SISTEMAS OPERATIVOS

GRADO EN INGENIERÍA INFORMÁTICA

Práctica 2: Minishell

Curso 2019/2020

Jorge Rodríguez Fraile, 100405951, Grupo 81, 100405951@alumnos.uc3m.es
Carlos Rubio Olivares, 100405834, Grupo 81, 100405834@alumnos.uc3m.es

Índice

Descripción de código

Codificación para leer comandos únicos

Para esta implementación hemos empezado controlando que no se pasa el nº máximo de comandos mediante la constante MAX_COMMANDS, y si no se pasa de este, los imprimimos con sus extensiones.

Si el command_counter es 1, entramos a la parte de código que realiza esta función. Empezamos haciendo un fork para hacer que el hijo realice la función, se controla que pueda dar fallo con su correspondiente perror y después mediante el valor que nos devuelve el fork diferenciamos lo que hará el padre y el hijo; el hijo ejecutará la función mediante el execvp y el padre esperará a que la función termine mediante wait, pero sólo si el comando no se ejecuta en background. Por otro lado también se comprueba si el hijo ha ejecutado correctamente mediante otro perror.

Mandatos conectados por pipes

Para este apartado empezamos creando nuestro descriptor de fichero para poder crear la pipe, y una variable status para ir comprobando la ejecución de todos los hijos que vayamos creando.

Una de las partes más importantes de la parte de este código es la variable in, que irá siendo el puntero de salida del proceso anterior, que se convertirá en el de entrada del proceso siguiente. Al inicio tendrá el valor de la entrad estándar, como es obvio.

Después de esto, se empieza creando un bucle que recorra todos los comandos y creando las pipes, solo que en el último proceso no se creará un pipe (nº pipes = nº procesos-1)

Una vez creada la pipe, se hace un fork para para establecer las funciones que realizarán el padre y el hijo, que serán parecidas a las del apartado anterior, solo que con algunas modificaciones de las direcciones de entrada/salida para poder conectar las pipes.

Si el fork tiene valor 0 significa que estamos en el proceso hijo, lo que hará será cerrar su entrada por defecto y se sustituirá por la de in (la salida del proceso anterior, o entrada estándar en el caso de la primera iteración) hecho esto liberará la entrada de in ya que ya está asignada como entrada estándar

Si una vez hecho esto no estamos en la última iteración eso significa que tenemos que cerrar nuestra salida estándar y poner la de salida de la pipe, al terminar estos cambios, ejecutamos el comando que se haya leído.

En el caso del proceso padre, empezamos liberando in ya que lo dejaremos de usar, y, en el caso de no ser el último proceso, haremos que in tome el valor de la entrada de la pipe que se crea, para que el proceso hijo puede tomar los datos de salida de datos del padre como entrada.

Una vez acabado el bucle, el primer proceso esperará al último hijo que irá despertando a los demás procesos, todo esto hecho mediante un while.

Mandatos simples y secuencias de mandatos con redirecciones

Primero, hemos implementado las redirecciones en mandatos simples (como es obvio) para ello, hemos comprobado si filev[1] no esta vacío, si este es el caso, significa que tenemos que hacer una redirección de salida, por tanto, cerramos el descriptor de salida estándar y creamos el fichero si no existe,o lo abrimos en su caso. Haremos lo mismo para filev[0] y filev[2], que se corresponde a redireccionamiento de entrada y error respectivamente.

Para hacerlo con múltiples mandatos haremos el mismo código, solo que la redirección de entrada se podrá hacer únicamente si es el primer proceso que se crea y la redirección de salida si es el último proceso de la pipe. La redirección de error se puede hacer en cualquier proceso.

Mandatos internos

mycalc

Para este mandato hemos creado otra condicional al nivel de los de command_counter para que si se lee como primer argumento en la array arg_execvp mycalc se ejecute el código correspondiente a este.

Para empezar vemos si el comando tiene el formato adecuado, y una vez hecho esto diferenciamos si el argumento usado ha sido add o mod.

Si ha sido add, guardamos nuestros operandos en dos variables x e y con un cast utilizando atoi, sumamos las dos y mostramos el resultado, que se guardará en nuestra variable acumulativa ACC.

Si en cambio ha sido mod, guardamos los operandos en otras dos variables y hacemos la operación que se nos pide, solo que en este caso el módulo o resto no se sumará a nuestra variable acumulativa.

mycp

En mycp, empezamos como en mycalc, comprobando que se ha leído el mandato y está guardado en la primera posición de argv_execvp. Si se da el caso, abrimos el fichero del que copiaremos y controlamos su error si no existe o o se ha abierto correctamente. Guardamos el indicador de fichero en una variable y creamos un buffer de 1024 bytes y abrimos (o creamos) el fichero destino, donde también guardaremos su indicador en una variable.

Una vez hecho esto, creamos un bucle para leer del fichero a nuestro buffer, y dentro de este bucle, otro más para trasladar los datos del buffer al fichero destino. La implementación es muy parecida al método mycat de la primera práctica, solo que, en vez de que los datos vayan del buffer a la salida por defecto, van al descriptor del fichero destino que habíamos guardado en una variable, al terminar, cerramos los dos ficheros para liberar recursos.

Pruebas realizadas

Codificación para leer comandos únicos

Introducir un mandato correcto, entrada myls, esperamos que realice la operación y vuelva a permitirnos introducir otro mandato. El resultado de la prueba es el esperado, imprime lo que debe la función y vuelve a aparecer el MSH para introducir otra.

Introducir un mandato que no exista, entrada salto, esperamos que nos permita introducir otro mandato. Ocurre lo esperado, no hace la operación que no existe y nos vuelve a aparecer MSH que nos permite continuar con el minishell.

Introducir un mandato simple que queremos ejecutar en background, probamos con un sleep 10 &, esperamos que el proceso se ejecute y mientras podemos hacer otras operaciones. Obtenemos el resultado esperado, el sleep se hace de fondo lo que no nos inutiliza la consola y podemos introducir nuevo comando que funcionan.

Mandatos conectados por pipes

Introducir un mandato correcto, entrada ls -l | sort, esperamos que un proceso le pase a otro la salida para que pueda hacer su funcionalidad con esos datos. El resultado obtenido es el esperado, aparece la salida del ls -l pero ordenada.

Introducir una secuencia de mandatos de longitud mayor que 2, esperamos que se pasen de unos a otros las salidas y todo quede conectado. Obtenemos el resultado esperado, la salida del mandato anterior pasa como entrada para el siguiente y este hace su función y la pasa al siguiente, hasta que llega al último y lo imprime por pantalla.

Mandatos simples y secuencias de mandatos con redirecciones

Introducir un fichero no existente con >, pero en este caso si se debe crear y que no salte ningún error. El resultado obtenido es el esperado crea el archivo y escribe en el.

Introducir un fichero no existente con <, pero en este caso salta un error al no recibir un archivo de entrada existente. El resultado obtenido es el esperado salta el error de archivo no encontrado.

En conexiones con pipes, introducir un símbolo de redireccionamiento > o < en el medio, con lo que saltaría error. Tras la prueba obtenemos errores al no estar permitido.

Introducir un elemento de redireccionamiento de salida (>) con el primer proceso de una pipe, entrada ls -l > exit.txt | sort | wc -l, esperamos que salte un error al no estar permitido que la salida sea a un fichero y no al pipe. Tras la prueba recibimos lo esperado, un error.

Introducir un elemento de redireccionamiento de entrada (<) con el último proceso de una pipe, entrada ls -l | sort | wc -l < input.txt, esperamos que ignore el archivo y ejecute con normalidad. Se recibe lo esperado tras la prueba de ls -l | sort | wc -l, ignorando la redirección de entrada.

Introducir una secuencia de mandatos conectados redirigiendo los errores a un fichero, esperamos que escriba lo mensajes de error en ese documento. La salida es la esperada, escribe el error y podemos seguir ejecutando comandos.

Mandatos internos

mycalc

Introducir menos parámetros, entrada mycalc 12 add, esperamos que nos salte el mensaje de formato incorrecto. El resultado es el esperado salta el error elegido para formato invalido

Introducir mas parametros, entrada mycalc 12 add 14 add 56, el error esperado es el de mensaje de formato invalido. El resultado obtenido es el esperado.

Introducir una operación que no sea add o mod, entrada mycalc 12 mul 14, se espera que salte el error de formato. Tras la prueba se obtiene el error esperado.

Introducir un formato válido de add, entrada mycalc 12 add 15, esperamos que devuelva 27 y como es la primera operación de acumulado también 27, todo ello en el mensaje correspondiente. Obtenemos [OK] 12 + 15 = 27; Acc 27 que es lo esperado.

Hacer una suma consecutiva, esperamos que acumule las sumas, tras la operación anterior hacemos mycalc 34 add 65, esperamos que lo sume y de acumulacion aparezca 125. Obtenemos [OK] 34 + 65 = 99; Acc 125 que es lo correcto.

Introducir un formato válido de mod, entrada mycalc 23 add 4, esperamos que devuelva que 23 es el producto de 4 por 3 más el resto 11, todo ello en el mensaje correspondiente. Obtenemos [OK] 23 % 4 = 4 * 3 + 11 qué es lo esperado.

mycp

Introducir un archivo que no exista, el resultado esperado es el mensaje de error [ERROR] Error al abrir el fichero origen : No such file or directory. Tras realizar la prueba se obtiene el mensaje esperado.

Introducir un archivo que del que no tenemos permiso de lectura. Esperamos que nos de el error al abrir el archivo. El resultado obtenido ha sido [ERROR] Error al abrir el fichero origen : No such file or directory, que es el esperado.

Introducir un archivo valido, le pasamos el nombre de un archivo que está en el directorio del que tenemos permisos de lectura y que contenga texto. Esperamos que nos copie el archivo pasado como primer parámetro en el pasado como segundo parámetro. El resultado obtenido es un archivo idéntico al original, pero con el nombre indicado.

Introducir una secuencia de mandatos en la que faltan parámetros, esperamos que nos de un error. Tras la prueba se ha obtenido el mensaje [ERROR] La estructura del comando es mycp.

Conclusiones

En esta práctica nuestra mayor dificultad ha sido a la hora de crear pipes, principalmente porque no controlamos muy bien cómo concatenar la salida de un proceso y llevarla a la entrada de otro sin saber su pid, pero gracias a los ejercicios mostrados en clase hemos podido darle otro encaje al problema y resolverlo. Por otra parte, la creación de mycalc ha sido algo difícil sobre todo a la hora de hacer que los mensajes salieran de la manera correcta y no simplemente con un printf.

Por último, hemos hecho muchas modificaciones a nuestro código inicial al intentar hacer todas las pruebas posibles, esto nos ha ayudado bastante a perfeccionar nuestro código y así poder entender mejor lo que estamos codificando.