Universidad Carlos III
Grado en ingeniería informática
Curso Sistemas Operativos 2023-24
Práctica 1
Curso 2023-24

Fecha: **08/03/2024** - ENTREGA: **1**

GRUPO: **81**

Alumnos: **Mario Ramos Salsón (100495849), Miguel Fidalgo García (100495770) y Víctor Martínez de las Heras (100495829)**

ÍNDICE

1. MYWC.C **2**
1.1 Descripción del código 2
1.2 Batería de pruebas 3
2. MYLS.C **4**
2.1 Descripción de código 4
2.2 Batería de pruebas 4
3. MYISHERE.C **5**
3.1 Descripción de código 5
3.2 Batería de pruebas 6
4.CONCLUSIONES **7**
4.1 Conclusión mywc 7
4.2 Conclusión myls 7
4.3 Conclusión myishere 7

1. MYWC.C

1.1 Descripción del código

El programa ***mywc*** espera como parámetro un archivo y realiza el conteo de los bytes, palabras y líneas de este. Finalmente muestra por pantalla el resultado.

Para ello definimos un buffer de tamaño 1 y así poder ir leyendo de 1 byte en 1 byte, tal y como se solicita. Realizamos las comprobaciones necesarias para asegurarnos que tenemos dos argumentos: el primero, el comando para la ejecución del programa y el segundo el archivo a leer.

A continuación, abrimos el archivo en modo solo lectura. Para ello hacemos uso de la función ***open()*** perteneciente a la librería ***<fcntl.h>*** y como parámetros, le pasamos el segundo argumento y la flag ***"O_RDONLY"***, para especificar el solo lectura. Está función devuelve un descriptor de fichero que guardamos en una variable para su posterior utilización. Si la función ***open()*** falla devuelve un -1, y en caso contrario el descriptor de fichero correspondiente.

A continuación creamos las variables necesarias para la ejecución del código. Estas son el número de bytes que leeremos, las líneas y palabras totales y un par más de variables que nos ayudarán a gestionar la lógica del bucle.

Para hacer el bucle, iremos leyendo el archivo guardado en el descriptor de fichero y cuando la variable sea 0 significa que ya no tiene más para leer y sale del bucle. Para leer hacemos uso de la función ***read()*** perteneciente a la librería ***<unistd.h>***, que nos devuelve el número de bytes contados. En este caso, al estar leyendo de byte en byte, siempre nos devuelve 1 hasta que finalice la lectura, entonces devolverá 0. Mientras está condición está activa, llevaremos a cabo toda la lógica para contar las palabras. Para ello hemos creado una variable ***found*** que es 0 por defecto y cambia a 1 cuando estamos leyendo algo que sea distinto de un espacio en blanco, '\t', '\n' o retorno de carro. Cuando el ***found*** es 1 simplemente contamos un byte y se avanza el puntero a la siguiente posición del fichero. Cuando el ***found*** es 0 es porque estamos en alguna de las posiciones anteriormente mencionadas, así que analizamos si el anterior elemento a la posición del buffer actual es distinto de un espacio en blanco, '\t', '\n' o retorno de carro, para contarlo como palabra. Esto se debe a que si tenemos dos espacios seguidos, dos saltos de línea o dos tabuladores seguidos, no deberá contar una palabra más.

Para contar las líneas se añade una condición que sume siempre que encontremos un '\n' ya que es lo que simboliza el salto de línea.

En el caso de que estemos con un archivo que solo tenga una línea de código sin '\n', '\t' o espacios en blanco, nos fijamos en el último carácter del documento mediante un condicional que analiza si este es diferente de un espacio en blanco, '\t' o '\n'. En este caso sumamos una palabra. Finalmente cerramos el fichero con la función ***close()***.

Nota: Debemos tener en cuenta el retorno de carro. De esto nos dimos cuenta al imprimir el ascii de todo el fichero, ya que aparecía el número 13. Esto hacía que la condición de cuando llegásemos a un '\n' y lo anterior no fuese una letra no funcionase. El motivo era que nosotros buscábamos espacios en blanco, '\t' o '\n' para determinar el conteo de una palabra, y al existir este retorno de carro, siempre había un 13 antes del '\n' que hacía que nuestro código se rompiese.

1.2 Batería de pruebas

Prueba	**Descripción**	**Resultado obtenido (./mywc)**	**Resultado esperado (wc)**
./mywc p1_tests/f1.txt	Fichero que contiene unas líneas de texto normales	2 15 79 p1_tests/f1.txt	2 15 79 p1_tests/f1.txt
./mywc p1_tests/empty.txt	Fichero completamente en blanco	0 0 0 p1_tests/empty.txt	0 0 0 p1_tests/empty.txt
./mywc p1_tests/aes.txt	Fichero que contiene una línea de 5 aes	0 1 5 p1_tests/aes.txt	0 1 5 p1_tests/aes.txt
./mywc p1_tests/f2.txt	Un fichero inexistente	File not found Devuelve -1	wc: f2.txt: No existe el archivo o el directorio
./mywc	No pasamos argumentos	Too few arguments Devuelve -1	No funciona el wc

2. MYLS.C

2.1 Descripción de código

El programa ***myls*** espera como parámetro un directorio o por el contrario usará el actual, e imprimirá el nombre de los ficheros que hay en él.

En primer lugar, verificamos si nos han pasado un directorio o tenemos que usar el actual. Para ello nos fijamos en el valor de la variable **argv** del main. Si nos lo han pasado, el nombre estará en ***argv[1],*** sino usamos la función ***getcwd()*** de la librería ***<unistd.h>*** para obtener el nombre del directorio actual. Para utilizar está función, incluiremos la librería ***<linux/limits.h>*** para hacer uso del valor ***PATH_MAX*** del sistema. Esto le dará el tamaño necesario al buffer para poder leer el nombre del directorio completo.

Después, abrimos el directorio y verificamos que este existe con la función ***opendir()*** perteneciente a la librería ***<dirent.h>,*** que devuelve un descriptor de directorio en caso de que exista, o NULL en caso contrario.

Para llevar a cabo la lectura de todo el directorio, iteramos sobre la función ***readdir()*** perteneciente a la librería ***<dirent.h>***. Esta nos devolverá un puntero a una estructura "***dirent"*** creada anteriormente. Una vez recibida esta información haremos uso del atributo ***d_name*** perteneciente a la estructura ***"dirent"***, para imprimir el nombre del fichero leído.

Repetimos todo este proceso hasta que la función ***readdir()*** devuelva ***NULL***, lo que indica que ya ha leído todo el directorio. Por último cerramos el directorio con la función ***closedir()***.

2.2 Batería de pruebas

Prueba	Descripción	Resultado obtenido (./myls)	Resultado esperado (ls -f -1)
./myls p1_tests/	Ejecutar la función en un directorio normal	dirA .. f1.txt . dirB	dirA .. f1.txt . dirB

./myls	Ejecutar la función sobre el propio directorio	.. . probador_ssoo_p1 .py mywc.c myishere p1_tests Makefile myls autores.txt myishere.c mywc myls.c	.. . probador_ssoo_p1 .py mywc.c myishere p1_tests Makefile myls autores.txt myishere.c mywc myls.c
./myls p2_tests/	Intentamos abrir un directorio que no existe	Could not open directory Devuelve -1	wc: p2: No existe el archivo o el directorio

3. MYISHERE.C

3.1 Descripción de código

El programa ***myishere*** se encarga de comprobar si un fichero pasado por parámetro, está en un directorio que también ha sido pasado por parámetro. Si este fichero existe devolverá un texto confirmándolo, y en caso contrario dirá que no existe.

Para ellos creamos un puntero de tipo directorio (***DIR***) donde almacenamos el valor que nos devuelve la función ***opendir()*** perteneciente a la librería ***<dirent.h>***. Está función recibe por parámetro el nombre del directorio y devolverá un puntero tipo ***DIR*** que apunte a ese directorio.

Después de esto, creamos una estructura de tipo ***dirent*** ya que ejecutamos la función ***readdir()*** perteneciente a la librería ***<dirent.h>***, que espera por parámetro un puntero al directorio y nos devuelve una estructura tipo dirent que guardamos en la variable ****dirp***.

Posteriormente comprobamos que sea distinto de null y que no hayamos encontrado ya el archivo que buscamos (mediante la variable found). Para realizar la comparación importamos la librería ***<string.h>*** para usar la función ***strcmp()***. Esta recibe dos parámetros: el nombre de la estructura dirp, que es el nombre del archivo que estamos revisando y el argumento a la hora de invocar la función que

era el archivo que se quería encontrar. Esta devolverá 0 si son iguales por lo que imprimimos la información y marcamos found a 1.

Si la variable found es 0 después significa que no hemos encontrado el archivo en ese directorio por lo que imprimimos que no está en el directorio. Finalmente, cerramos el directorio.

3.2 Batería de pruebas

Prueba	**Descripción**	**Resultado esperado (./myishere)**	**Resultado obtenido**
./myishere p1_tests f1.txt	Intentamos buscar un fichero en un directorio que sabemos que existe	File f1.txt is in directory p1_tests	Usando el explorador de archivos vemos que sí está el fichero en ese directorio
./myishere . p1_tests	Intentamos buscar un fichero en el directorio actual mediante el “.”	File autores.txt is in directory .	Usando el explorador de archivos vemos que sí está el fichero en ese directorio
./myishere p1_tests f2.txt	Intentamos buscar un fichero en un directorio que sabemos que no existe	File f2.txt is not in directory p1_tests	Usando el explorador de archivos vemos que no está el fichero en ese directorio
./myishere p1_tests/	No pasamos los suficientes parámetros	Too few arguments	No existente en linux

4.CONCLUSIONES

4.1 Conclusión mywc

Este ha sido el programa que más se nos ha complicado de la práctica. En primer lugar intentamos hacer un read de un determinado número de bytes mayor que uno, pero en la práctica se indicaba que la lectura tenía que hacerse byte a byte. En segundo lugar, intentamos hacer una solución que utiliza la función ***mmap***, guiándonos por un ejemplo resuelto de las diapositivas, pero tras preguntar en clase el profesor nos indicó que no había que hacer uso de estas librerías. En tercer lugar, lo intentamos hacer comparando el byte actual con el anterior, pero había demasiados casos posibles como para cubrirlos todos. Al final hicimos una variable auxiliar llamada found, que nos facilita que mientras no haya algo diferente a un espacio en blanco, \t, \n, o 13(retorno de carro), que no hiciera nada.

4.2 Conclusión myls

Este programa nos ha parecido bastante más sencillo. Simplemente creamos las variables y estructuras correspondientes e iteramos sobre el fichero para ir imprimiendo uno a uno los nombres de los ficheros. Cabe destacar que también tuvimos que hacer uso de la función ***getcwd()*** para cuando tuviésemos que obtener el nombre del directorio actual.

4.3 Conclusión myishere

Este programa fue el más sencillo de todos, ya que sigue exactamente la misma estructura que el programa ***ls*** cambiando simplemente que en vez de imprimir, haga uso de la función ***strcmp()*** para ver si existe el fichero en el directorio pedido.