Sistemas Operativos
Práctica 2: Mini Shell

Juan Banga Pardo
100383373@alumnos.uc3m.es

Alejandro de la Cruz Alvarado
100383497@alumnos.uc3m.es

7 de abril de 2019

Índice

1. Descripción del código

El código se se estructura en funciones (para reducir el main ya que estas funcionalidades se repiten varias veces) y el main en sí.

1.1. Funciones

1.1.1. Background

Background es la función destinada a en caso de que el parser haya detectado un '&' y cambiado el valor de la variable bg a 1 imprima el pid del proceso[1] y no le espere, en caso contrario debe esperar a todos los hijos existentes.

1.1.2. Redirect

Redirect, como su propio nombre indica, redirige las salidas/entrada estándar a los descriptores de fichero proporcionados que el propio parser utiliza. Tiene varias formas: si el parámetro introducido es 0 sólo redirige la entrada (usado en la primera pipe), si es 1 sólo la salida y salida de errores (para la última pipe) y si es 2 redirige todo, usado en el resto del código.

1.1.3. MyPwd

MyPwd es la encargada de imprimir el directorio actual, esto lo hace con la función getcwd(), utilizada en la práctica anterior.

1.1.4. MyTime

Y por último MyTime imprime el tiempo que ha necesitado para finalizar la ejecución de un mandato. Para ello toma el tiempo al inicio de la ejecución con la función gettimeofday(), hace un fork y ejecuta como un sólo comando (explicado más tarde) y al final vuelve a tomar el tiempo y devuelve el double resultante de la operación final - principio y pasándolo a segundos. A la hora de ejecutar el comando hemos tenido que desplazar el puntero del argvv[0] en una posición para que empiece a ejecutar pasado 'mytime'.

[1]Esta funcionalidad está comentada en el código, ya que aunque en el enunciado se pide que se imprima el pid de los procesos en background el corrector, sin embargo, busca que no se imprima nada. Para ver tal impresión descomentar la línea 26.

1.2. Main

Lo primero que hacemos es detectar los comandos especiales (mypwd y mytime) con la función strcmp(), en caso afirmativo llamamos a su respectiva función.
Después diferenciamos si ha habido un comando o más de uno. En caso de uno solo primero hacemos el fork para crear el hijo que ejcutará el comando, redireccionando con la función ya comentada y utilizando un exec para ejecutar el comando introducido, también comprobamos si dicha ejecución tiene que ser en modo background.
Existe la posibilidad de que el usuario haya introducido varios comandos con pipes, ésto lo solventamos creando una matriz de pipes[2], hay tantas como número de mandatos menos uno (ya que el último escribe en la salida estándar directamente).
Lo siguiente que hacemos es, dentro de un for que recorra todos los mandatos, crear la pipe correspondiente con la función pipe() y marcar sus descriptores como cerrables, lo hacemos con la función fcntl() y el flag O_CLOEXEC, esta flag hace que cuando se haga un exec se cierren automáticamente esos desciptores (ahorrándonos así posibles olvidos al hacerlo a mano.
Diferenciamos primer, último y procesos del medio, ya que tienen diferentes tareas: el primero redirecciona la entrada y el último las salidas. Por otro lado, el primero escribe en su pipe, los de en medio leen de la pipe anterior y escriben en su pipe y el último lee de la pipe anterior y escribe en pantalla.
Después de hacer establecer los enlaces con las pipes ejecutamos los comandos utilizando el exec de un comando pero con el correspondiente (usando la variable iteradora del bucle).
Por último identificamos el pid del proceso msh antes del for con la función getpid() y al final de la ejecución del comando el padre tiene que cerrar todos los descriptores de las pipes a mando, no se puede aprovechar el O_CLOEXEC porque el padre no va a hacer ningún exec y de no hacerlo a mano se quedarían abiertos y el programa no podría acabar.
Al igual que con un mandato también comprobamos si la ejecución debe ser en modo background.

[2]Somos conscientes de que se puede hacer con únicamente dos pipes alternándolas entre procesos pares e impares por ejemplo, pero así lo pensamos en un principio y no hemos querido cambiarlo porque funciona bien y probablemente nos hubiera causado más problemas que soluciones cambiar el modelo.

2. Batería de pruebas

Datos	Expectativa	Resultado
Comando de un solo argumento (ls)	ok	ok
Comando de dos argumentos (ls -la)	ok	ok
Comando de tres argumentos (ls -l -a)	ok	ok
Comando incorrecto (lda)	error	error
Comando en background (okular test.txt&)	ok	ok
Comando en background (ls&)	ok	ok
Comando en background (sleep 1&)	ok	ok
Ejecutar otro comando mientras está otro en background	ok	ok
Ejecutar background sin comando (&)	error	error
Ejecución normal de mypwd	ok	ok
Añadir argumentos a mypwd	error	error
Ejecución de mytime en un comando sin espera (ls)	ok	ok
Ejecución de mytime en un comando con espera (sleep 1)	ok	ok
Ejecución del mytime con mucha espera (sleep 100)	ok	ok
Ejecución del mytime con poca espera (sleep 0.1)	ok	ok
Ejecución del mytime con un comando iventado (test)	error	ok
Una pipe (ls | grep test)	ok	ok
Dos pipes (ls | grep test | grep test02)	ok	ok
Diez pipes	ok	ok
Pipes en background	ok	ok
Pipes con comandos no existentes	error	error
Ctrl+D	salida del programa	salida del programa

*El programa compila perfectamente sin ningún tipo de warning.

3. Conclusiones

Tanto a mi compañero como a mí nos ha gustado esta práctica, ya que al igual que la primera, al ser usuarios de Arch Linux es una gran oportunidad recrear una terminal que usamos en el día a día. Nos hubiera gustado hacer un producto más çompletoçomo modificando el prompt, añadiendo atajos de teclado como Ctrl+L para borrar la pantalla o las flechas para acceder a los comandos previos en el bash_history, pero es algo que tendremos que hacer por nuestra cuenta en algún momento para seguir profundizando.

La práctica la hicimos con bastante rapidez, los únicos aspectos que nos causaron problema fueron el mytime a la hora de desplazar el puntero de ejecución para obviar 'mytime' y el uso de las pipes ya que al principio lo habíamos hecho mal pero de casualidad funcionaba y cuandob nos pusimos a arreglarlo para que fuera correcto nos encontramos con bastantes problemas a la hora de redirigir los descriptores de fichero.