Integrantes del grupo:

- Natalia Montoya Gómez
- Rebeca Tamara González Sánchez
- Marcel Andrei Voicu
- Carlos Javier Hellín Asensio

Profesora: Julia Clemente Párraga
Grupo de laboratorio: jueves de 15:00 a 17:00

Fases de la práctica realizadas:

1. Ciclo de ejecución de órdenes
2. Ejecución de órdenes externas simples en primer plano
3. Ejecución de órdenes externas simples en segundo plano
4. Realización de Makefile
5. Ejecución de secuencia de órdenes
6. Tratamiento de redirecciones
7. Implementación de tuberías o pipes

Mejoras adicionales:
Se ha solucionado el problema de que cuando se pulsa CTRL+D se entra en un bucle infinito en el que muestra el prompt continuamente y ahora sale de la minishell como ocurre en bash. Para ello se ha modificado la función leer_lineas_ordenes en entrada_minishell.c para que devuelva un 0 o 1 dependiendo de si fgets da NULL o no.

Se ha movido todo lo relacionado con señales a los ficheros sig.c y sig.h para una mejora en la estructura del código. Antes de entrar al bucle en minishell.c se llama a **iniciar_signals()** que se encarga de preparar todos los manejadores de señales.

- Control de jobs (órdenes simples)
En minishell.c se utiliza **es_ord_job(orden)** y **ejecutar_ord_job(orden)** que se encuentran en job.c para comprobar y ejecutar respectivamente si la orden se trata de las órdenes internas jobs, fg o bg. (En el caso de fg y bg también se le puede pasar como argumento un número de la lista mostrada en jobs)
Cuando se introduce la orden jobs se ejecuta la función **mostrar_jobs()** en job.c encargada de recorrer y mostrar la lista enlazada, que se trata de una estructura de datos llamada job en job.h y uno de sus miembros contiene un puntero al siguiente job.
En el caso de fg o fg id siendo id el número del job, se ejecuta **foreground(id)** que pone el job en estado foreground, envía la señal SIGCONT al pid del proceso hija con kill() y espera a una señal con pause() como SIGINT y SIGTSP o espera a que el proceso hija termine y envía la señal SIGCHLD al padre.
Lo mismo para bg o bg id que ejecuta **background(id)**, comprueba que el job este en estado stopped para ponerlo en running mandando la señal SIGCONT al pid de la hija con kill() pero esta vez no espera ya que se ejecuta en segundo plano.

Después de ejecutar_orden en ejecutar.c y si el pid que devuelve es > 0 y nordenes es == 1 se llama a **agregar_job(pid, orden, backgr)** encontrado en job.c que añade el job a la lista enlazada solicitando memoria dinámica con malloc y dependiendo del backgr lo pone en estado running o foreground.

Antes de terminar la minishell con return 0 en minishell.c se llama a **terminar_jobs()** que se encuentra en job.c para que el proceso padre envíe la señal SIGHUP con kill() a todos los jobs.

Se ha modificado manejar_sigchild() para cuando el padre reciba la señal SIGCHILD de la hija también elimine el job y libere la memoria dinámica con **eliminar_job()** en job.c después de eliminar su task_struct en waitpid.

Se han agregado a sig.c los manejadores de señales de CTRL+C en **manejar_sigint()** que envía la señal SIGINT al job que este en ejecución en primer plano y CTRL+Z en **manejar_sigtstp()** que pone el estado del job en stopped y envía la señal SIGTSP al job que también este en ejecución en primer plano usando en ambos casos **foreground_job()** encontrado en job.c que obtiene el job de la lista enlazada que se encuentre en estado foreground.

Aprovechando la función **ignorar_espacios(buf)** en job.c que se usa en ese mismo fichero, también se usa en minishell.c para eliminar espacios a la izquierda y que funcionen órdenes como ls ; exit