Área de Arquitectura y Tecnología de Computadores

Universidad Carlos III de Madrid

SISTEMAS OPERATIVOS

Práctica 1. Llamadas al sistema operativo

Grado de Ingeniería en Informática

Curso 2013/2014

Índice

1. Enunciado de la Práctica

Esta práctica permite al alumno familiarizarse con las llamadas al sistema operativo (especialmente, 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, que para esta práctica será el 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 archivo*. Un descriptor de archivo es un entero no negativo. Cuando abrimos, `open`, un archivo que ya existe, el núcleo devuelve un descriptor de archivo al proceso. Cuando queremos leer o escribir de/en un archivo, identificamos el archivo con el descriptor de archivo 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 núcleo 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***, ***myenv*** y ***myishere***. Para ello, dispondrán de los correspondientes ficheros de código *mywc.c*, *myenv.c* y *myishere.c*.

mywc

El primer programa, ***mywc***, abrirá un fichero especificado como argumento, contará el número de líneas, palabras y bytes del mismo, y mostrará estas cuentas 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 en base a los bytes leídos. Se entiende que dos líneas están separadas por el carácter '\n', mientras que dos palabras pueden estar separadas por los caracteres ' ' o '\t'. No habrá más de un espacio o salto de línea sucesivo.
- Mostrará los resultados por la salida estándar, seguidos del nombre del fichero. Separe cada valor del siguiente con un espacio.
- Finalmente cerrará el fichero.

```
./mywc f1.txt
6 25 1048 f1.txt
```

Uso: `./mywc <fichero de entrada>`

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*:* *<líneas><espacio><palabras><espacio><bytes><espacio><nombre 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.

myenv

El segundo programa, ***myenv***, guardará en un fichero las variables de entorno almacenadas en la variable *environ*, proporcionada en el programa. Para ello:

- Abrirá el fichero pasado como parámetro.
- Leerá las entradas de la variable de una en una.
- Guardará dichas variables en el fichero.
- Finalmente cerrará el fichero.

```
./myenv salida.txt
```

Uso: `./myenv <fichero de salida>`

Requisitos:

- El programa debe almacenar todas las entradas de la variable, en el orden en que estén almacenadas.
- El programa debe separar cada entrada por un salto de línea (\n).
- 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.

Notas:

- El formato de la variable de entrono *environ* es un array de cadenas de caracteres, por tanto, cada posición del array constará de un array de caracteres terminado en '\0'.
- La última posición del array *environ* apuntará a NULL.
- El contenido del fichero generado por el programa será similar a la salida del comando `env`.

myishere

El tercer programa, ***myishere***, abrirá un directorio pasado como parámetro y buscará en el mismo si existe un fichero cuyo nombre también se pasará como parámetro. Para ello:

- Abrirá el directorio pasado como parámetro mediante la llamada al sistema correspondiente.
- Luego, leerá las entradas del directorio sucesivamente mediante *readdir* hasta que o bien se encuentre una entrada cuyo nombre coincida con el parámetro, o bien se terminen las entradas.
- Si el nombre se ha localizado, se imprimirá por pantalla el mensaje "File <nombre a buscar> is in directory <nombre directorio>". En caso de no haberse localizado, se imprimirá por pantalla el mensaje "File <nombre a buscar> is not in directory <nombre directorio>".
- Finalmente cerrará el directorio.

../myishere /tmp ishere.txt

File ishere.txt is in directory /tmp

../myishere /tmp isnothere.txt

File isnothere.txt is not in directory /tmp

Uso: `./myishere <directorio> <nombre de fichero>`

Requisitos:

- El programa deberá poder revisar un directorio cualquiera y comprobar si alguna de sus entradas coincide con el nombre que se desea localizar.
- El programa deberá mostrar uno de los dos mensajes descritos anteriormente, según corresponda.
- El programa debe devolver -1 si hubo un error al abrir el directorio.
- El programa debe devolver un 0 si todo funcionó correctamente (se entiende que el programa funciona correctamente si es capaz de determinar que el fichero está o no en el directorio sin que se produzcan errores).

Notas:

- La librería `string.h` contiene una función de comparación de cadenas llamada `strcmp`.

1.2. Código Fuente de Apoyo

Para facilitar la realización de esta práctica se dispone del fichero p1_llamadas_2014.tgz que contiene código fuente de apoyo. Para extraer su contenido ejecutar lo siguiente:

```
tar zxvf p1_llamadas_2014.tgz
```

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
Fichero fuente para la herramienta `make`. **NO debe ser modificado.** 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
Fichero fuente de C en donde los alumnos deberán codificar el programa ***mywc***.

myenv.c
Fichero fuente de C en donde los alumnos deberán codificar el programa ***myenv***.

myishere.c
Fichero fuente de C en donde los alumnos deberán codificar el programa ***myishere***.

format.sh
Shell script que permite comprobar si los ficheros de entrega cumplen las normas especificadas de formato.

1.3. Corrector de formato proporcionado

Se proporciona a los alumnos el shell script **format.sh** que verifica que el formato del entregable de la práctica es el correcto (sigue las especificaciones de nombrado, y está bien comprimido). También realiza una prueba básica de formato para aquellos programas solicitados en la práctica que muestran algo por consola. El script del corrector contiene comentarios que describen su funcionamiento. El corrector se deberá ejecutar en las máquinas Linux de las aulas informáticas de la universidad.

El comando para ejecutar el corrector es el siguiente:

sh format.sh <entregable.zip>

Siendo *entregable.zip* el fichero que se va a entregar por aula global (ver siguiente apartado). Ejemplo:

```
$ sh format.sh ssoo_p1_100254896_10004714.zip
```

El corrector imprimirá mensajes por pantalla indicando si el formato es o no correcto para cada punto revisado.

2. Entrega

2.1. Plazo de entrega

La fecha límite de entrega de la práctica se podrá consultar en AULA GLOBAL.

2.2. Procedimiento de entrega de las prácticas

La entrega de las prácticas ha de realizarse de forma electrónica. En AULA GLOBAL se habilitarán unos enlaces a través de los cuales 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.zip donde A…A y B…B 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**
- **myenv.c**
- **myishere.c**

Para comprimir mediante línea de comandos, se puede utilizar el comando:

```
zip ssoo_p1_AAAAAAAAA_BBBBBBBBB Makefile mywc.c myenv.c myishere.c
```

La memoria se entregará en formato PDF en un fichero llamado **ssoo_p1_AAAAAAAAA_BBBBBBBBB.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:

- Que un programa compile correctamente y sin advertencias (*warnings*) no es garantía de que funcione correctamente.
- 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 la memoria de la práctica (entregador TURNITIN).

NOTA: La única versión registrada de su práctica es la última entregada. La valoración de esta es la única válida y definitiva.

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) **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) obtendrán una calificación de cero en la práctica, siendo incompatible con el seguimiento de la evaluación continua por normativa.**

3) **Los programas deben compilar sin warnings. De lo contrario, se penalizará la nota de la práctica.**

4) **Un programa no comentado, obtendrá una calificación de 0.**

5) **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.**

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

7) **Se debe realizar un control de errores en cada uno de los programas.**

8) **Una práctica que no supere el corrector de formato recibirá una penalización en la nota.**

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, into 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 en n 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 `read` o `write` 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 la siguiente entrada de un directorio (cuyo descriptor de directorio *dirp* se obtuvo al abrirlo. La estructura *dirent* contiene un campo *d_name* (*char * d_name*) con el nombre de la entrada y un capo *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 2 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`)