Sistemas Operativos

Practica 2 – MINISHELL

Paula Monrobel Ugidos

Mario Recio Montero

Universidad
Rey Juan Carlos

TABLA DE CONTENIDO

Autores

Mario Recio Montero

- Correo: m.recio.2020@alumnos.urjc.es
- DNI: 01191458N

Paula Monrobel Ugidos

- Correo: p.monrobel.2019@alumnos.urjc.es
- DNI: 53667478Z

Descripción del Código

Diseño del Código

El bucle principal del codigo es un while, que lee de entrada estandar los mandatos, y termina al ejecutar el mandato exit.

Los hijos y las pipes se crean en bucles for, y cada hijo ejecuta una función, donde cerrara las que no use, y ejecutará exec. El padre mientras tanto, esparará a los hijos, de forma bloqueante o no bloqueante, dependiendo de si el mandato es o no en background.

Al superar esta fase, comprueba si algun mandato en background terminó para mostrarlo y repite el while.

Para mover todas las variables de los procesos en brackground, cuando terminan o se ejecuta una orden fg, se ha utilizado un algoritmo para mover los datos de las variables una posición.

Para hacer las pipes, hemos creado un algoritmo que perimite cerrar las pipes que no van a usarse, dependiendo de la posición del hijo dentro del mandato.

Para implementar el background, nos hemos fijado en la terminal por defecto, para saber que mostrar. Con la función jobs, pudimos gestionar como se iban mostrando en nuestra shell, y haciendo modificaciones.

Para comprobar los mandatos en background, nos hemos servido principalmente del sleep, para comprobar que el madato terminaba trasncurrido el tiempo indicado, y que además se mostraba como terminado al introducir un nuevo mandato, y no se mostraba al ejecutar un nuevo jobs.

Las señales que hemos utilizado han sido principalmente para transmitir errores entre los hijos, y para redirijir señales que cerrarían lan shell, o dentendrían los procesos en background.

Hemos utilizado diversas estructuras para almacenar toda la información de los mandatos

-Array de pids para procesos en fg

-Array de punteros para los hijos de cada proceso en bg

-Array de strings para guardar los mandatos

-Array para guardar el numero de hijos terminados de cada proceso

-Array para guardar en numero de hijos de cada proceso

Principales Funciones

	Main	Nombre	Tipo	Descripción
Variables Locales	1	**hijosb**	**Pid_t ****	**Pid Hijos en Background**
	2	**Tmp_procesosb**	**Pid_t ****	**Variable para reasignar memoria**
	3	**Tmp_hijosb**	**Pid_t ***	**Variable para reasignar memoria**
	4	**pid**	**Pid_t**	**Pid del hijo que se acaba de crear**
	5	**pidterm**	**Pid_t**	**Variable para guardar el pid de forma temporal**
	6	**terminadosb**	**Int ***	**Numero de hijos en bg terminados**
	7	**Tmp_terminadosb**	**Int ***	**Variable para guardar los hijos terminados de manera temporal**
	8	**numcommandsb**	**Int ***	**Numero de comandos en bg del trabajo [x]**
	9	**Tmp_numcommandsb**	**Int ***	**Variable para reasignar memoria**
	10	**numcommands**	**Int ***	**Numero de comandos**
	11	**promt**	**Char ***	**Promt**
	12	**Tmp_trabajos**	**Char ****	**Variable para reasignar memoria**
	13	**trabajos**	**Char ****	**String de la cadena de mandatos**
	14	**pwd**	**Char ***	**Directorio de trabajo actual**
	15	**p**	**Int ****	**Array de Pipes**
	16	**Tem_p**	**Int ****	**Array de pipes temporal**
	17, 18, 19, 20	**SizePr, sizeH, sizeC, sizeP**	**Int**	**Tamaños**
	21	**contador**	**Int**	**Contador**

	22	**idproceso**	**Int ***	**Identificar de procesos en bg**
	23	**Tmp_idproceso**	**Int ***	**Variable para reasignar memoria**
	24, 25, 26, 27	**i, j, z, numhijos**	**Int**	**Contadores de for y while**
	28	**buf**	**Char ***	**Bufer para los comandos introducidos**
	29	**comprobacion**	**bool**	**Boolean para comprobaciones**
Descripción de la Función				**Proceso Principal de la Minishell**

	Command	**Nombre**	**Tipo**	**Descripción**
Argumentos	Argumento 1	**posicion**	**int**	**Posición del hijo en el mandato**
	Argumento 2	**numcommands**	**int**	**Numero de comandos del mandato**
	Argumento 3	**p**	**Int ****	**Array de pipes**
Variables Locales	Variable 1	**i**	**int**	**Contador del for**
Descripción de la Función				**Funcion para cerrar las pipes necesarias y realizar los execv de los hijos**

	execjobs	**Nombre**	**Tipo**	**Descripción**
Argumentos	Argumento 1	**trabajos**	**Char ****	**Trabajos en bg**
Variables Locales	Variable 1	**i**	**int**	**Contador del for**
Descripción de la Función				**Escribe por pantalla los trabajos que se esten realizando en background**

	execfg	**Nombre**	**Tipo**	**Descripción**
Argumentos	Argumento 1	**idproceso**	**Int ***	**Id del proceso**

	Argumento 2	**hijosB**	**Pid_t ****	**Hijos del mandato de bg**
	Argumento 3	**terminadosb**	**Int ***	**Procesos terminados en bg**
	Argumento 4	**numcommandsb**	**Int ***	**Numero de comandos del mandato de bg**
	Argumento 5	**trabajos**	**Char ****	**String de la cadena de mandatos**
Variables Locales	Variable 1	**i**	**int**	**Contador del for**
	Variable 2	**id**	**int**	
Descripción de la Función				**Función que realiza el mandato interno fg**

	execumask	**Nombre**	**Tipo**	**Descripción**
Variables Locales	Variable 1	**deffich**	**int**	**Mascara por defecto de los ficheros**
	Variable 2	**defcarp**	**int**	**Mascara por defecto de los directorios**
	Variable 3	**count**	**int**	**contador**
	Variable 4	**Erroneo**	**bool**	**Comprobacion**
Descripción de la Función				**Función que realiza el mandato interno umask**

	checkbg	**Nombre**	**Tipo**	**Descripción**
Argumentos	Argumento 1	**idproceso**	**Int ***	**Id del proceso en bg**
	Argumento 2	**contador**	**Void ***	**Contador de hijos del mandato**
	Argumento 3	**hijosb**	**Pid_t ****	**Pid de los hijos del mandato**
	Argumento 4	**terminadosb**	**Int ***	**Numero de hijos terminados de mandato**
	Argumento 5	**numcommmandsb**	**Int ***	**Numero de comandos del mandato**

	Argumentos 6	**trabajos**	**Char ****	**String de la cadena de mandatos**
Variables Locales	Variable 1	**i**	**int**	**Contador del while**
	Variable 2	**completados**	**int**	**Numero de hijos completados**
Descripción de la Función				**Comprueba si el proceso en background ha terminado y escribe por pantalla el mensaje correspondiente**

	quitarultimo	**Nombre**	**Tipo**	**Descripción**
Argumentos	Argumento 1	**hijosb**	**Pid ****	**Pid de los hijos en bg**
	Argumento 2	**idproceso**	**Int ***	**Id del proceso en bg**
	Argumento 3	**Terminadosb**	**Int ***	**Numero de hijos terminados**
	Argumento 4	**numcommandsb**	**Int ***	**Numero de comandos del mandato**
	Argumento 5	**trabajos**	**Char ****	**String de la cadena de mandatos**
Variables Locales	Variable 1	**j**	**int**	**Contador del for**
Descripción de la Función				**Elimina el ultimo trabajo en bg**

	quitarbg	**Nombre**	**Tipo**	**Descripción**
Argumentos	Argumento 1	**Hijosb**	**Pid ****	**Pid de los hijos en bg**
	Argumento 2	**posterm**		**Posicion del mandato terminado**
	Argumento 3	**Idproceso**	**Int ***	**Id del proceso que se elimina**
	Argumento 4	**terminadosb**	**Int ***	**Numero de hijos terminados en bg**
	Argumento 5	**numcommandsb**	**Int ***	**Numero de comandos de comandos en el mandato**

	Argumento 6	**trabajos**	**Char ****	**String de la cadena de mandatos**
Variables Locales	Variable 1	**i, j**	**int**	**Contador de for**
	Variable 2	**limite**	**int**	**Limite**
Descripción de la Función				**Elimina de jobs el mandato en bg terminado del bg y mueve el resto de trabajos que quedan**

Casos de Prueba

Para los mandatos en foreground, hemos probado principalmente el ls, para la redirección de salida, y wc para la de entrada. Mirando el fichero, podemos comprobar que el mandato está leyendo y escribiendo correctamente.

Para comprobar que funcionan los mandatos en background, nos hemos servido de las funciones jobs y fg.

Cada vez que se incluye un mandato en background, se comprueba que se muestra en el jobs, y cuando acaba se muestra que está hecho, y ya no aparece en el jobs.

Para el mandato cd, hemos incluido el directorio actual en el promt, comprobando que se cambia al ejecutar la función.

Para comprobar que funcionan las pipes, hemos probado comandos como ls | wc, para comprobar que realmente lee lo que devuelve ls.

Para comprobar que funcionan las redirecciones, hemos usado un fichero de pruebas, para ir comprobando que leía y escribía donde se le redireccionaba.

Para comprobar el umask, hemos llamado a la función, después creado archivos y visto sus permisos con ls –l, comprobando que eran los adecuados. Además, debe mostrar el umask introducido en último lugar al ejecutarlo sin parámetros.