
13 }
```

touch0.c

1.4. stat0

El comando stat muestra información de un archivo. A diferencia de esto, el comando stat0 será una versión reducida la cual solo mostrará el tamaño (en bytes), el nombre y el tipo de archivo (regular o directorio). Como precondición se indica que el archivo existe.
La solución al problema es la siguiente:

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

int main(int argc, char* argv[]);
void stat0(const char* file);
```

stat0.h

```
#include "stat0.h"

int main(int argc, char* argv[]) {
    if (argc != 2)
        return -1;
    stat0(argv[1]);
    return 0;
}

void stat0(const char* file) {
    struct stat buf;
    stat(file, &buf);
    printf("Size: %ld\n", buf.st_size);
    printf("File: %s\n", file);
    char* fileType;
    if (S_ISREG(buf.st_mode))
        fileType = "regular file";
    else
        fileType = "directory";
    printf("Type: %s\n", fileType);
}
```

stat0.c

2. Comandos opcionales

2.1. rm1

En este comando opcional se agrega una extensión al comando rm0 desarrollado anteriormente. Para este caso se elimina la precondición que indica que el archivo pasado por parámetro debe ser regular. De esta forma, se debe hacer un manejo de errores para el caso en que el archivo sea un directorio.
Para logarlo se debe manejar la variable errno y la función perror. La oslución propuesta es la siguiente:

```
1 #define _POSIX_C_SOURCE 200809L
2 #include <unistd.h>
3 #include <errno.h>
4 #include <stdio.h>
5 #include <string.h>
6
7 int main(int argc, char *argv[]);
8 void rm1(char* file);
```

rm1.h

```
1  #include "rm1.h"
2
3  int main(int argc, char *argv[]) {
4      if (argc != 2)
5          return -1;
6      rm1(argv[1]);
7      if (errno == EISDIR) {
8          char errMsg[strlen(argv[1]) + 20]; // mensaje + directorio
9          sprintf(errMsg, "rm: cannot remove '%s'", argv[1]);
10         perror(errMsg);
11     }
12     return 0;
13 }
14
15 void rm1(char* file) {
16     unlink(file);
17 }
```

rm1.c