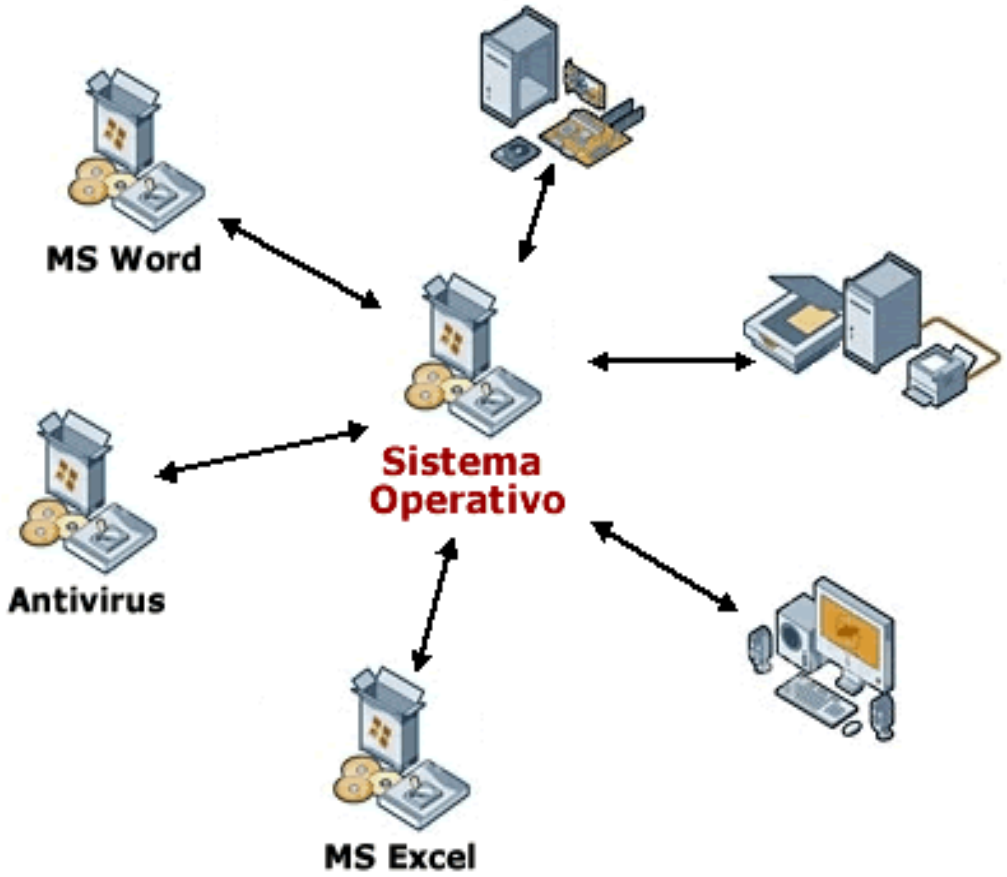
Titulación: GRADO INGENIERÍA INFORMÁTICA Y ADE Año Académico: 2023/24 – 2º cuatrimestre Asignatura: Sistemas Operativos (2º curso) Memoria de Prácticas 1: Llamadas al sistema operativo	uc3m | Universidad Carlos III de Madrid

Alumno/a:	**Juan Marín Olloqui**	**NIA:**	**100475063**
Alumno/a:	**David Sierra Fernández**	**NIA:**	**100471913**
Alumno/a:	**Paulo Álvarez Da Costa**	**NIA:**	**100475757**



Titulación: GRADO INGENIERÍA INFORMÁTICA Y ADE
Año Académico: 2023/24 – 2º cuatrimestre
Asignatura: Sistemas Operativos (2º curso)

Memoria de Prácticas 1: Llamadas al sistema operativo

Titulación: GRADO INGENIERÍA INFORMÁTICA Y ADE Año Académico: 2023/24 – 2º cuatrimestre Asignatura: Sistemas Operativos (2º curso) Memoria de Prácticas 1: Llamadas al sistema operativo	uc3m | Universidad Carlos III de Madrid

ÍNDICE DE CONTENIDOS

Mywc

- Descripción del código

El programa comienza con la inclusión de varias bibliotecas necesarias para su funcionamiento como `<stdlib.h>`, `<fcntl.h>` y `<unistd.h>`. Luego, define una constante `BUFFER_SIZE` que se establece en 100 MB.

.El programa primero verifica si se ha proporcionado al menos un argumento (además del nombre del programa en sí). Si no se proporciona ningún argumento, el programa imprime un mensaje de error y retorna `-1`, terminando su ejecución. Esto también ocurre si se proporcionan más argumentos de los requeridos.

A continuación, el programa intenta abrir el archivo especificado en modo de solo lectura utilizando la función `open`. Si la apertura del archivo falla por cualquier motivo (por ejemplo, el archivo no existe), `open` retorna `-1` y el programa imprime un mensaje de error y termina.

El programa luego declara varias variables para realizar el seguimiento del número de líneas, palabras y bytes en el archivo. También declara un búfer de tamaño `BUFFER_SIZE` para leer el archivo. Si la asignación de memoria para el búfer falla, el programa imprime un mensaje de error y termina.

El programa entonces entra en un bucle `while`, donde lee el archivo en bloques de `BUFFER_SIZE` bytes a la vez hasta que no queden más bytes por leer. Dentro de este bucle, hay otro bucle `for` que recorre cada byte leído.

Para cada byte, el programa incrementa el contador de bytes. Si el byte es un salto de línea (`'\n'`), incrementa el contador de líneas. Si el byte es un espacio en blanco, un salto de línea o una tabulación, y el byte anterior no era ninguno de estos, incrementa el contador de palabras.

Después de leer todo el archivo, el programa verifica si el último carácter leído no es un espacio en blanco, una tabulación o un salto de línea. Si no lo es, incrementa el contador de palabras, ya que esto significa que la última palabra del archivo no fue seguida por un espacio en blanco, una tabulación o un salto de línea.

Finalmente, el programa intenta cerrar el archivo. Si el cierre del archivo falla por cualquier motivo, el programa imprime un mensaje de error y termina. Si el cierre del archivo tiene éxito, el programa imprime el número de líneas, palabras y bytes en el archivo, seguido del nombre del archivo. Luego, libera la memoria asignada al búfer y retorna `0`, indicando que el programa se ejecutó con éxito.

- Batería de Pruebas

Descripción	Uso	Resultado esperado (descripción)	Resultado obtenido
Recibe un nº menor de parámetros	**./mywc**	Uso: ./mywc <nombre_del_fichero>	Uso: ./mywc <nombre_del_fichero>
Recibe un nº mayor de parámetros	**./mywc hello.c p1.c holaa.c**	Uso: ./mywc <nombre_del_fichero>	Uso: ./mywc <nombre_del_fichero>
Recibe un archivo vacío	**./mywc holaa.c**	0 0 0 holaa.c	0 0 0 holaa.c
Recibe un archivo con la palabra holaa	**./mywc holaa.c**	1 1 6 holaa.c	1 1 6 holaa.c
Recibe un archivo de 296 kb	**./mywc grande.c**	6148 22019 153860 grande.c	6148 22019 153860 grande.c
Recibe un archivo que solo tiene un caracter y empieza y termina en la línea 7	**./mywc linea.c**	7 1 8 línea.c	7 1 8 línea.c
Recibe un archivo que solo tiene una linea y tiene 500 caracteres	**./mywc linea2.c**	1 1 601 linea2.c	1 1 601 linea2.c
Recibe un archivo que pesa más que el BUFFER_SIZE asignado	**./mywc 200MB.c**	1 1 104857600 200MB.c	1 1 104857600 200MB.c

Myls

- Descripción del código

Este programa en C está diseñado para listar el contenido de un directorio. Al inicio, incluye varias bibliotecas necesarias para sus operaciones. En la función `main`, primero verifica si se ha pasado un argumento (que debería ser un directorio). Si se pasa un argumento, intenta abrir ese directorio. Si no se pasa ningún argumento, obtiene el directorio de trabajo actual con `getcwd` y luego intenta abrir ese directorio. Si hay algún problema al obtener el directorio de trabajo actual o al abrir el directorio, el programa imprime un mensaje de error y termina con un retorno de -1. Si puede abrir el directorio correctamente, procede a leer el contenido del directorio con `readdir`. Para cada entrada en el directorio, imprime el nombre de la entrada con `printf`. Finalmente, cierra el directorio con `closedir` y termina con un return 0.

Estos son los elementos más importantes del código:

1. `DIR *dir;` y `struct dirent *ent;`: Declara un puntero a una estructura `DIR` y un puntero a una estructura `dirent`. Estos se utilizarán para abrir y leer el directorio, respectivamente.
2. `if (argc == 2) {...}`: Si se pasa un argumento al programa (es decir, el nombre de un directorio), intenta abrir ese directorio.
3. `else if (argc == 1) {...}`: Si no se pasa ningún argumento, obtiene el directorio de trabajo actual con `getcwd` y luego intenta abrir ese directorio.
4. `else {...}`: Si se pasan más de dos argumentos,se imprime como se ha de utilizar el programa y se retorna -1.
5. `if (dir == NULL) {...}`: Si hay un error al abrir el directorio (por ejemplo, si el directorio no existe), imprime un mensaje de error y se retorna -1.
6. `while ((ent = readdir(dir)) != NULL) {...}`: Lee las entradas del directorio una por una. Para cada entrada, imprime el nombre de la entrada.
7. `closedir(dir);`: Cierra el directorio una vez que se han leído todas las entradas.
8. `return 0;`: Termina el programa con un estado de salida de 0, indicando que el programa se ha ejecutado correctamente.

- Batería de Pruebas

Descripción	Uso	Resultado esperado (descripción)	Resultado obtenido
Recibe un directorio vacío	**./myls <directorio_vacio>**	.. .	.. .
Recibe un archivo en vez de un directorio	**./myls <nombre_archivo>**	Error al abrir el directorio	Error al abrir el directorio
Recibe dos directorios en vez de uno	**./myls <dir1> <dir2>**	Uso 1: ./myls <nombre_del_directorio> Uso 2: ./myls	Uso 1: ./myls <nombre_del_directorio> Uso 2: ./myls
No recibe ningún directorio	**./myls**	Tiene que devolver el contenido del directorio en el que está myls.c	Devuelve el contenido del directorio en el que está myls.c
Comparamos el resultado de **myls** con el de **ls-f-1** sobre el mismo directorio	**./myls <dir>** **ls-f-1 <dir>**	Tiene que mostrar el mismo resultado	Muestra el mismo resultado
Recibe un directorio que necesita permisos para acceder a él	**./myls <dir_especial>**	Error al abrir el directorio	Error al abrir el directorio
Recibe un directorio con archivos cuyos nombres son muy largos	**./myls <dir_grande>**	Queremos que el nombre de los archivos aparezca completo y sin cortes	El nombre de los archivos largos aparece correctamente
Recibe un directorio que tiene archivos con nombres iguales, solo cambia su numeración	**./myls <nombres_parecidos>**	Queremos que muestre el nombre de cada archivo, por ejemplo, si un archivo es hello y otro es hello.c, tiene que imprimir los dos	Imprime el nombre de todos los archivos, a pesar de que sean iguales y solo cambie su formato.

Myishere

- Descripción del código

Este programa en C busca un archivo específico dentro de un directorio dado. Comienza verificando si se han proporcionado los argumentos correctos. Si es así, intenta abrir el directorio especificado. Luego, lee las entradas del directorio una por una, comparando el nombre de cada entrada con el nombre del archivo que se busca. Si encuentra una coincidencia, establece una bandera y termina la búsqueda. Finalmente, imprime un mensaje indicando si se encontró el archivo y en qué directorio, luego cierra el directorio y termina el programa. Si en cualquier momento ocurre un error (como no poder abrir el directorio), el programa imprime un mensaje de error y termina inmediatamente.

Estos son los elementos más importantes del código:

1. `if (argc != 3 ) {...}`: Comprueba si se han proporcionado suficientes argumentos. Si no se proporcionan exactamente dos argumentos (el nombre del directorio y el nombre del archivo), imprime un mensaje de uso y termina el programa con un estado de salida de -1.
2. `DIR *dir = opendir(argv[1]);`: Intenta abrir el directorio proporcionado como primer argumento. Si hay un error al abrir el directorio, imprime un mensaje de error y termina el programa.
3. `struct dirent *ent; int found = 0;`: Declara un puntero a una estructura `dirent` y una variable `found` para rastrear si se ha encontrado el archivo.
4. `while (!found && (ent = readdir(dir)) != NULL) {...}`: Lee las entradas del directorio una por una hasta que se encuentre el archivo o se hayan leído todas las entradas.
5. `if (strcmp(ent->d_name, argv[2]) == 0) {...}`: Compara el nombre de cada entrada con el nombre del archivo proporcionado como segundo argumento. Si los nombres coinciden, establece `found` en 1.
6. `if (found) {...} else {...}`: Si se ha encontrado el archivo, imprime un mensaje indicando que el archivo está en el directorio. Si no se ha encontrado el archivo, imprime un mensaje indicando que el archivo no está en el directorio.
7. `closedir(dir);`: Cierra el directorio una vez que se han leído todas las entradas o se ha encontrado el archivo.

- Batería de Pruebas

Descripción	Uso	Resultado esperado (descripción)	Resultado obtenido
Recibe un directorio no vacío y un archivo que está en el directorio	**./myishere <dir> <file>**	Debe imprimir por pantalla **File <file> is in directory <dir>**	Imprime **File <file> is in directory <dir>**
Recibe más de 3 argumentos o menos de 3	**./myishere <dir>** **ó** **./myishere <dir> <file> <file>**	Debe imprimir cual es el uso de la función myishere, es decir **Uso: ./myishere <nombre_del_directorio> <nombre_del_fichero>**	Imprime **Uso: ./myishere <nombre_del_directorio> <nombre_del_fichero>**
Recibe un directorio no existente	**./myishere <dir> <file>**	Debe imprimir **Error al abrir el directorio <dir>**	Imprime **Error al abrir el directorio <dir>**
Recibe un directorio que contiene muchos archivos y subdirectorios	**./myishere <dir> <file>**	Debe imprimir **File <file> is in directory <dir>** **ó** **File <file> is not in directory <dir>**	Imprime **File <file> is not in directory <dir>**
Recibe un directorio al que hay que acceder con una serie de permisos	**./myishere <dir> <file>**	Debe imprimir **Error al abrir el directorio <dir>**	Imprime **Error al abrir el directorio <dir>**
Recibe un archivo oculto, es decir que empieza por ‘.’ , y no es visible en el directorio, pero si está dentro de él	**./myishere <dir> <hidden_file>**	Debe imprimir igualmente que el archivo está en el directorio, ya que aunque no sea visible, el archivo sigue estando ahí.	Imprime **File <hidden_file> is in directory <dir>**

Conclusiones

- Problemas encontrados y cómo se han solucionado

Uno de los problemas más comunes que hemos tenido es cuando el directorio proporcionado no existe o el programa no tiene permisos para leerlo. En este caso, la función `opendir` devolverá NULL y el programa terminará. Para solucionar este problema, hemos tenido que verificar la existencia del directorio y los permisos antes de intentar abrirlo.

Otro problema es la falta de manejo de errores para la función `readdir`. Esta función podría fallar, pero el código actual no maneja este caso. Para arreglarlo hemos agregado un manejo de errores para `readdir`.

El programa también podría mejorarse en términos de manejo de argumentos. Actualmente, requiere exactamente dos argumentos, pero no maneja casos en los que se proporcionen menos o más argumentos. Hemos mejorado el manejo de argumentos para que el programa sea más flexible y fácil de usar.

Por último, el programa podría proporcionar más retroalimentación al usuario sobre lo que está haciendo. Esto no solo facilita la depuración de problemas, sino que también mejora la usabilidad del programa. Para ello, hemos agregado más declaraciones de impresión para informar al usuario sobre el progreso del programa.

En resumen, aunque los programas cumplan con sus objetivos principales hay varias áreas en las que podría mejorarse para aumentar su robustez, flexibilidad y usabilidad. Esto es lo que hemos intentado llevar a cabo.

- Opiniones personales

Las impresiones generales son positivas, especialmente antes de conocer el resultado final de la práctica. Inicialmente, la cantidad de información parecía abrumadora dada la brevedad del tiempo, pero con el paso del tiempo, hemos mejorado gradualmente. El ejercicio ha sido esencial para comprender los procesos subyacentes durante la ejecución del código. Los principales desafíos se encontraron al realizar las pruebas y al examinar el código. En resumen, ha sido un ejercicio bastante ameno y estimulante.