Área de Arquitectura y Tecnología de Computadores

Universidad Carlos III de Madrid

SISTEMAS OPERATIVOS

Práctica 2. Gestión de procesos

Grado de Ingeniería en Informática

Curso 2013/2014

Índice

1. Introducción

El objetivo de esta práctica es mostrar al alumno una visión práctica de cómo se lleva a cabo la gestión de procesos, la comunicación entre ellos y con el sistema operativo. Existen llamadas al sistema que proporcionan las funciones necesarias para la creación, comunicación y finalización de los procesos. A lo largo de esta práctica, el alumno debe demostrar que es capaz de hacer un buen uso de este conjunto de llamadas dentro de su programa.

Para llevar a cabo la verificación de conocimientos, el alumno debe desarrollar un programa que, dada la entrada introducida por teclado, ejecute un comando u otro. Los comandos que deben ejecutarse en función del parámetro introducido por teclado ya están predefinidos y el usuario podrá elegir cual desea ejecutar en cada momento. Una vez finalizada la ejecución de un comando, el programa volverá a solicitar al usuario otro comando para ejecutar. Esto será de así hasta que el usuario escriba la palabra "exit".

Los comandos que debe ofrecer el programa deben ser implementados empleando llamadas al sistema. Cada comando puede componerse de una o más tareas. Para ejecutar estas tareas el alumno debe crear los procesos que estime necesario y comunicarlos para obtener el resultado indicado. Para llevar a cabo esta funcionalidad, puede emplear las siguientes llamadas al sistema:

- Para crear nuevos procesos y ejecutar diferentes comandos, el alumno puede emplear **fork** y **excepvp**.

- Las tuberías (**pipe, dup** y **close**) serán utilizadas por el alumno para comunicar los distintos procesos creados entre sí.

- Si el usuario requiere de alguna redirección de entrada o salida serán tratadas con **open, dup** y **close.**

- El alumno usará **wait** y **waitpid** para que el proceso principal (main) espere hasta la finalización de los procesos creados antes de indicarle al usuario que puede ejecutar otro comando.

- Para manejar y capturar una señal determinada, el alumno usará **signal.**

A continuación se procederá a la descripción de cada uno de los comandos que el alumno debe ejecutar y las llamadas al sistema necesarias en cada una de ellos.

1.1. Descripción

La práctica ha sido diseñada para que el alumno la realice de forma incremental. En primer lugar el alumno debe realizar un ejercicio 0 y, una vez terminado, se pondrá con el uno, luego con el dos y así hasta finalizar todos los ejercicios planteados. Toda la práctica se realizará obre el archivo *minishell.c*. Este código al finalizar la práctica debe contener las funciones necesarias para ejecutar desde el núcleo o ejercicio 0 hasta el último comando.

1.1.0. Ejercicio 0: núcleo de la minishell

El alumno debe empezar su minishell creando un bucle infinito. Por cada ejecución del bucle se deberá mostrar por pantalla el prompt de la minishell ("*minishell*> ") y esperar a que el usuario introduzca una línea por teclado. Si la línea introducida por el usuario es "*exit*", la minishell deberá finalizar su ejecución retornando un 0. Si por el contrario es cualquier otra cosa, el programa no realizará nada y esperará por otra línea. A continuación se muestra un ejemplo de ejecución del núcleo de la minishell:

```
user@ubuntu:./minishell
minishell>do-something
minishell>do-other-thing
minishell>exit
user@ubuntu:
```

Notas:

- Se pueden utilizar las funciones *scanf* (de stdio.h) y *strcmp* (de string.h) para leer y comparar la línea que introduzca el usuario.
- El programa debe imprimir el prompt ("*minishell*>")
- El programa debe ser capaz de aceptar un número ilimitado de entradas.
- El programa retornará 0 si el usuario introduce "*exit*".

1.1.1. Ejercicio 1: ejecución de un comando simple

Si el usuario introduce por teclado "***command1***", la minishell debe ejecutar el comando ***uptime***. Este comando imprime por consola información sobre cuánto tiempo lleva la máquina encendida.

```
minishell>command1
15:31:00 up 1 day, 22:31,  3 users,  load average: 0.00, 0.01, 0.05
minishell>
```

Notas:

- Use *fork* para crear un nuevo proceso.
- Use *execvp* en el código del proceso hijo para ejecutar el comando ***uptime.***
- Use *wait* o *waitpid* en el padre para esperar hasta la finalización del comando.
- El programa debe retornar -1 si existe algún problema realizando cualquiera de las llamadas al sistema.

1.1.2. Ejercicio 2: ejecución de un comando simple con argumento

Si el usuario introduce por teclado "***command2***", la minishell debe ejecutar el comando ***uname –a.*** Este comando imprime por pantalla la información acerca de la distribución y versión del kernel en ejecución.

```
minishell>command2
Linux ubuntu 3.2.0-57-generic #87-Ubuntu SMP Tue Nov 12 21:35:10 UTC 2013 x86_64 x86_64 x86_64 GNU/Linux
minishell>
```

Notas:

- Use *fork* para crear un nuevo proceso
- Use *execvp* en el código del proceso hijo para ejecutar el comando ***uname -a***
- Use *wait* o *waitpid* en el padre para esperar hasta la finalización del comando.
- El programa debe retornar -1 si existe algún problema realizando cualquiera de las llamadas al sistema.

1.1.3. Ejercicio 3: ejecución de un comando simple con redirección de salida

Si el usuario introduce por teclado "***command3***", la minishell debe ejecutar el comando ***cat /proc/cpuinfo*** **>** **<*output_file*>**. Este comando debe imprimir en el fichero indicado la información sobre la CPU.

minishell>command3 output_file.txt *minishell>* *-- The output from /proc/cpuinfo will have been saved to output_file.txt --*

Notas:

- El usuario debe introducir el nombre del fichero de salida en la misma línea que el comando. Se puede invocar la función scanf dos veces: la primera obtendrá el nombre del comando y la segunda obtendrá el nombre del fichero.
- La salida del comando ***cat /proc/cpuinfo*** debe ser redirigida al fichero. Se deben emplear para ello las funciones *open, dup* y *close.*
- Use *wait* o *waitpid* en el padre para esperar hasta la finalización del comando. El proceso principal de la minishell realizará el *fork* de todos los procesos hijos.
- El programa debe retornar -1 si existe algún problema realizando cualquiera de las llamadas al sistema.

1.1.4. Ejercicio 4: ejecución de un comando simple con redirección de entrada

Si el usuario introduce por teclado "***command4***", la minishell debe ejecutar el comando ***grep "model name" < <input_file>***. Este comando filtrará la información de la CPU mostrando sólo el modelo de procesador. En el ejemplo que se muestra a continuación, se ha introducido como fichero de entrada el generado con el ejercicio 3.

```
minishell>command4 input_file.txt                    -- Outputted file from the previous exercise --
model name      : Intel(R) Core(TM) i5 CPU      760  @ 2.80GHz
model name      : Intel(R) Core(TM) i5 CPU      760  @ 2.80GHz
model name      : Intel(R) Core(TM) i5 CPU      760  @ 2.80GHz
model name      : Intel(R) Core(TM) i5 CPU      760  @ 2.80GHz
minishell>
```

Notas:

- El usuario debe introducir el nombre del fichero de entrada en la misma línea que el comando. Se puede invocar la función scanf dos veces: la primera obtendrá el nombre del comando y la segunda obtendrá el nombre del fichero.
- Se debe redirigir la entrada del comando, para ello se deben emplear las funciones *open*, *dup* y *close.*
- Use *wait* o *waitpid* en el padre para esperar hasta la finalización del comando. El proceso principal de la minishell realizará el *fork* de todos los procesos hijos.
- El programa debe retornar -1 si existe algún problema realizando cualquiera de las llamadas al sistema.

1.1.5. Ejercicio 5: comandos con tuberías

Si el usuario introduce por teclado "***command5***", la minishell debe ejecutar el comando ***cat /proc/cpuinfo | grep "model name"***. Este comando realizará lo mismo que el anterior pero sin utilizar un fichero de entrada.

```
minishell>command5
model name      : Intel(R) Core(TM) i5 CPU      760  @ 2.80GHz
model name      : Intel(R) Core(TM) i5 CPU      760  @ 2.80GHz
model name      : Intel(R) Core(TM) i5 CPU      760  @ 2.80GHz
model name      : Intel(R) Core(TM) i5 CPU      760  @ 2.80GHz
minishell>
```

Notas:

- Utilizar una tubería para redirigir la salida de la primera parte del comando a la entrada de la segunda parte del comando (*pipe*, *dup* y *close*).
- Utilizar *wait* o *waitpid* en el código del padre para esperar hasta que el comando finalice. El proceso principal de la minishell realizará el *fork* de todos los procesos hijos.
- El programa debe devolver **-1** si hay algún problema en cualquiera de las llamadas al sistema.

1.1.6. Ejercicio 6: comandos con tuberías, redirección de la salida

Si el usuario introduce por teclado "***command6***", la minishell debe ejecutar el comando ***du –ha . | grep 4\.0K > <output_file>***. La minishell debe solicitar al usuario el fichero de salida antes de ejecutar el comando. Este comando busca ficheros de 4KB o menos en el directorio actual (y sus subdirectorios).

minishell>command6 output_file.txt *minishell>* *-- The output from du –ha . | grep 4\.0K will have been saved to output_file.txt --*

- Utilizar una tubería para redirigir la salida de la primera parte del comando a la entrada de la segunda parte del comando (*pipe*, *dup* y *close*).
- El usuario debe introducir el nombre del fichero de salida en la misma línea que el comando. Se puede invocar la función scanf dos veces: la primera obtendrá el nombre del comando y la segunda obtendrá el nombre del fichero.
- La salida de la segunda parte del comando se redirigirá a dicho fichero. Se debe utilizar las funciones *open*, *dup* y *close*.
- Utilizar *wait* o *waitpid* en el código del padre para esperar hasta que el comando finalice. El proceso principal de la minishell realizará el *fork* de todos los procesos hijos.
- El programa debe devolver **-1** si hay algún problema en cualquiera de las llamadas al sistema.

1.1.7. Ejercicio 7: tres comandos con tuberías, redirección de la salida y de error

Si el usuario introduce por teclado "***command7***", la minishell debe ejecutar el comando ***du –ha . | grep 4\.0K | cut –f2 > <output_file> &> <error_file>.*** La minishell debe solicitar al usuario dos nombres de fichero antes de ejecutar el comando. La primera vez solicitará el fichero de salida y la segunda vez el fichero de error. Este comando busca ficheros de 4KB o menos en el directorio actual (y sus subdirectorios). Solo se imprimirá el nombre del fichero.

minishell>command7 output_file.txt error_file.txt

minishell>

-- The output from du –ha . | grep 4\.0K | cut –f2 will have been saved to output_file.txt --

-- The error from du –ha . | grep 4\.0K | cut –f2 will have been saved to error_file.txt --

Notas:

- Utilizar dos tuberías
  - La primera para redirigir la salida de la primera parte del comando a la entrada de la segunda parte del comando (*pipe*, *dup* y *close*).
  - La segunda para redirigir la salida de la segunda parte del comando a la entrada de la tercera parte del comando.
- El usuario debe introducir el nombre del fichero de salida y del fichero de error en la misma línea que el comando. Se puede invocar la función scanf tres veces: la primera obtendrá el nombre del comando, la segunda obtendrá el nombre del fichero de salida y la tercera obtendrá el nombre del fichero de error.
- La salida de la tercera parte del comando se redirigirá al fichero de salida. Se debe utilizar las funciones *open*, *dup* y *close*.
- La salida de de error de todas las partes del comando se redirigirán al fichero de error. Se debe utilizar las funciones *open*, *dup* y *close*.
- Utilizar *wait* o *waitpid* en el código del padre para esperar hasta que el comando finalice. El proceso principal de la minishell realizará el *fork* de todos los procesos hijos.
- El programa debe devolver **-1** si hay algún problema en cualquiera de las llamadas al sistema.

1.1.8. Ejercicio 8: tres comandos con tuberías, redirección de la salida y ejecución en segundo plano

Si el usuario introduce por teclado "***command8***", la minishell debe ejecutar el comando ***du –ha . | grep 4\.0K > <output_file> &***. Este commando es el mismo que el del ejercicio 6, pero debe ser ejecutado en Segundo plano (***background***). Esto significa que la minishell no espera a la finalización del comando para solicitar un nuevo comando inmediatamente tras la creación del proceso.

```
minishell>command8 output_file.txt
[3459]
minishell>                    -- command is currently on execution while minishell is asking for another command --
```

Notas:

- Utilizar una tubería para redirigir la salida de la primera parte del comando a la entrada de la segunda parte del comando (*pipe*, *dup* y *close*).
- El usuario debe introducir el nombre del fichero de salida en la misma línea que el comando. Se puede invocar la función scanf dos veces: la primera obtendrá el nombre del comando y la segunda obtendrá el nombre del fichero.
- La salida de la segunda parte del comando se redirigirá a dicho fichero. Se debe utilizar las funciones *open*, *dup* y *close*.
- NO UTILIZAR *wait* o *waitpid* en el código del padre para esperar hasta que el comando finalice. El proceso principal de la minishell realizará el *fork* de todos los procesos hijos.
- En lugar de realizar la espera, se debe imprimir el PID del proceso que ejecuta la última parte del comando (*grep*) utilizando el siguiente formato: ***"[%d]\n"***, e inmediatamente después solicitar al usuario el siguiente comando a ejecutar.
- El programa debe devolver **-1** si hay algún problema en cualquiera de las llamadas al sistema.
- Los procesos zombie que queden tras las ejecuciones en background serán recogidos por la minishell cuando ejecuten otro comando en primer plano.

1.1.9. Ejercicio 9: manejo de señales

Si el usuario pulsa ***Ctl+Z*** durante la ejecución de un comando, la minishell debe imprimir el mensaje: ***Executing command <name of the command>.***

minishell>command7 output_file.txt error_file.txt	
	-- executing… --
^Z	*-- Ctrl-Z pressed --*
Executing command command7	
	-- executing… --

Notas:

- Crear una variable que almacene el comando actual en ejecución. Actualizar dicha variable cada vez que se ejecuta un comando.
- Utilizar *signal* para configurar una función como manejador de la señal SIGTSTP.
- Imprimir el mensaje requerido en dicha función.

1.2. Código inicial

Para facilitar la realización de esta práctica se dispone del fichero p2_process_2014.zip que contiene código fuente de apoyo. Para extraer su contenido ejecutar lo siguiente:

```
unzip p2_process_2014.zip
```

Al extraer su contenido, se crea el directorio *p2_process_2014/*, 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.

minishell.c

Fichero de código fuente en el que se debe codificar la ***minishell***.

format.sh

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

1.3. Comprobador de formato

El script **format.sh** verifica que la entrega del alumno sigue estos requisitos:

- Requisitos sobre el nombre del fichero a entregar.
- Formato de compresión y estructura del fichero a entregar.
- Ausencia de errores de compilación.

Para obtener más información sobre los contenidos del fichero format.sh puede ser abierto y analizado. Es obligatorio utilizar este comprobador de formato. El comando para ejecutarlo es el siguiente (se debe ejecutar en Linux):

sh format.sh <fichero_a_entregar.zip>

Ejemplo:

```
$ sh format.sh ssoo_p2_100254896_10004714.zip
```

El comprobador de formato imprimirá por pantalla mensajes acerca de si el formato requerido se cumple o no.

2. Entrega

2.1. Plazo de entrega y procedimiento

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

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.

2.2. Ficheros a entregar

Se debe entregar un archivo comprimido en formato zip con el nombre ssoo_p2_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_p2_AAAAAAAAA.zip. El archivo zip se entregará en el entregador correspondiente al código de la práctica. El archivo debe contener:

- **Makefile**
- **minishell.c**

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

```
zip ssoo_p2_AAAAAAAAA_BBBBBBBBB minishell.c
```

La memoria se entregará en formato PDF en un fichero llamado **ssoo_p2_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) **El entorno de pruebas de cara la evaluación de la práctica será guernika. Asegúrese de que su práctica funciona correctamente en guernika mediante ssh o bien en los laboratorios del departamento de informática.**

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

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

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

6) **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.**

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

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

9) **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. 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`)