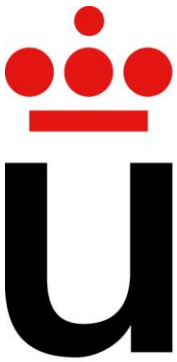

Universidad
Rey Juan Carlos

Sistemas Operativos

[PRÁCTICA II - MINISHELL]

DANIEL SANTOS LÓPEZ / DIEGO SÁNCHEZ RINCÓN

TABLA DE CONTENIDO

Autores

Grado en Ingeniería Software – 2024/25

Nombre	GitHub	LinkedIn
Daniel Santos López	danisntoss	Daniel Santos López
Diego Sánchez Rincón	CuB1z	Diego Sánchez Rincón

Descripción del Código

Diseño del Código

Estructuras

Estructura tjob

- Id: entero que guarda el identificador del job.
- Status: entero que guarda 0 si el proceso esta pausado o 1 si está en ejecución.
- Line: de tipo tline, guarda la línea parseada que ha introducido el usuario.
- Pids: array que guarda el id de los distintos procesos que componen el job.
- Pipes: array que guarda los distintos pipes que está utilizando el job.
- Command: guarda la línea en una cadena tal cual la introduce el usuario.
- Background: entero que guarda 0 si el proceso está en Foreground o 1 si está en Background.

Variables Globales

- MAX_COMMANDS: Constante precompilada que guarda el número máximo de jobs simultáneos.
- Jobs (tjob * [MAX_COMMANDS]): array de jobs de tamaño MAX_COMMANDS.
- Count (int): entero que guarda el número de Jobs ejecutados por el usuario, también es empleado para asignar el id de manera autoincremental.
- BgJobs (int): entero que contiene el total de procesos en background.
- StoppedJobs (int): entero que contiene el total de procesos pausados.
- LastStoppedJobId (int): entero que contiene el id del último proceso pausado.

Programa Principal

Variables de la función:

- Line (tline *): estructura que contiene una línea parseada introducida por el usuario.
- I (int): variable auxiliar para iterar.
- AllowExit (int): variable entera binaria que indica si un usuario puede terminar la ejecución mediante exit.
- Buffer (char *): variable auxiliar para leer entrada de teclado.

Reservamos memoria de forma dinámica para inicializar los Jobs mediante la función initializeJob, que se encarga de reservar una memoria inicial para el array de pipes y de pids mediante la función malloc.

Asignamos las funciones manejadoras CtrlC y CtrlZ a las señales SIGINT y SIGTSTP, modificando de su funcionamiento.

Asignamos la función manejadora TerminatedChildHandler a la señal SIGCHLD, que se lanza cuando termina un hijo, comprueba si todos los procesos de ese job han terminado y así marcarlo como terminado.

Limpiamos la pantalla y comienza el bucle principal del programa:

- Guardamos la línea leída en el buffer y pasamos este por la función tokenize.
- Comprobamos si hay errores o comandos vacíos en la línea.
- Ahora gestionamos el comando, según sea interno o externo, si es interno (cd, exit, umask, Jobs, bg) llamamos a la función correspondiente que resuelve este comando, si es un comando externo, llamamos a la función externalCommand, que se encargará de ejecutarlo.

Cuando salimos del bucle principal, es decir, el usuario ha hecho exit, liberamos la memoria y el programa termina con código de error 0.

Background

A la hora de manejar los comandos en segundo plano, tenemos en cuenta el atributo entero “background” de la estructura devuelta por la función Tokenize (parser.h).

Tomamos su valor y lo interpretamos como un booleano, si el proceso es requerido que se ejecute en segundo plano, crearemos los procesos de igual manera que si fuesen en primer plano. Sin embargo, en el momento de esperar por la finalización de estos procesos hijo, utilizaremos el parámetro WHNOHANG, lo que nos permite esperar por los hijos de manera no bloqueante.

Señales

Ctrl + C (SIGINT)

Definimos la variable runningJobIndex, que es un entero inicializado a -1.

Llamamos a getRunningJobIndex, una función que nos devuelve el índice del job que se está ejecutando en primer plano.

Si no hay ningún job en marcha, sale de la función.

Si recibe un índice, manda una señal (SIGINT) a su grupo de proceso y termina todos los procesos asociados al mismo.

Ctrl + Z (SIGTSTP)

Definimos la variable runningJobIndex, que es un entero inicializado a -1.

Llamamos a getRunningJobIndex, una función que nos devuelve el índice del job que se está ejecutando en primer plano.

Si no hay ningún job en marcha, sale de la función.

Si recibe un índice, manda una señal (SIGTSTP) a su grupo de proceso y termina todos los procesos asociados al mismo.

Fin del proceso hijo (SIGCHLD)

Definimos las variables:

- I, J (int): auxiliares para iterar
- Status (int): guarda el estado de terminación cuando se llama a waitpid.
- All_terminated (int): es una flag para ver si todos los procesos del job han terminado.
- Pid (pid_t)

Iteramos sobre el array de Jobs.

Inicializamos all_terminated a 1, suponiendo que todos los procesos de ese Jobs han terminado.

Para cada proceso, se llama a waitpid para verificar si el proceso ha terminad. Si devuelve 0, significa que el proceso sigue en ejecución y se actualiza el valor de all_terminated a 0.

Si all_terminated sigue siendo 1, significa que el comando / job ha terminado por completo, por lo que se decrementa el contador de jobs en segundo plano si este se trataba de uno de ellos. Además, se reinicia el id y el status a -1 para que esta posición del array pueda ser ocupada por otro comando / job.

Comandos internos

Change Directory (cd)

Variables:

- Dir (char *): Variable auxiliar para almacenar el directorio final

Si el path que recibe la función es NULL, cambia el directorio al directorio contenido en la variable de entorno HOME, de lo contrario, cambia el directorio al path que ha recibido.

La función devuelve 0 si el cambio ha sido correcto, -1 si ha habido algún problema.

Umask (umask)

Variables:

- Mode (t_mode): Variable auxiliar para almacenar la máscara que se desea actualizar o imprimir

Si el parámetro es NULL, devuelve la máscara actual, si recibe una máscara (en octal), actualiza el valor de esta.

Jobs (jobs)

Variables:

- Count(int): Variable auxiliar que cuenta los procesos reales (ID distinto de -1)
- I(int): auxiliar para iterar
- OutputFormat(char *): Variable auxiliar que contiene el string “Stopped” o “Running”

Llama a la función sortJobsById que se encarga de ordenar el array de Jobs.

Imprime con el formato correspondiente todos los Jobs, cuyo id sea distinto de -1.

Indicando si están pausados o en marcha, así como la información de estos.

Background (bg)

Variables:

- id(int): Variable auxiliar que almacena el id real del proceso que se quiere reanudar.
- I(int): auxiliar para iterar.
- Len (int): Variable auxiliar que almacena la longitud del string que contiene el comando.
- Found(int): Variable entera binaria que indica si se encontró un job con ese id.

Llama a la función sortJobsById que se encarga de ordenar el array de Jobs.

Si el recibe NULL por parámetro, es decir, no se le indica que proceso reanudar, busca el último job, si lo encuentra, marca found a 1.

Si se le pasa un id correcto, guarda en i el valor del id y marca found a 1.

Si no encuentra nada que reanudar (found = 0), sale de la función.

Actualiza el número de bgJobs y de stoppedJobs, así como los id, status y background.

Manda la señal SIGCONT para que este job continúe.

Imprime con formato el Job que se ha puesto a trabajar en segundo plano.

Exit (exit)

Si el flag AllowExit vale 1, permite salir del bucle principal del programa y así finalizar este.

Si hay trabajos pausados, avisamos al usuario de esto y cambiamos el valor de allowExit a 1, para que, volviendo a intentarlo, pueda salir.

Si no hay trabajos pausados, sale del bucle principal

Comandos externos

ExternalCommand (tline * line, char * command)

Variables:

- Current(int): Variable auxiliar que contiene el índice en el que se ha almacenado el job
- I(int): auxiliar para iterar.
- Status(int): Variable auxiliar que recoge el estado con el que terminó un hijo
- Pid(pid_t): Variable auxiliar que almacena un pid

Agregamos el Job a la lista de Jobs y comprobamos si ha habido algún error.

Si el trabajo es en segundo plano, actualizamos el valor de bgJobs e imprimimos el id del job

Inicializamos los pipes, creamos las necesarias para la comunicación entre los procesos hijos.

Para cada comando, creamos un hijo, le asignamos el process group ID a su propio pid, le asignamos los manejadores por defecto a las señales SIGTSTP y SIGINT.

También redirigimos la entrada y salida según sea necesario.

Ejecutamos el comando mediante execvp.

El padre, maneja si hubo error en el fork. Si no lo hubo, establece el id del grupo de procesos para los procesos hijos almacena el id del proceso hijo y se actualiza el estado del job.

Luego, se cierran todas las tuberías en el proceso padre y espera a que los hijos terminen.

Si el trabajo se está ejecutando en foreground, se espera a que el proceso hijo termine o sea detenido, si es en background, usamos WNOHANG para no bloquear.

+ Redirección de entrada y salida: La función redirectIO es una función auxiliar que encapsula la redirección de entradas y salidas para cada uno de los procesos hijos en un job.

Variables:

- Line(tline*): Variable auxiliar que simplifica el acceso a la variable line de un job
- J(int): auxiliar para iterar.

Entrada estándar

- Si no es el primer comando, redirige la entrada estándar desde el pipe del comando anterior.
- Si es el primer comando y además hay un fichero en “redirect_input” se redirige la entrada estándar desde ese archivo.

Salida estándar

- Si no es el último comando, redirige la salida estándar hacia el pipe del siguiente comando.
- Si es el último comando y hay un fichero al que redirigir la salida, la envía allí.

Salida error

- Si hay un error y una salida de error especificada, la envía al fichero correspondiente.

Por último, cierra todos los descriptores de fichero de los pipes.

Principales Funciones

	Main	Nombre	Tipo	Descripción
Argumentos	Argumento 1	**argc**	**int**	**Número de argumentos que recibe el programa**
	Argumento 2	**argv**	**char****	**Array que contiene los distintos argumentos**
Variables Locales	Variable 1	**tline**	**tline ***	**Estructura que contiene una línea parseada introducida por el usuario**
	Variable 2	**i**	**int**	**Variable auxiliar para iterar**
	Variable 3	**allowExit**	**int**	**Variable entera binaria que indica si un usuario puede terminar la ejecución mediante exit**
	Variable 4	**buffer**	**char ***	**Variable auxiliar para leer entrada de teclado**
Valor Devuelto			**int**	**Resultado de la ejecución**
Descripción de la Función	**Programa principal que se encarga de manejar toda la MiniShell.**			

	RedirectIO	Nombre	Tipo	Descripción
Argumentos	Argumento 1	**job**	**tjob ***	**Job generado a partir de un comando introducido por el usuario**
	Argumento 2	**i**	**int**	**Variable entera que permite identificar la posición del hijo creado y redirigir entrada / salida en consecuencia**
Variables Locales	Variable 1	**j**	**int**	**Variable auxiliar para iterar**
	Variable 2	**line**	**tline ***	**Variable auxiliar que simplifica el acceso a la variable line de un job**
Valor Devuelto			**void**	**No devuelve ningún valor**
Descripción de la Función	**Función auxiliar que encapsula la redirección de entradas y salidas para cada uno de los procesos hijos en un job.**			

	IsInputOk	**Nombre**	**Tipo**	**Descripción**
Argumentos	Argumento 1	**line**	**tline ***	**Línea introducida por el usuario**
Variables Locales	Variable 1	**i**	**int**	**Variable auxiliar para iterar**
Valor Devuelto			**int**	**Devuelve 1 si la lína es correcta.** **Devuelve 0 si la línea está vacía.** **Devuelve -1 si la línea contiene errores.**
Descripción de la Función	**Función auxiliar que comprueba si la línea introducida por el usuario es correcta (1), vacía (0) o contiene errores (-1).**			

	AddJob	**Nombre**	**Tipo**	**Descripción**
Argumentos	Argumento 1	**job**	**tjob ***	**Job que debe ser almacenado en la estructura global**
Variables Locales	Variable 1	**i**	**int**	**Variable auxiliar para iterar**
	Variable 2	**j**	**int**	**Variable auxiliar para iterar**
	Variable 3	**updated**	**int**	**Variable entera binaria que indica si se ha podido añadir el nuevo job o no**
Valor Devuelto			**int**	**Devuelve el índice en el que fue añadido el job en caso de que se pudiese.** **Devuelve -1 si hubo un error y no se pudo añadir.**
Descripción de la Función	**Función auxiliar que encapsula el proceso de añadir un nuevo job a la estructura global. En caso de no haberse podido añadir, devuelve -1.**			

	CtrlZ	**Nombre**	**Tipo**	**Descripción**
Argumentos	Argumento 1	**sig**	**int**	**Señal recibida**
Valor Devuelto			**void**	**No devuelve ningún valor**
Descripción de la Función	**Función manejadora que reprograma el funcionamiento del atajo Ctrl + Z (SIGTSTP).**			

	CtrlC	Nombre	Tipo	Descripción
Argumentos	Argumento 1	**sig**	**int**	**Señal recibida**
Valor Devuelto			**void**	**No devuelve ningún valor**
Descripción de la Función	**Función manejadora que reprograma el funcionamiento del atajo Ctrl + C (SIGINT).**			

	TerminatedChildHandler	Nombre	Tipo	Descripción
Argumentos	Argumento 1	**sig**	**int**	**Señal recibida**
Variables Locales	Variable 1	**i**	**int**	**Variable auxiliar para iterar**
	Variable 2	**j**	**int**	**Variable auxiliar para iterar**
	Variable 3	**all_terminated**	**int**	**Variable entera binaria que indica si todos los procesos hijo de un job han terminado**
	Variable 4	**pid**	**pid_t**	**Variable auxiliar que almacena un pid**
Valor Devuelto			**void**	**No devuelve ningún valor**
Descripción de la Función	**Función manejadora que es ejecutada cuando un proceso hijo ha terminado. Determina si un job ha finalizado cuando todos sus hijos han terminado.**			

	GetRunningJobIndex	Nombre	Tipo	Descripción
Variables Locales	Variable 1	**i**	**int**	**Variable auxiliar para iterar**
Valor Devuelto			**int**	**Devuelve el índice del job ejecutándose actualmente.** **Devuelve -1 si no hay ningún job ejecutándose en foreground.**
Descripción de la Función	**Función auxiliar que devuelve el índice del job que está siendo ejecutado en foreground. En caso de no haber ninguno, devuelve -1.**			

	SortJobsById	Nombre	Tipo	Descripción
Argumentos	Argumento 1	**jobs**	**tjob ****	**Listado de jobs a ordenar**
Valor Devuelto			**void**	**No devuelve ningún valor**
Descripción de la Función	**Función auxiliar que ordena la lista de Jobs mediante el algoritmo QuickSort con la ayuda de la función comparadora CompareJobs.**			

	CompareJobs	Nombre	Tipo	Descripción
Argumentos	Argumento 1	**a**	**const void ***	**Primer parámetro que comparar**
	Argumento 2	**b**	**const void ***	**Segundo parámetro que comparar**
Variables Locales	Variable 1	**jobA**	**tjob ***	**Variable a casteada a un tipo tjob ***
	Variable 2	**jobB**	**tjob ***	**Variable b casteada a un tipo tjob ***
Valor Devuelto			**int**	**Devuelve un número positivo si el id de a es mayor que b.** **Devuelve un número negativo en caso contrario.**
Descripción de la Función	**Función auxiliar que compara el argumento 1 con el argumento 2 y determina si a es mayor que b o viceversa. Es utilizada por la función SortJobsById para la ordenación.**			

	ChangeDirectory	Nombre	Tipo	Descripción
Argumentos	Argumento 1	**path**	**char ***	**Path al directorio que se desea cambiar (NULL para directorio home)**
Variables Locales	Variable 1	**dir**	**char ***	**Variable auxiliar para almacenar el directorio final**
Valor Devuelto			**int**	**Devuelve 0 si la ejecución fue exitosa** **Devuelve -1 si hubo un fallo**
Descripción de la Función	**Función que ejecuta la función interna de la Shell "cd", si el path es NULL, se cambia al directorio de la variable de entorno HOME.**			

	ExternalCommand	Nombre	Tipo	Descripción
Argumentos	Argumento 1	**line**	**tline ***	**Línea parseada que se desea ejecutar**
	Argumento 2	**command**	**char ***	**Línea sin parsear leída de stdin**
Variables Locales	Variable 1	**I**	**int**	**Variable auxiliar para iterar**
	Variable 2	**current**	**int**	**Variable auxiliar que contiene el índice en el que se ha almacenado el job**
	Variable 3	**status**	**int**	**Variable auxiliar que recoge el estado con el que terminó un hijo**
	Variable 4	**pid**	**pid_t**	**Variable auxiliar que recoge el pid a la hora de ejecutar el fork**
Valor Devuelto			**int**	**Devuelve 0 si la ejecución fue exitosa** **Devuelve -1 si hubo un fallo**
Descripción de la Función	**Función que ejecuta los comandos externos a la Shell que pueden usar pipes o ejecutarse en segundo plano.**			

	UmaskCommand	Nombre	Tipo	Descripción
Argumentos	Argumento 1	**mask**	**char ***	**Máscara a la que se desea actualizar (NULL para mostrar la máscara actual)**
Variables Locales	Variable 1	**mode**	**mode_t**	**Variable auxiliar para almacenar la máscara que se desea actualizar o imprimir**
Valor Devuelto			**void**	**No devuelve ningún valor**
Descripción de la Función	**Función que ejecuta la función interna de la Shell "umask", si mask es NULL, se imprime por pantalla el valor actual de la máscara.**			

	JobsCommand	Nombre	Tipo	Descripción
Variables Locales	Variable 1	**i**	**int**	**Variable auxiliar para iterar**
	Variable 2	**count**	**int**	**Variable auxiliar que cuenta los procesos reales (ID distinto de -1)**
	Variable 3	**outputFormat**	**char ***	**Variable auxiliar que contiene el string "Stopped" o "Running"**
Valor Devuelto			**void**	**No devuelve ningún valor**
Descripción de la Función	**Función que ejecuta la función interna de la Shell "jobs".**			

	BgCommand	Nombre	Tipo	Descripción
Argumentos	Argumento 1	**job_id**	**char ***	**Id del job que se quiere reanudar (NULL para reanudar el último proceso parado)**
Variables Locales	Variable 1	**i**	**int**	**Variable auxiliar para iterar**
	Variable 2	**id**	**int**	**Variable auxiliar que almacena el id real del proceso que se quiere reanudar**
	Variable 3	**len**	**int**	**Variable auxiliar que almacena la longitud del string que contiene el comando**
	Variable 4	**found**	**int**	**Variable entera binaria que indica si se encontró un job con ese id**
Valor Devuelto			**void**	**No devuelve ningún valor**
Descripción de la Función	**Función que ejecuta la función interna de la Shell "bg".**			

Casos de Prueba

1. Ejecutar en foreground líneas con un solo mandato y 0 o más argumentos:
   - Entrada: **echo Hello World**.
   - Salida: **Hello World**.
2. Ejecutar en foreground líneas con un solo mandato y 0 o más argumentos, redirección de entrada desde archivo y redirección de salida a archivo**:**
   - Entrada: Archivo **input.txt [grep hola < input.txt > output.txt]**.
   - Salida: Archivo **output.txt [hola]**.
   - Error: Ninguno.
3. Ejecutar en foreground líneas con dos mandatos con sus respectivos argumentos, enlazados con '|', y posible redirección de entrada desde archivo y redirección de salida a archivo:
   - Entrada: **ls -l | wc -l**.
   - Salida: **13**.
4. Ejecutar en foreground líneas con más de dos mandatos con sus respectivos argumentos, enlazados con '|', redirección de entrada desde archivo y redirección de salida a archivo:
   - Entrada: Archivo **input.txt [cat < input.txt | grep hola | wc -l > output.txt]**.
   - Salida: Archivo **output.txt [1]**.
5. Ejecutar el mandato `cd` y acceder a través de rutas absolutas y relativas, además de acceso al directorio HOME:
   - Entrada: **~/dev/MiniShell> cd .. .**
   - Salida: **~/dev>**.
   - Entrada: **~/dev/MiniShell> cd**.
   - Salida: **~/>**.
6. Ejecutar tanto en foreground como en background líneas con más de dos mandatos con sus respectivos argumentos, enlazados con '|', redirección de entrada desde archivo y redirección de salida a archivo.
   - Entrada: Archivo **input.txt [cat < input.txt | grep hola | wc -l > output.txt]** + **CtrlZ**.
   - Salida: Archivo **output.txt [1]**.

   Jobs
   - Entrada: **jobs**.
   - Salida: **[1] Stopped cat < input.txt | grep hola | wc -l > output.txt &**.

   Bg
   - Entrada: **bg**.
   - Salida: **Continua la ejecución del mandato**.
7. Reprogramar las funciones CtrlZ y CtrlC.
   - Entrada: **CtrlZ / CtrlC**.
   - Salida: **Funcionan correctamente**.

8. Ejecutar el mandato exit, termina la MiniShell de manera ordenada y muestra el prompt de la Shell desde dónde se ejecutó.

   - Entrada: **exit**.
   - Salida: **Termina la MiniShell de manera ordenada, si hay procesos parados advierte al usuario antes de terminar y le da opción a ejecutar otro mandato**.

9. **Ejecutar el mandato umask.**

   - Entrada: **umask**.
   - Salida: **0002.**

   - Entrada: **umask 0003**.
   - Salida: **Se cambia la máscara por defecto a 0003**.

Comentarios Personales

Problemas Encontrados

Reprogramación de Señales CtrlC y CtrlZ

En un principio, implementamos las señales de una manera en la que no teníamos en cuenta los procesos en segundo plano, por lo que, tras la implementación del background nos tocó reprogramarlas casi en su totalidad.

Esperar por procesos parados

Cuando conseguimos implementar finalmente la señal CtrlZ, nos preguntábamos por qué no nos devolvía el prompt a la hora de pausar el proceso activo. Tras una investigación en el manual, nos dimos cuenta de que existía un parámetro en la función waitPid que hacía justamente lo que queríamos. La opción era WUNTRACED.

Pipes Dinámicos

Inicialmente comenzamos la implementación del código suponiendo que existía una limitación en el número de pipes disponibles por job. Tras enterarnos de que esta limitación no existía, cambiamos este concepto a un array dinámico mediante la función realloc, que nos permite redistribuir la memoria dinámica reservada para los pipes.

Jobs

Para la implementación de Jobs tuvimos algunos inconvenientes debido a los problemas encontrados en las secciones anteriores con el background. Tras solventar estos problemas, la implementación de Jobs que habíamos planeado inicialmente se asemejaba bastante a la final que hemos implementado.

Críticas Constructivas

Aunque la práctica se ajusta al temario de la asignatura, es evidente que supone un aumento en la complejidad respecto a la práctica 1. Si bien nos ha gustado la propuesta, consideramos que resulta algo tediosa y repetitiva, ya que no amplía significativamente nuestro conocimiento más allá de emular el comportamiento de una terminal de Linux. Aunque esta tarea es técnicamente desafiante, también puede llegar a ser monótona.

Propuestas de Mejoras

Librería Parser.h

La librería parser.h es una herramienta muy útil para resolver la práctica, pero sería ideal que incluyera soporte para manejar las variables de entorno del sistema cuando se empleen con la sintaxis $VAR. Aunque pueda parecer un detalle menor, esta funcionalidad aportaría un nivel adicional de sofisticación a la MiniShell.

Sin embargo, decidimos no implementarla, ya que esto implicaría reprogramar gran parte de la funcionalidad ya ofrecida por la librería.

Evaluación del Tiempo Dedicado

Programación

Hemos dedicado 3 semanas al desarrollo de la MiniShell. Considerando los problemas que enfrentamos durante el proceso, la duración del proyecto se ha alargado más de lo deseado.

Memoria

Hemos dedicado tres mañanas para intentar cumplir de la mejor manera posible todos los requisitos de la memoria.