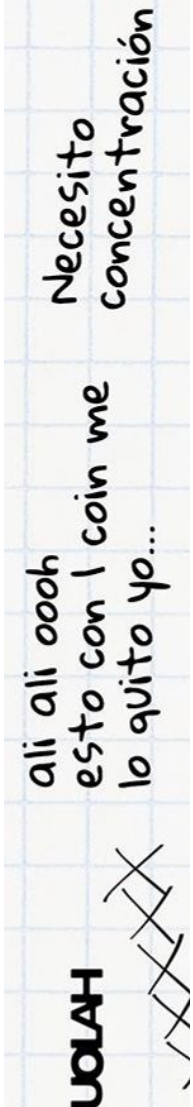
MemoriaPractica2.pdf

sirenejota

Sistemas Operativos

2º Grado en Ingeniería de Computadores

Escuela Técnica Superior de Ingeniería Informática. Campus de Móstoles
Universidad Rey Juan Carlos



Universidad Rey Juan Carlos

Sistemas Operativos

[PRÁCTICA 2 – MINISHELL]

NOAH JOSEPH MATEO ALMAGRO

TABLA DE CONTENIDO

Autores

Noah Joseph Mateo Almagro: Estudiante del doble grado en Diseño y Desarrollo de Videojuegos e Ingeniería de Computadores por la Universidad Rey Juan Carlos.

Descripción del Código

Diseño del Código

Para la implementar el programa requerido en el guión de prácticas, se ha dividido el código en los siguientes apartados:

- **Inclusión de bibliotecas y definiciones de constantes**: El código comienza incluyendo varias bibliotecas estándar de C, como <stdio.h>. Estas bibliotecas proporcionan funciones y macros necesarias para diversas operaciones del programa. También se definen también algunas constantes simbólicas, como READ_END y WRITE_END, que se utilizan para identificar los extremos de lectura y escritura de las tuberías en el código.

- **Creación de la estructura "job":** Esta estructura define un trabajo como el conjunto de un pid y un buffer.

- **Manejadores de las señales *SIGINT y SIGQUIT:*** Se han declarado ambos manejadores y se ha implementado cada uno de ellos enviando la señal correspondiente mediante el uso de un puntero al PID del proceso en primer plano, ya que este es el que se verá afectado por las correspondientes combinaciones de teclas.

- **Funciones para los mandatos especiales:**

    - **`doCd`:** Esta función implementa la funcionalidad del mandato "cd", recibiendo como argumentos el contador y el vector de argumentos de la función que la llama.

      Dentro de esta función se declara la variable "Path" para almacenar el directorio al que se redirigirá al usuario, y un buffer para mostrar el mismo.

      Mediante el uso de la función chdir(), el programa remite al usuario al directorio HOME en caso de no haber escrito ningún argumento, y a la ruta introducida en caso de haber introducido uno. En el resto de los casos, se muestra un error respecto al número de argumentos, indicando el uso correcto del mandato.

      Por último, se imprime por pantalla el directorio actual.

    - **`showJobs`:** Esta función muestra información sobre el estado de los trabajos en segundo plano (comando "Jobs"). Para ello, se recorre la lista de trabajos, almacenados en el array "Jobs", y se imprime el estado de cada uno de ellos. Todas las estructuras necesarias se pasan como argumentos a esta función.

      La comprobación del estado de cada trabajo se ha implementado mediante un bucle "for" que recorre todos los elementos del array "Jobs", y tres bucles "if" que, mediante la función "waitpid()", implementan las tres salidas posibles: Terminado, Ejecutándose o Error.

Para especificar si se trata del último (+), penúltimo (-) u otro trabajo, se ha añadido otro bucle "if" en cada uno de los casos, el cual comprueba el lugar de la lista en la que se encuentra el trabajo, especificando esta información en la salida por pantalla, junto al estado de dicho trabajo.

Tal y como pasa en el comando en el que se basa esta función, tras mostrar el estado de los trabajos, se eliminan aquellos terminados y erróneos, de manera que no volverán a mostrarse.

- **`executeForeground`:** Esta función permite cambiar un trabajo en segundo plano a primer plano, y esperar a que el trabajo termine.

  Para ello, la función debe recibir como argumentos el array de trabajos y un puntero al número de trabajos restante, así como al array de argumentos de la función llamadora.

  Primero, se elige el trabajo que pasará a primer plano. Con ayuda de un bucle "if", se escoge el último trabajo en caso de no haber introducido ningún argumento, y el trabajo correspondiente al número introducido como argumento en caso contrario.

  Después, se utiliza la función "waitpid()" para esperar que termine el trabajo especificado. Y, por último, se mueven los trabajos restantes en el array "Jobs" para llenar el espacio vacío.

- **Función principal (`main`):** Comienza configurando los manejadores de señales para SIGINT y SIGQUIT mediante llamadas a "signal()", y definiendo variables locales para almacenar la entrada del usuario y otros datos relacionados. Además, se duplican los descriptores de archivo estándar (stdout, stdin, stderr) para que puedan restaurarse después de las redirecciones.

  Tras esto, se declara un bucle "while" infinito que se ejecuta hasta que se ingrese el comando de salida.

  En cada iteración del bucle infinito:

  - Se imprime el prompt para el usuario (>>), se lee la entrada mediante la función "fgets()", se realiza el análisis de la información introducida convirtiendo la línea en tokens mediante la función "tokenize()", y se almacena en un objeto de la clase "tline". Además, se reserva memoria dinámica para los pipes mediante "malloc()" y se muestran por separado los comandos y sus argumentos.

  - Se comprueba si es necesario realizar alguna redirección (usando las opciones "redirect_input", "redirect_ output" y "redirect_error" de la clase "tline"). En caso afirmativo, se utiliza la función "dup2()" para cambiar la entrada, salida o salida de errores, respectivamente. Si falla alguna de estas redirecciones, el descriptor de archivo correspondiente contendrá un número negativo y se mostrará un error por pantalla.

  - Se comprueba si el mandato es "cd", "jobs" o "fg", en cuyo caso se llama a la función correspondiente, "doCd()", "showJobs()" y "executeForeground()", respectivamente.

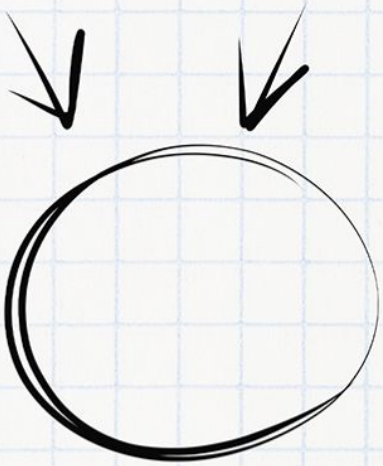
  - Si el comando no es ninguno de los anteriores, se crea un nuevo proceso hijo utilizando "fork()".

Imagínate aprobando el examen

Necesitas tiempo y concentración

Planes	PLAN TURBO	PLAN PRO	PLAN PRO+
Descargas sin publi al mes	10	40	80
Elimina el video entre descargas	✓	✓	✓
Descarga carpetas	✗	✓	✓
Descarga archivos grandes	✗	✓	✓
Visualiza apuntes online sin publi	✗	✓	✓
Elimina toda la publi web	✗	✗	✓
Precios Anual	0,99 € / mes	3,99 € / mes	7,99 € / mes

Ahora que puedes conseguirlo, ¿Qué nota vas a sacar?



WUOLAH

Nótese que, en el código proporcionado, se utiliza un array de pipes para establecer la comunicación entre los diferentes procesos hijos. El tamaño del array de pipes es una unidad más pequeño que el número de comandos en la lista de comandos.

Después de cerrar los extremos de las pipes que no se utilizarán en cada caso, el proceso hijo ejecuta el comando utilizando la función execvp(). Esta función reemplaza la imagen del proceso hijo con la del nuevo programa especificado por el comando. Si el comando no existe, se muestra un error.

El proceso padre, por otro lado, es responsable de gestionar los extremos de los pipes antes de que el proceso hijo ejecute el comando. Al cerrar los extremos de las pipes que no se utilizarán, el padre asegura que la comunicación se realice correctamente entre los procesos.

Es importante destacar que el proceso padre no ejecuta el comando directamente, sino que lo delega al proceso hijo utilizando execvp(). Esto permite que cada comando en la lista de comandos sea ejecutado por un proceso hijo independiente.

Además, el proceso padre se gestiona mediante un condicional basado en la variable background, que solo puede tomar los valores 1 o 0. Esta variable indica si el proceso hijo se ejecuta en segundo plano (background) o en primer plano (foreground).

Si la variable background es verdadera y el proceso hijo se está ejecutando en segundo plano, el padre muestra por pantalla el PID del hijo y el número de trabajo que le corresponde. Por otro lado, si la variable background es falsa, el padre espera a que el proceso hijo, que se ejecuta en primer plano, termine su ejecución antes de continuar.

- Después de manejar el comando, se liberan los recursos y se restablecen los descriptores de archivo estándar.

Casos de Prueba

Para garantizar la corrección y el rendimiento del programa, se realizaron diversas pruebas, detalladas a continuación.

Estas pruebas verifican la funcionalidad de cada función individual y aseguran que todas las partes del programa funcionan correctamente juntas.

- **Ejecución de un mandato:** Se ha evaluado mediante la ejecución del comando “man man”, el cual accede correctamente a la página correspondiente de manual. Al cerrarlo con la señal quit. La Shell continúa activa, a la espera de la siguiente orden.

- **Ejecución del comando cd exitosa:** Para evaluar esta opción se ha creado la siguiente estructura de directorios y archivos:

  Carpeta fuente
  Pruebas
  Archivo 1.txt
  Archivo2.txt
  Archivo3.txt
  Archivo4
  Archivo5
  Archivo6

  Partiendo de la carpeta padre, se ha ejecutado el comando “cd” correspondiente para acceder a la carpeta “Pruebas”. Como era de esperar, el programa ha respondido correctamente. A continuación, para comprobar que se ha cambiado de directorio, se ha ejecutado el comando “ls”, el cual ha mostrado correctamente los archivos que se encuentran dentro de la carpeta accedida.

- **Ejecución de dos comandos unidos mediante una tubería:** Para esto se ha partido del apartado anterior, manteniendo la estructura de archivos y situando el programa en la carpeta “Pruebas”. Después, se ha evaluado la funcionalidad mencionada uniendo mediante una tubería los comandos “ls -l” y “grep .txt” El programa responde correctamente, mostrando por pantalla únicamente los archivos con formato “.txt”

- **Ejecución de más de dos comandos unidos mediante una tubería:** De nuevo, se ha partido de la estructura de archivos anterior. Esta vez, se han utilizado los comandos “ls -l”, “grep .txt” y “wc -l”, unidos en este orden, para contar los archivos .txt que se encuentran en este directorio.
  Al ejecutar la orden, el programa muestra correctamente el número 3 que, como se ha demostrado en el apartado anterior, es la cantidad correcta.

- **Redirección de la salida a un archivo de texto (en segundo plano):** Para este apartado se ha repetido la operación de la ejecución con dos comandos, pero esta vez añadiendo “>salidaDosMandatos.txt &” al final de la orden. Esto provoca que la orden se ejecute en segundo plano, por lo que, inmediatamente, la Shell nos da la opción de seguir trabajando con ella.

  Tras esto, el archivo ha sido creado de manera correcta, y contiene exactamente lo que se obtendría como salida de la orden original.

- **Redirección de la entrada desde un archivo de texto (en segundo plano):** Esta vez, se han escrito los siguientes nombres en el nuevo archivo "nombres.txt": Pepe, María, Juan y Adolfo.

  Después se ha realizado el comando "cat < nombres.txt | grep a" redirigiendo la entrada desde este archivo. Como podía esperarse, programa responde correctamente, mostrando únicamente las palabras María y Juan.

- **Comando "jobs":** Para evaluar esta funcionalidad se han creado tres procesos en segundo plano: "sleep 1", "sleep 30" y "sleep 300".

  Al ejecutar Jobs por primera vez se muestra correctamente que el primer proceso que fue creado ha terminado, y los otros dos se encuentran en ejecución.

  Tras esperar 30 segundos, se vuelve a ejecutar el comando para comprobar si el proceso "sleep 1" ha sido eliminado, recibiendo así, exitosamente, información únicamente de los procesos restantes.

- **Comando "fg":** Para realizar esta prueba se han vuelto a utilizar los comandos "sleep 30" y "sleep 300". Tras haberlos creado, se ha introducido la instrucción "fg" seguida del identificador de trabajo devuelto por la Shell tras ejecutar el primer mandato.

  Tal y como se esperaba, la Shell retoma la espera de 30 segundos y, tras pasar estos (o introducir el comando CTRL + C) se muestra disponible para recibir nuevas instrucciones.

- **Redirección de errores a un archivo de texto y error en "cd":** Para esto se ha ejecutado el comando "cd" de la siguiente forma: cd directorioErroneo >& errorCd.txt.

  Como resultado, se ha creado un archivo llamado errorCd.txt en la carpeta fuente, el cual es contenedor de una línea de texto que especifica el error al cambiar de directorio.

  Tras esto la Shell continúa activa y a la espera de órdenes, por lo que funciona correctamente.

NOTA: La creación y edición de todos los archivos utilizados en estas pruebas se ha realizado mediante los comandos pertinentes en la propia miniShell.

Comentarios Personales

Problemas Encontrados

Durante el desarrollo de la práctica, me he encontrado con un único contratiempo, el tiempo. La práctica se encuentra en un punto muy comprometido del cuatrimestre y, en el doble grado con videojuegos, de la carrera en general. Esto ha ocasionado que, inevitablemente, me viese obligado a retrasar la realización de esta a la convocatoria extraordinaria.

Críticas Constructivas

Me encuentro bastante satisfecho en lo relativo a la duración de la práctica y su dificultad, no tengo ninguna crítica.

Propuesta De Mejoras

No es realmente culpa del profesorado que la práctica ponga a los alumnos a contra reloj, sin embargo, creo que adelantar un poco su realización sería beneficioso para todos.

Evaluación Del Tiempo Dedicado

El tiempo empleado en la realización de esta práctica ha sido, en total, de veinte horas. Diecisiete de ellas dedicadas a la realización de la práctica en sí y, las otras tres, dedicadas a la realización de pruebas y el desarrollo de la memoria.