P1 SSOO ESP 23 24 - Practica 1

Sistemas Operativos (Universidad Carlos III de Madrid)

Área de Arquitectura y Tecnología de Computadores (ARCOS)

Universidad Carlos III de Madrid

SISTEMAS OPERATIVOS

Práctica 1. Llamadas al sistema operativo

Grado de Ingeniería en Informática
Grado en Matemática Aplicada y Computación
Doble Grado en Ingeniería Informática y Administración de Empresas

Curso 2023/2024

Índice

1 Enunciado de la Práctica

Esta práctica permite al alumno familiarizarse con las llamadas al sistema operativo (en concreto, el sistema de ficheros) siguiendo el estándar POSIX. Unix permite efectuar llamadas al sistema directamente desde un programa realizado en un lenguaje de alto nivel, en particular en lenguaje C. La mayor parte de las entradas/salidas (E/S) sobre ficheros en Unix pueden realizarse utilizando solamente cinco llamadas: `open, read, write, lseek y close`.

Para el kernel del sistema operativo, todos los archivos abiertos son identificados por medio de descriptores de fichero. Un descriptor de fichero es un entero no negativo. Cuando abrimos, `open`, un archivo que ya existe, el kernel devuelve un descriptor de fichero al proceso. Cuando queremos leer o escribir de/en un archivo, identificamos el archivo con el descriptor de fichero que fue devuelto por la llamada anteriormente descrita.

Cada archivo abierto tiene una posición de lectura/escritura actual ("**current file offset**"). Está representado por un entero no negativo que mide el número de bytes desde el comienzo del archivo. Las operaciones de lectura y escritura comienzan normalmente en la posición actual y provocan un incremento en dicha posición, igual al número de bytes leídos o escritos. Por defecto, esta posición es inicializada a 0 cuando se abre un archivo, a menos que se especifique la opción `O_APPEND`. La posición actual (`current_offset`) de un archivo abierto puede cambiarse explícitamente utilizando la llamada al sistema `lseek`.

Para manipular directorios, se pueden utilizar las llamadas al sistema `opendir, readdir y closedir`. Un directorio abierto se identifica con un descriptor de directorio, que es un puntero a un tipo DIR (*DIR**). Cuando abrimos un directorio con `opendir`, el kernel devuelve un descriptor de directorio, sobre el cual se pueden leer las entradas de dicho directorio mediante llamadas a la función `readdir`. La llamada `readdir` devuelve una entrada de directorio en un puntero a una estructura dirent (*struct dirent**). Dicha estructura contendrá los campos correspondientes a esa entrada como el nombre de la entrada, o el tipo (si es un fichero normal, si es otro directorio, enlaces simbólicos, etc.). Sucesivas llamadas a la función `readdir` irán devolviendo las sucesivas entradas de un directorio abierto.

1.1 Descripción de la Práctica

Se pretende implementar tres programas en C que utilicen las llamadas al sistema anteriormente descritas. Dichos programas serán **mywc**, **myls** y **myishere**. Para ello, dispondrán de los correspondientes ficheros de código *mywc.c*, *myls.c* y *myishere.c*.

1.1.1 mywc

El primer programa, **mywc**, abrirá un fichero especificado como argumento, **contará el número de líneas, palabras y bytes del mismo**, y mostrará estas por la salida estándar (la consola) utilizando las llamadas al sistema que considere oportunas. Para ello:

- Abrirá el fichero pasado como parámetro.
- Leerá los contenidos del fichero de **byte en byte**.

- Actualice los contadores con base en los bytes leídos. Se entiende que **dos líneas están separadas por el carácter '\n'**, mientras que **dos palabras pueden estar separadas por los caracteres ' ' (espacio en blanco) o '\t'**. Además, no habrá más de un espacio o salto de línea sucesivo.
- Mostrará los resultados por la consola, seguidos del nombre del fichero. Separe cada valor del siguiente con un espacio.
- Finalmente cerrará el fichero.

```
$ ./mywc p1_tests/f1.txt
 2 15 85 p1_tests/f1.txt
```

- **Uso:** `./mywc <nombre_del_fichero>`
- **Requisitos:**
  - El programa debe mostrar el **número de líneas, palabras y bytes por consola, seguidos del nombre del fichero leído**.
  - El programa mostrara los datos en el siguiente formato:
    `<lineas><espacio><palabras><espacio><bytes><espacio><nombre_del_fichero>`
  - El programa debe devolver -1 si no se le ha pasado ningún argumento de entrada.
  - El programa debe devolver -1 si hubo un error al abrir el fichero (e.g. el fichero no existe).
  - El programa debe devolver 0 si todo funcionó correctamente.
- **Sugerencia de prueba:**[1] Constatar que la salida del programa sobre un fichero coincide con la del mandato ***wc*** (sin argumentos) sobre ese mismo fichero. Para ello se recomienda comparar las dos salidas con el comando ***diff***.

1.1.2 myls

El segundo programa, ***myls***, abrirá un directorio especificado como argumento (o el directorio actual si no se especifica ningún directorio como argumento), y mostrará por pantalla el nombre de todas las entradas de dicho directorio, **imprimiendo una entrada por línea**. Para ello:

- Obtendrá el directorio a listar, de los argumentos del programa u obtendrá el directorio actual mediante la llamada ***getcwd***. Utilice la constante `PATH_MAX` como tamaño máximo que puede llegar a tener la ruta del directorio actual.
- Lo abrirá mediante ***opendir***.

[1] Cumplir esta prueba no es garantía de tener la máxima nota en el ejercicio. Se trata solo de una sugerencia para que los alumnos comprueben el funcionamiento general de su programa. Los alumnos deberán además cumplir los otros requisitos del programa, realizar el código adecuado, comentarlo, probar casos extremos, y en general cumplir con las demás exigencias descritas en el enunciado de la práctica.

- Luego, leerá cada una de las entradas del directorio mediante ***readdir*** e imprimirá el nombre de la entrada mediante ***printf***.
- Finalmente cerrará el descriptor de directorio mediante ***closedir***.

```
$ ./myls p1_tests/
dirC
f1.txt
dirA
f2.txt
.
..
```

- **Uso 1:** `./myls <nombre_del_directorio>`
- **Uso 2:** `./myls`
- **Requisitos:**
  - El programa debe listar **todas las entradas** del directorio, en el orden en que las vaya devolviendo la llamada *readdir*, y mostrando cada entrada línea por línea.
  - El programa debe listar las entradas del directorio pasado como parámetro (uso 1), o del directorio actual si no le han pasado ningún parámetro (uso 2).
  - **myls** deberá mostrar el directorio actual (.) y el directorio padre (..).
  - El programa debe devolver **-1** si hubo un error al abrir el directorio (e.g. el directorio no existe).
- **Sugerencia de prueba:**[2] Constatar que la salida del programa sobre un directorio coincide con la del mandato **ls −f −1** sobre ese mismo directorio: **ls -f -1 <nombre_del_directorio>**. Para ello se recomienda comparar las dos salidas con el comando ***diff***.

1.1.3 myishere

El tercer programa, ***myishere***, recibe el nombre de un directorio como primer argumento y comprueba si el fichero cuyo nombre recibe como segundo argumento se encuentra en ese directorio. Para ello:

- Se abre el directorio recibido como argumento usando ***opendir***.
- Se lee cada entrada llamando a ***readdir*** y se compara el nombre de la entrada con el nombre de fichero recibido como argumento. Realice esta operación hasta que se encuentre una coincidencia o no haya más entradas en el directorio.

[2]Cumplir esta prueba no es garantía de tener la máxima nota en el ejercicio. Se trata solo de una sugerencia para que los alumnos comprueben el funcionamiento general de su programa. Los alumnos deberán además cumplir los otros requisitos del programa, realizar el código adecuado, comentarlo, probar casos extremos, y en general cumplir con las demás exigencias descritas en el enunciado de la práctica.

- Si se encuentra el fichero, se imprime por la salida estandar (la consola) el nombre del fichero con el siguiente formato:
  `File <nombre_del_fichero> is in directory <nombre_del_directorio>`
- Si no se encuentra el fichero, se imprime en la salida estandar (la consola) un mensaje con el siguiente formato:
  `File <nombre_del_fichero> is not in directory <nombre_del_directorio>`
- Por último, cierre el directorio mediante ***closedir***.

```
$./myishere p1_test f1.txt
  File f1.txt is in directory p1_test
$./myishere p1_test f2.txt
  File f2.txt is not in directory p1_test
```

- **Uso:** `./myishere <nombre_del_directorio> <nombre_del_fichero>`
- **Requisitos:**
  - El programa debe funcionar con cualquier directorio y comprobar si alguna de sus entradas coincide con el fichero proporcionado.
  - El programa debe mostrar uno de los mensajes anteriormente mencionados, dependiendo de si el fichero ha sido encontrado o no.
  - El programa debe devolver -1 si no se proporcionan suficientes argumentos.
  - El programa debe devolver -1 si hay algún problema al abrir el directorio.
  - El programa debe devolver 0 si todo funciona correctamente (por funcionar correctamente se entiende comprobar si el fichero está en el directorio o no sin generar ningún tipo de error).
- **Nota:** La librería `string.h` contiene la función `strcmp`. Puede utilizarla para comparar dos cadenas.

1.1.4 Código Fuente de Apoyo

Para facilitar la realización de esta práctica se dispone del fichero **p1_llamadas_2024.zip** que contiene código fuente de apoyo. Para extraer su contenido ejecutar el siguiente comando:

`unzip p1_llamadas_2024.zip`

Al extraer su contenido, se crea el directorio `p1_llamadas/`, donde se debe desarrollar la práctica. Dentro de este directorio se habrán incluido los siguientes ficheros:

- **Makefile**
  NO debe ser modificado. Fichero fuente para la herramienta `make`. Con él se consigue la recompilación automática sólo de los ficheros fuente que se modifiquen. Utilice *make* para compilar los programas, y *make clean* para eliminar los archivos compilados.

- **mywc.c**
  Debe modificarse. Fichero fuente de C donde los alumnos deberán codificar el programa **mywc**.
- **myls.c**
  Debe modificarse. Fichero fuente de C donde los alumnos deberán codificar el programa **myls**.
- **myishere.c**
  Debe modificarse. Fichero fuente de C donde los alumnos deberán codificar el programa **myishere**.
- **autores.txt**
  Debe modificarse. Fichero en formato `txt` donde incluir los autores de la práctica.
- **probador_ssoo_p1.py**
  NO debe ser modificado. Corrector proporcionado de la práctica.
- **p1_tests/**
  Este directorio contiene ficheros y directorios de ejemplo, para poder ejecutar y probar los programas.

1.1.5 Corrector proporcionado

Se proporciona a los alumnos el script en **python (versión 3) probador_ssoo_p1.py** que verifica que el formato del entregable de la práctica es el correcto (sigue las especificaciones de nombrado, y está bien comprimido) y ejecuta tests de funcionalidad, dando como resultado una nota tentativa del código proporcionado. El probador deberá ejecutarse en las máquinas virtuales con Ubuntu Linux o en la Aulas Virtuales proporcionadas por el laboratorio del Departamento de Informática. El comando para ejecutar el corrector es el siguiente:

```
python3 probador_ssoo_p1.py <entregable.zip>
```

Siendo `entregable.zip` el fichero que se va a entregar por Aula Global (ver siguiente apartado). Ejemplo:

```
$ python3 probador_ssoo_p1.py ssoo_p1_100254896_100047014.zip
```

El corrector imprimirá mensajes por pantalla indicando si el formato es o no correcto.

2 Entrega

2.1 Plazo de Entrega

La fecha límite de entrega de la práctica en AULA GLOBAL será el **8 de marzo de 2024 (hasta las 23:55h)**

2.2 Procedimiento de entrega de las prácticas

La entrega de las prácticas ha de realizarse de forma electrónica y por **un único integrante del grupo**. En AULA GLOBAL se habilitarán unos enlaces a través de los cuales se podrá realizar la entrega de las prácticas. En concreto, **se habilitará un entregador para el código de la práctica, y otro de tipo TURNITIN** para la memoria de la práctica.

2.3 Documentación a Entregar

Se debe entregar un archivo comprimido en formato zip con el nombre:

ssoo_p1_AAAAAAAAA_BBBBBBBBB_CCCCCCCC.zip

Donde A...A, B...B y C...C son los NIAs de los integrantes del grupo. En caso de realizar la práctica en solitario, el formato será **ssoo_p1_AAAAAAAAA.zip**. **El archivo zip se entregará en el entregador correspondiente al código de la práctica**. El archivo debe contener:

- **Makefile**
- **mywc.c**
- **myls.c**
- **myishere.c**
- **autores.txt: Fichero de texto en formato csv con un autor por línea. El formato es: NIA, Apellidos, Nombre**

NOTA

Para comprimir dichos ficheros y ser procesados de forma correcta por el probador proporcionado, se recomienda utilizar el siguiente comando:

```
zip ssoo_p1_AAA_BBB_CCC.zip Makefile mywc.c myls.c myishere.c autores.txt
```

La memoria se entregará en formato PDF en un fichero llamado:

ssoo_p1_AAAAAAAAA_BBBBBBBBB_CCCCCCC.pdf.

Solo se corregirán y calificarán memorias en formato pdf. Tendrá que contener al menos los siguientes apartados:

- **Descripción del código** detallando las principales funciones implementadas. NO incluir código fuente de la práctica en este apartado. Cualquier código será automáticamente ignorado.
- **Batería de pruebas** utilizadas y resultados obtenidos. Se dará mayor puntuación a pruebas avanzadas, casos extremos, y en general a aquellas pruebas que garanticen el correcto funcionamiento de la práctica en todos los casos. Hay que tener en cuenta:

1. Que un programa compile correctamente y sin advertencias (warnings) no es garantía de que funcione correctamente.
2. Evite pruebas duplicadas que evalúan los mismos flujos de programa. La puntuación de este apartado no se mide en función del número de pruebas, sino del grado de cobertura de las mismas. Es mejor pocas pruebas que evalúan diferentes casos a muchas pruebas que evalúan siempre el mismo caso.

- **Conclusiones**, problemas encontrados, cómo se han solucionado, y opiniones personales.

Se puntuará también los siguientes aspectos relativos a la **presentación** de la práctica:

- Debe contener portada, con los autores de la práctica y sus NIAs.
- Debe contener índice de contenidos.
- La memoria debe tener números de página en todas las páginas (menos la portada).
- El texto de la memoria debe estar justificado.

El archivo pdf se entregará en el entregador correspondiente a la memoria de la práctica (entregador TURNITIN).

NOTA: Es posible entregar el código de la práctica tantas veces como se quiera dentro del plazo de entrega, siendo la última entrega realizada la versión definitiva. **LA MEMORIA DE LA PRÁCTICA ÚNICAMENTE SE PODRÁ ENTREGAR UNA ÚNICA VEZ A TRAVÉS DE TURNITIN.**

3 Normas

1. Las prácticas que no compilen o que no se ajusten a la funcionalidad y requisitos planteados, obtendrán una calificación de 0.

2. **Los programas que utilicen funciones de biblioteca (`fopen`, `fread`, `fwrite`, `etc.`) o similares, en vez de llamadas al sistema, obtendrán una calificación de 0. Tampoco se permite utilizar sentencias o funciones como `goto` o `stat`.**

3. Se prestará especial atención a detectar funcionalidades copiadas entre dos prácticas. En caso de encontrar implementaciones comunes en dos prácticas, los alumnos involucrados (copiados y copiadores) perderán las calificaciones obtenidas por evaluación continua.

4. Los programas deben compilar sin **warnings**.

5. Los programas deberán funcionar bajo un sistema Linux, no se permite la realización de la práctica para sistemas Windows. Además, para asegurarse del correcto funcionamiento de la práctica, deberá chequearse su compilación y ejecución en máquina virtual con Ubuntu Linux o en las Aulas Virtuales proporcionadas por el laboratorio de informática de la universidad. Si el código presentado no compila o no funciona sobre estas plataformas la implementación no se considerará correcta.

6. Un programa no comentado, obtendrá una **calificación muy baja**.

7. La entrega de la práctica se realizará a través de Aula Global, tal y como se detalla en el apartado Entrega de este documento. No se permite la entrega a través de correo electrónico sin autorización previa.

8. Se debe respetar en todo momento el formato de la entrada y salida que se indica en cada programa a implementar.

9. Se debe realizar un control de errores en cada uno de los programas, **más allá de lo solicitado explícitamente en cada apartado**.

Los programas entregados que no sigan estas normas no se considerarán aprobados.

4 Anexo (Llamadas al Sistema)

Las llamadas al sistema proporcionan la interfaz entre el sistema operativo y un programa en ejecución. UNIX permite efectuar llamadas al sistema directamente desde un programa realizado en un lenguaje de alto nivel, en particular en lenguaje C, en cuyo caso las llamadas se asemejan a llamadas a funciones, tal y como si estuvieran definidas en una biblioteca estándar. El formato general de una llamada al sistema es:

```
status = funcion_estandar (arg1, arg2,.....)
```

En caso de realizar una llamada sin éxito, devolvería en la variable status un valor -1. En la variable global `errno` se coloca el número de error, con el cual podemos obtener la asociación del error con lo que realmente ha ocurrido en el fichero `errno.h`, (contenido en la ruta: `/usr/src`. En linux : `/usr/src/linux/include/asm/errno.h`).

4.1 Llamadas al sistema relacionadas con archivos

```
int open(const char * path, int flag, ...)
```

Abre o crea un fichero especificado por `path`. El fichero abierto puede utilizarse para lectura, escritura, o ambas, en función de lo especificado por `flag`. Devuelve un descriptor de fichero que se puede utilizar para la lectura o escritura en el archivo.

Más ayuda en: `man 2 open`

```
int close(int fildes)
```

Cierra un archivo abierto anteriormente asociado al descriptor `fildes`. Si n = -1 → Error al cerrar el fichero.

Más ayuda en: `man 2 close`

```
ssize_t read(int fildes, void * buf, size_t nbyte)
```

Intenta leer de un archivo (cuyo descriptor de fichero `fildes` se obtuvo de abrirlo) tantos bytes como indica `nbyte`, colocando la información leída a partir de la dirección de memoria `buf`. Devuelve el número de bytes leídos (que puede ser menor o igual a `nbyte`). Si retorno = 0 → Fin de fichero (EOF). Si retorno = -1 → Error de lectura.

Más ayuda en: `man 2 read`

```
ssize_t write(int fildes, const void * buf, size_t nbyte)
```

Intenta escribir en un archivo (cuyo descriptor de fichero *fildes* se obtuvo de abrirlo) tantos bytes como indica *nbyte*, tomándolos de la dirección de memoria indicada *buf*. Devuelve el número de bytes que realmente se han escrito (que puede ser menor o igual a *nbyte*). Si retorno = -1 → Error de escritura.

Cada `write` (así como cada `read`), actualiza automáticamente la posición actual del fichero que se usa para determinar la posición en el archivo del siguiente `write` o `read`.

Más ayuda en: `man 2 write`

```
off_t lseek(int fildes, off_t offset, int whence)
```

Modifica el valor del apuntador del descriptor *fildes* en el archivo, a la posición explícita en desplazamiento a partir de la referencia impuesta en origen, de forma que las llamadas `write` o `read` pueden iniciarse en cualquier parte del archivo.

- Si retorno = -1 → Error de posicionamiento.

El parámetro *whence* puede tomar los siguientes valores:

- SEEK_SET → desde el principio del fichero.
- SEEK_CUR → desde la posición actual.
- SEEK_END → desde el final del fichero.

El parámetro offset se expresa en bytes, y toma un valor positivo o negativo. Ejemplo:
`a b c d e f g h i`
`lseek(5,4,SEEK_SET)` → Al avanzar 4 bytes, la siguiente lectura sería la "e". El descriptor del fichero abierto "fd" es 5.

Más ayuda en: `man 2 lseek`

4.2 Llamadas al sistema relacionadas con directorios

```
DIR * opendir(const char * dirname)
```

Abre un directorio existente especificado por *dirname*. Devuelve un descriptor de directorio que se puede utilizar para la lectura de las entradas de dicho directorio. Si retorno = NULL → Error al abrir el directorio.

Más ayuda en: `man opendir`

```
struct dirent * readdir(DIR * dirp)
```

Lee de un directorio (cuyo descriptor de directorio *dirp* se obtuvo al abrirlo) la siguiente entrada del directorio. La estructura dirent contiene un campo *d_name* (*char * d_name*) con el nombre de la entrada y un campo *d_type* (*unsigned char d_type*) con el tipo de la entrada (fichero, otro directorio, etc.).

Siguientes llamadas a esta función sobre el mismo descriptor de directorio devuelven las subsiguientes entradas. Cuando no quedan más entradas por devolver, esta llamada devuelve NULL.

Más ayuda en: `man readdir`

```
int closedir(DIR * dirp)
```

Cierra un directorio abierto anteriormente. Si retorno = -1→ Error al cerrar el directorio.

Más ayuda en: `man closedir`

4.3 Manual (man function)

man es el paginador del manual del sistema, es decir permite buscar información sobre un programa, una utilidad o una función. Véase el siguiente ejemplo:

```
man [seccion] open
```

Una página de manual tiene varias partes. Éstas están etiquetadas como NOMBRE, SINOPSIS, DESCRIPCIÓN, OPCIONES, FICHEROS, VÉASE TAMBIÉN, BUGS, y AUTOR. En la etiqueta de SINOPSIS se recogen las librerías (identificadas por la directiva #include) que se deben incluir en el programa en C del usuario para poder hacer uso de las funciones correspondientes. **Para salir de la página mostrada, basta con pulsar la tecla 'q'.**

Las formas más comunes de usar man son las siguientes:

- **man sección elemento**: Presenta la página de elemento disponible en la sección del manual.
- **man -a elemento**: Presenta, secuencialmente, todas las páginas de elemento disponibles en el manual. Entre página y página se puede decidir saltar a la siguiente o salir del paginador completamente.
- **man -k palabra-clave**: Busca la palabra-clave entre las descripciones breves y las páginas de manual y presenta todas las que casen.

5 Bibliografía

- El lenguaje de programación C: diseño e implementación de programas Félix García, Jesús Carretero, Javier Fernández y Alejandro Calderón. Prentice-Hall, 2002.
- The UNIX System S.R. Bourne Addison-Wesley, 1983.
- Advanced UNIX Programming M.J. Rochkind Prentice-Hall, 1985.
- Sistemas Operativos: Una visión aplicada Jesús Carretero, Félix García, Pedro de Miguel y Fernando Pérez. McGraw-Hill, 2001.
- Programming Utilities and Libraries SUN Microsystems, 1990.
- Unix man pages (`man function`)