

Sistemas Operativos

[Práctica 2 - MINISHELL]

Amanda Castro López y Carlos Rodríguez Gómez

Tabla de contenido

[Autores 2](#_Toc115812632)

[Descripción del Código 3](#_Toc115812633)

[Diseño del Código 3](#_Toc115812634)

[Principales Funciones 3](#_Toc115812635)

[Casos de Prueba 3](#_Toc115812636)

[Comentarios Personales 4](#_Toc115812637)

# Autores

Amanda Castro López

DNI: 21155084Y

Correo de la administración: c.rodriguezgo.2018@alumnos.urjc.es

Carlos Rodríguez Gómez

DNI: 21155084Y

Correo de la administración: a.castrol.2020@alumnos.urjc.es

# Descripción del Código

**<< MÁXIMO 6 PÁGINAS >>**

## Diseño del Código

**<< EXPLICACIÓN DEL DISEÑO DEL CÓDIGO:**

**Para comenzar a podernos plantear los requisitos y objetivos de esta práctica, decidimos realizar un desglosamiento de cada requisito y de que significaba a nivel de código cada uno de ellos para de esta forma ir dando, al menos en nuestra cabeza forma a la estructura de la práctica. Como esta vez era una practica mas densa que la anterior, aunque pensamos en hacer pseudocódigo tuvimos que descartarlo porque necesitábamos optimizar todo el tiempo posible, ya que no íbamos con demasiado tiempo debido a la gran carga de trabajo que hemos tenido.**

* **ALGORITMOS UTILIZADOS**

**Como algoritmos mas significativos que hemos utilizado. Podemos encontrar estructuras FOR para recorrer arrays como el de Jobs, donde tenemos almacenados los diferentes procesos con sus pids o para recorrer los diferentes valores de line.**

* **ESTRATÉGIA DE EJECUCIÓN DE MANDATOS Y GESTIÓN DE TUBERRÍAS**

**A la hora de ejecutar mandatos y gestionar las tuberías, lo hicimos centrándonos en conseguir primero tener una versión en primer plano funcional y ejecutable, y una vez viésemos y comprendiésemos el segundo plano, trataríamos de implementarlo para el segundo plano.**

**De esta manera comenzamos, para cumplir con la ejecución en primer plano tenemos un código capaz de ejecutar uno o más mandatos. Cada mandato será un nuevo proceso y a su vez todos estos procesos serán hijos del mismo padre, que será la propia MiniShell. El código se ha separado en dos partes principales, si hay un único mandato o si hay más de un mandato.**

**En caso de que haya un solo mandato, el proceso hijo simplemente contralara cualquier posible redirección y sencillamente ejecutará el mandato en cuestión. Su padre, por otro lado, tendrá que esperarlo mediante un waitpid antes de poder volver a mostrar el promt.**

**Si hay más de un mandato entonces, será necesario la utilización de tuberías que conecten los distintos hijos para que de esta forma puedan pasarse la información. Para cumplir con esto, primero creamos una tubería para el número total de comandos menos uno, es decir si hay tres comandos, habrá dos tuberías. En función de si estamos en el padre o el hijo, y dependiendo de si el proceso hijo es el primero, intermedio o último se irán abriendo o cerrando las tuberías de diferentes formas**

**En caso de que nos encontremos en el primer mandato conectaremos mediante el dup2 la salida del primer proceso hijo y la entrada del segundo proceso padre, y después se cerraran todas las tuberías, además podremos controlar si hay redirección de salida.**

**Si el proceso hijo es el intermedio tendremos como entrada del hijo la salida de la tubería del anterior proceso hijo, y conectaremos nuestra salida, a la entrada de la siguiente tubería, para que de esta forma el siguiente proceso hijo reciba por entrada la salida de esta tubería, de nuevo cerraremos todas las tuberías.**

**Por último, si nos encontramos en el proceso hijo del último mandato, recibiremos por entrada la salida de la última tubería y cerraremos el resto de las tuberías, además tendremos en cuenta las redirecciones de salida y de salida de error.**

**Ahora tendremos que controlar las tuberías del padre, debido a que cada vez que un hijo nuevo es creado, heredará las mismas tuberías que el padre tenga abiertas. Y por eso el proceso padre según vaya avanzando tendrá que ir cerrando las tuberías ya usadas, para que de esta forma los procesos hijos no hereden tuberías inútiles y se puedan producir fallos a la hora te transmitir los datos, de esta forma si el padre ahora lo siguiente dependiendo del mandato:**

**Cuando se encuentre en el primer mandato cerrara la escritura de la primera tubería.**

**En caso de que el padre se encuentre en un mandato intermedio, se cerrara la lectura de la tubería anterior y la de escritura de la actual.**

**Por último, cuando se encuentre valga la redundancia en el último mandato se cerrará la lectura de la tubería actual.**

**De esta manera conseguimos gestionar las tuberías, pudiendo ejecutar las instrucciones con múltiples mandatos y haciendo que de la misma manera que en caso de un solo comando el padre espere.**

* **IMPLEMENTACIÓN DE BACKGROUND**
* **IMPLEMENTACIÓN DE SEÑALES**
* **IMPLEMENTACIÓN DE JOBS+FG**
* **ESTRUCTURAS DE DATOS ESPECÍFICAS**
* **…**

**>>**

**<< NO INCLUIR CÓDIGO FUENTE >>**

**<< SE PUEDE UTILIZAR PSEUDOCÓDIGO O DIAGRAMAS DE APOYO A LA EXPLICACIÓN >>**

## Principales Funciones

**<< DESCRIPCIÓN DE LAS PRINCIPALES FUNCIONES IMPLEMENTADAS INCLUYENDO LA FUNCIÓN PRINCIPAL, MAIN>>**

**<< UTILIZAR EL FORMATO DE LA SIGUIENTE TABLA PARA CADA FUNCIÓN >>**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Nombre Función** | **Nombre** | **Tipo** | **Descripción** |
| **Argumentos** | Argumento 1 |  |  |  |
|  | Argumento 2 |  |  |  |
|  | … |  |  |  |
| **Variables Locales** | Variable 1 |  |  |  |
|  | Variable 2 |  |  |  |
|  | Variable 3 |  |  |  |
|  | … |  |  |  |
| **Valor Devuelto** |  |  |  |  |
| **Descripción de la Función** |  |  |  |  |

## Casos de Prueba

**<< MÁXIMO 2 PÁGINAS >>**

**<< LISTA Y BREVE EXPLICACIÓN DE CASOS DE PRUEBA UTILIZADOS PARA VALIDAR LA PRÁCTICA >>**

# Comentarios Personales

**<< MÁXIMO 2 PÁGINAS >>**

**<< PROBLEMAS ENCONTRADOS>>**

**<< CRÍTICAS CONSTRUCTIVAS >>**

**<< PROPUESTA DE MEJORAS >>**

**<< EVALUACIÓN DEL TIEMPO DEDICADO >>**