Sistemas Operativos

Práctica 1: Llamadas al sistema operativo

Jesús María Llanes Muñoz - 100472261
Luis Guzmán Garrido Calvo - 100451684
Alejandro González Núñez - 100429135

Índice

1. Descripción del código....2
1.1. mywc.c.... 2
1.2. myls.c.... 2
2. Batería de pruebas....4
2.1. mywc.c.... 4
2.2. myls.c.... 5
2.3 myishere.c.... 6
3. Conclusiones....7

1. Descripción del código

1.1. mywc.c

Primero definimos un MAX_BUFFER que establece el tamaño máximo del búfer para leer el archivo de 512 La función principal toma los argumentos de la línea de comandos, argc y argv ya que serán necesarias para realizar la funcionalidad.

Se declaran las variables que se utilizarán para contar líneas, palabras, si estamos en un palabra o no y bytes, así como una variable para almacenar temporalmente cada carácter leído del archivo. Verifica si el número de argumentos es válido. Si no es así, imprime un mensaje de error utilizando fprintf() en el flujo de error estándar (stderr) y sale del programa con un código de salida -1.

Abre el archivo especificado en modo de solo lectura utilizando la función open(), y maneja cualquier error. Entra en un bucle while que lee el archivo carácter por carácter utilizando la función read(). Dentro del bucle, se cuentan las líneas, palabras y bytes según los criterios establecidos. Una vez finalizada la lectura del archivo, imprime las estadísticas recopiladas junto con el nombre del archivo utilizando printf(). Finalmente, cierra el archivo abierto con close() y retorna 0 para indicar que el programa se ejecutó exitosamente.

1.2. myls.c

Al igual que en el programa anterior recibimos los valores en argc y argv para poder realizar la funcionalidad.Lo primero que haremos es declarar las variables dir y entry para manejar el directorio y sus entradas respectivamente, y un arreglo dir_path para almacenar la ruta del directorio limitándose a un tamaño PATH_MAX estándar del sistema.

Verificamos el número de argumentos pasados al programa. Si argc es igual a 1, significa que no se proporcionaron argumentos, por lo que intenta abrir el directorio actual usando getcwd(), que obtiene el directorio de trabajo actual. Si falla, imprime un mensaje de error y termina el programa con salida -1. Luego, abre el directorio usando opendir() y lo asigna a la variable dir.

Si argc es igual a 2, significa que se proporcionó un argumento, que se asume que es el nombre del directorio a listar. Intenta abrir este directorio usando opendir(argv[1]). Si la operación falla (es decir, si dir es NULL), imprime un mensaje de error y termina el programa. Si argc es distinto de 1 o 2, imprime un mensaje de error indicando que el número de argumentos es incorrecto y termina el programa.

Utiliza un bucle while para iterar sobre todas las entradas del directorio usando readdir(). En cada iteración, imprime el nombre de la entrada usando printf(). Una vez que el bucle ha recorrido todas las entradas del directorio, cierra el directorio utilizando closedir() y devuelve 0 para indicar que el programa se ejecutó correctamente.

1.3. myishere.c

Declaramos las variables. Comprobamos el número de argumentos introducidos. Si el número de argumentos es distinto de 3, se lanzará un mensaje por la salida estándar de error y el programa terminará. Si el número de argumentos es correcto, se procederá a abrir el directorio pasado como parámetro con la llamada al sistema opendir.

Si el directorio se abre correctamente, se devolverá un descriptor de fichero dir y el programa continuará. De lo contrario, saltará un mensaje por la salida estándar de error y el programa terminará. Para ver las distintas entradas del directorio, se utilizará la llamada al sistema readdir a la que se le pasará el descriptor de fichero obtenido anteriormente. Esta irá iterando por las distintas entradas del directorio y comprobando si alguna de ellas coincide con el segundo parámetro introducido por el usuario.

Para esto utilizaremos strcmp. Si alguna entrada coincide, el valor de la variable fichero_encontrado se actualizará a 1 y se imprimirá el resultado deseado. De lo contrario, esta permanecerá en 0 y se indicará por pantalla que el fichero correspondiente no ha sido encontrado. Por último, cerramos el directorio.

2. Batería de pruebas

2.1. mywc.c

Nombre de la prueba	Descripción	Directorio/ Fichero	Resultado esperado	Válido	Resultado obtenido
Fichero vacío	Realizar mywc en un fichero txt vacío	./mywc p1_tests/**f_vacio.txt**	0 0 0 p1_tests/f_vacio.txt	OK	0 0 0 p1_tests/f_vacio.txt
Fichero de una sola palabra y línea	Realizar mywc en un fichero de una sola palabra y línea	./mywc p1_tests/**f_una_linea.txt**	1 1 N p1_tests/f_una_linea.txt	OK	1 1 2 p1_tests/f_una_linea.txt
Fichero de una palabra por línea	Realizar mywc en un fichero que tiene solo una palabra por línea	./mywc p1_tests/**f_multiples_lineas_sin_espacios.txt**	N N M p1_tests/f_multiples_lineas_sin_espacios.txt	OK	3 3 42 p1_tests/f_multiples_lineas_sin_espacios.txt
Fichero con tabs y espacios	Realizar mywc en un fichero con múltiples líneas, una con solo espacios, otra con solo tabs y la última con mezcla de las 2	./mywc p1_tests/**f_lineas_solo_espacios_tabulaciones.txt**	N 0 M p1_tests/f_lineas_solo_espacios_tabulaciones.txt	OK	3 0 45 p1_tests/f_lineas_solo_espacios_tabulaciones.txt
Fichero inexistente	Realizar mywc en un fichero que no existe	./mywc **random.txt**	ERROR al abrir el fichero: No such file or directory	Error	ERROR al abrir el fichero: No such file or directory
Prueba todo OK	Realizar una prueba genérica con datos normales	./mywc **p1_tests/f1.txt**	N M O p1_tests/f1.txt	OK	2 15 79 p1_tests/f1.tx

2.2. myls.c

Nombre de la prueba	**Descripción**	**Directorio/ Fichero**	**Resultado esperado**	**Válido**	**Resultado obtenido**
Directorio actual	Realizar myls en el directorio actual donde hemos compilado con la estructura del entregable	./myls	Todo el contenido del directorio junto con el directorio actual (.) y padre(..)	OK	mywc.o myls.o mywc.c myls.c mywc p1_tests .. myishere.c . Makefile myishere autores.txt myls myishere.o
Pasar un directorio como argumento	Realizar myls en un directorio diferente, en este caso p1_tests	./myls **p1_tests**	Todo el contenido que se encuentra en la ruta del directorio junto con el directorio actual (.) y padre(..).	OK	f1.txt .. f_multiples_lineas_sin_espacios.txt . f_vacio.txt f_una_linea.txt dirA dirB f_lineas_solo_espacios_tabulaciones.txt
Pasar más de un argumento	Realizar myls pasando la dirección de 2 directorios, en este caso p1_tests/dirA y p1_tests/dirB	./myls p1_tests/dirA **p1_tests/dirB**	Número de argumentos incorrecto: Success	Error	Número de argumentos incorrecto
Pasar una ruta inexistente	Realizar myls en un directorio que no existe	./myls **/random**	No se ha podido abrir el directorio indicado: No such file or directory	Error	No se ha podido abrir el directorio indicado: No such file or directory
Prueba todo ok	Realizar mywc en un directorio que general que todo linux tiene	./myls /boot	Todo el contenido del directorio boot	OK	Todo el contenido del directorio boot

2.3 myishere.c

Nombre de la prueba	**Descripción**	**Directorio/ Fichero**	**Resultado esperado**	**Válido**	**Resultado obtenido**
Sin argumentos	Realizar myishere sin proporcionar ningún argumento	./myishere	ERROR. Número de argumentos inválido	Error	ERROR. Número de argumentos inválido
Directorio inexistente	Realizar myishere en un directorio que no existe.	./mylishere p1_tests/**dirC** test1.txt	ERROR. No se ha podido abrir el directorio indicado	Error	ERROR. No se ha podido abrir el directorio indicado
Fichero inexistente	Realizar myishere en un fichero que no existe.	./mylishere p1_tests/dirA **test3.txt**	File test3.txt is not in directory p1_tests/dirA	OK	File test3.txt is not in directory p1_tests/dirA
Fichero de diferente tipo	Realizar myishere en un fichero java	./mylishere p1_tests/dirA **test2.java**	File test2.java is in directory p1_tests/dirA	OK	File test2.java is in directory p1_tests/dirA
Prueba todo Ok	Realizar myishere en un directorio con un fichero existente	./mylishere p1_tests/dirA test1.txt	File test1.txt is in directory p1_tests/dirA	OK	File test1.txt is in directory p1_tests/dirA

3. Conclusiones

El desarrollo de estos programas brindan una visión valiosa sobre la complejidad de trabajar con operaciones de archivos y directorios en un entorno Unix utilizando el lenguaje de programación C. Uno de los desafíos más significativos es comprender y manejar adecuadamente los punteros, estructuras y funciones proporcionadas por la biblioteca estándar de C, así como las llamadas al sistema operativo para interactuar con el sistema de archivos.

Además, se destaca la importancia de manejar correctamente los errores y excepciones que pueden surgir durante la ejecución del programa, ya que la manipulación incorrecta de archivos y directorios puede resultar en comportamientos inesperados o pérdida de datos. Esto resalta la necesidad de una cuidadosa planificación y prueba del código para garantizar su fiabilidad y robustez.

En términos de aprendizaje, realizar pruebas unitarias ayuda mucho a que podamos depurar nuestro código. Además, permite familiarizarse con las herramientas y técnicas utilizadas en el desarrollo de software en entornos Unix, lo que puede ser beneficioso para futuros proyectos y carreras en el campo de la informática.

En cuanto a los problemas encontrados, la verdad que no hemos tenido ninguno. Esto se debe a que la práctica estaba bien explicada y contábamos con un probador para saber si lo que estábamos haciendo estaba bien o no.