Universidad
Rey Juan Carlos

Sistemas Operativos

[PRÁCTICA 2 - MINISHELL]

MARIO ROJAS PADRÓN, ANA CRISTINA ACOSTA HERNÁNDEZ

TABLA DE CONTENIDO

Autores

Mario Rojas Padrón

m.rojas.2019@alumnos.urjc.es

Ana Cristina Acosta Hernández

ac.acosta.2017@alumnos.urjc.es

Descripción del Código

Diseño del Código

El código está dividido en 3 archivos:

- test.c: contiene el main y es el que se encarga de todo el funcionamiento del código, así como de las llamadas a las librerías.
- libparser.a: contiene la función tokenize, que recibe un puntero a una cadena de caracteres y devuelve un puntero a tline con la información sobre dicha cadena.
- parser.h: contiene los tipos de datos tcommand y tline. Por un lado, tcommand se refiere a cada mandato, y de él se almacena la ruta del mandato, los argumentos y su número. Por otro lado, tline almacena una línea con mandatos, y de ella guardamos el número de comandos y un array con los mismos, las diferentes redirecciones que apuntarían a un nombre de fichero o a null, y un entero para saber si cada mandato se ejecuta en backgroung o foreground.

Ejecución de mandatos y gestión de tuberías

Para comunicar los diferentes procesos se ha empleado el uso de pipes y se han redirigido las entradas y salidas dependiendo del número de mandato.

Por un lado, al tratarse del primer mandato, comprobamos si tenía redirección de entrada desde un fichero, si no leíamos desde teclado, y su salida la redirigimos al descriptor de escritura del pipe.

A la hora de comunicar un proceso con su contiguo, se ha cerrado el extremo de lectura del primer proceso y el extremo de escritura del siguiente, y también así se ha redirigido la salida del primero a la entrada del segundo.

Finalmente, en el caso del último mandato, redirigimos su entrada estándar al extremo de lectura del pipe (que contiene la salida del anterior comando), observamos si tiene una redirección de salida a fichero o si no, mostramos su salida por pantalla.

Implementación del background

Para implementar el background, hemos creado una copia (hijo) del proceso principal con fork(); sin embargo, al ejecutarse una nueva minishell en background, la shell no espera a la finalización de los diferentes procesos para terminar. Por otro lado, se abre con el dispositivo /dev/null para que no haya conflictos con el background, ya que descarta toda la información que se escribe o se redirige hacia él.

Principales Funciones

	main	Nombre	Tipo	Descripción
Argumentos	Argumento 1			
Variables Locales	Variable 1	**created_process**	**Array**	**Crea un array para guardar los procesos**
	Variable 2	**line**	**tline ***	**Almacena los mandatos de una línea**
	Variable 3	**buffer**	**Array**	**Se van almacenando temporalmente los valores de una línea para ver si es válida**
Descripción de la Función	**Ejecuta el programa, comprueba los comandos recibidos**			

	prompt	Nombre	Tipo	Descripción
Argumentos	Argumento 1	**buffer**	**Array ***	**Se van almacenando temporalmente los valores de una línea para ver si es válida**
Variables Locales	Variable 1	**cwd**	**char ***	**Guarda la ruta absoluta del directorio actual**
	Variable 2	**isSucces**	**Bool**	**Devuelve -1 en caso de fallo y en otro caso el número de caracteres leídos**
Valor Devuelto		**isSuccess**	**Bool**	**Devuelve -1 en caso de fallo y en otro caso el número de caracteres leídos**
Descripción de la Función	**Cataloga un mandato como válido o no**			

	execute_command	Nombre	Tipo	Descripción

Argumentos	Argumento 1	**commands**	**tcommand ***	**Almacena la ruta del mandato, los argumentos y su número**
	Argumento 2	**ncommands**	**int**	**Almacena el número de comandos**
	Argumento 3	**redirect**	**CommandRedirect**	**Guarda las diferentes redirecciones**
Variables Locales	Variable 1	**in**	**int**	**Bucle que recorre los comandos**
	Variable 2	**fd**	**int**	**Almacena los descriptores de fichero**
	Variable 3	**pid**	**pid_t**	**Guarda el id de los diferentes procesos**
Valor Devuelto	**Bool**			
Descripción de la Función	**Redirige salidas y entradas de los procesos y sus pipes**			

	executeMyShellCommand	**Nombre**	**Tipo**	**Descripción**
Argumentos	Argumento 1	**argv**	**Array**	**Almacena el mandato**
Valor Devuelto	**Bool**			
Descripción de la Función	**Comprueba si el mandato introducido coincide con alguno de los de la propia minishell y lo ejecuta**			

	myShell_cd	**Nombre**	**Tipo**	**Descripción**
Argumentos	Argumento 1	**argv**	**Array**	**Almacena el mandato**

Variables Locales	Variable 1	**path**	**char ***	**Almacena el directorio al que se desea cambiar**
Valor Devuelto	**Bool**			
Descripción de la Función	**Cambia del directorio actual al especificado**			

	execute_line	**Nombre**	**Tipo**	**Descripción**
Argumentos	Argumento 1	**line**	**tline ***	**Almacena los mandatos de una línea**
Variables Locales	Variable 1	**redirect**	**CommandRedirect**	**Guarda las diferentes redirecciones**
	Variable 2	**i**	**size_t**	**Iterador que va guardando el espacio ocupado**
	Variable 3	**has_space**	**Bool**	**Devuelve true si el espacio ocupado es menor que el espacio total disponible**
	Variable 4	**pid**	**pid_t**	**Guarda el id de los diferentes procesos**
Valor Devuelto				
Descripción de la Función	**Ejecuta una línea dependiendo de si está en background o foreground**			

	insert_process_data	**Nombre**	**Tipo**	**Descripción**
Argumentos	Argumento 1	**list**	**ProcessesList ***	**Almacena la lista de procesos**

	Argumento 2	**elem**	**ProcessData**	**Contiene la información relativa a un proceso**
Variables Locales	Variable 1	**node**	**struct Node ***	**Contiene el proceso almacenado en cada nodo y los punteros al anterior y siguiente del mismo**
Valor Devuelto	**Bool**			
Descripción de la Función	**Añade un nuevo proceso a la lista de procesos**			

	remove_process_data_by_index	**Nombre**	**Tipo**	**Descripción**
Argumentos	Argumento 1	**list**	**ProcessesList ***	**Almacena la lista de procesos**
	Argumento 2	**index**	**size_t**	**Índice del proceso a borrar**
Variables Locales	Variable 1	**pAux**	**Node**	**Puntero auxiliar que señala al nodo que se quiere borrar**
	Variable 2	**isFound**	**Bool**	**Devuelve true si el índice del proceso coincide con alguno existente**
Valor Devuelto	**Bool**			
Descripción de la Función	**Elimina un proceso de la lista de procesos**			

La función remove_process_data_by_pid es estructural y funcionalmente igual a la mencionada anteriormente, solo que se borra el proceso por pid y no por índice. Esto es así porque el usuario puede elegir el proceso a borrar por su índice, sin embargo, internamente se borra por pid.

Casos de Prueba

Para comprobar el correcto funcionamiento del programa, hemos testeado varios casos con los comandos deseados escritos por el terminal y también redirigiendo ficheros a la entrada y salida estándar. Todos los casos pasaron correctamente, obteniendo los resultados esperados. Además, se hizo uso del debugger para comprobar que internamente las estructuras de datos estaban correctamente compuestas.

Comentarios Personales

Uno de los obstáculos encontrados al encararnos con la práctica fue la familiarización con los pipes y sus respectivas redirecciones. Al condensar numerosos conceptos y de tanta importancia en este proyecto, debíamos de tener claro cómo funcionaban para pasar a la implementación: sus redirecciones de entrada y salida, cuándo cerrar qué extremos dependiendo del caso, etc.

Por otro lado, podríamos recalcar la complejidad a la hora de programar el background, ya que empleamos bastante tiempo en él porque a la hora de ejecutar la minishell se nos cerraba directamente. Debugeando, nos fijamos en el error que se nos mostraba, errno(4), y fijándonos en la tabla de errores, vimos que correspondía con interrupción de llamada al sistema. Tras eso, nos dimos cuenta del fallo que estaba ocurriendo, así que implementamos la macro sa_restart, que hace que sigaction reinicie la operación cuando valga 1.

Sin duda, el uso del debugger ha sido un punto clave para poder resolver los distintos problemas que se nos estaban planteando. Tal vez es un aspecto al que no se le dedica tanto tiempo en las sesiones de clase, pero que sin duda es muy importante: nos ayuda a comprender mejor lo que estamos haciendo y, en la mayoría de las veces en las que ocurren fallos, nos ahorran mucho tiempo.

Ha sido una práctica muy extensa y de alta complejidad, por lo cual el tiempo requerido ha ido acorde a la dificultad de la misma. Esta práctica condensa básicamente todos los conceptos adquiridos en clase, por lo que llevar la asignatura al día, tanto teoría como ejercicios propuestos, era necesario para poder encararla. Asimismo, al ser un proyecto que engloba tantos conceptos, nos ha hecho aprender en el proceso, lo encontramos útil de cara al entorno laboral, y desde luego, ha hecho que no nos aburriésemos en ningún momento.