

Informe de la Tarea 1



Sistemas Operativos

Autores:

José González Aguayo

Gonzalo Quilodrán Neira

Joaquín Sandoval Reyes

Universidad de Concepción

Fecha: 28 de septiembre de 2025

Resumen

Dentro de la informática, el conocimiento relacionado a los sistemas operativos es indispensable, ya que estos están encargados de facilitar la comunicación entre el usuario, el software y el hardware, además de gestionar la administración de recursos del computador. Los intérpretes de comandos o Shell son indispensables para la comunicación del usuario con el sistema operativo, permitiendo la ejecución de comandos o programas, la gestión de entradas y salidas, y el control de procesos del sistema. Ese fue el fin de este informe: poder entender de mejor forma qué es una shell y cómo funciona internamente, además de comprender todo lo relacionado con la creación de procesos. A continuación, veremos los datos más importantes respecto al código realizado para crear nuestra propia shell.

1. Introducción

La tarea 1 de la asignatura 501251-1 Sistemas Operativos, impartida por la Universidad de Concepción, consistió en la construcción de un intérprete de comandos (Shell) de Linux, esto mediante la escritura de un código en lenguaje C y utilizando llamadas al sistema, principalmente `fork` para crear procesos hijos, variantes de `exec` para cambiar el código de los procesos hijos creados, `wait`, `pipe`, entre otras llamadas utilizadas para emular completamente el funcionamiento de una shell. El código de la tarea se encuentra en el siguiente repositorio de GitHub:

<https://github.com/Gquilodran/tarea-1-Sistemas-Operativos-UdeC>

2. Objetivos

Para el desarrollo de la presente tarea, se tenían dos grandes objetivos: la construcción de un código que permitiera emular una Shell de Linux, y el poder crear una función propia "miprof" que permitiera visualizar el desempeño del sistema al ejecutar ciertos comandos. Además, se tenían los siguientes objetivos:

- Que nuestra shell reconozca y maneje Pipes.
- Reconocer comandos internos como `exit` o el mencionado `miprof`.
- Reconocimiento de comandos típicos utilizados en Linux, y en caso de no reconocer el comando deseado, manejar el error.

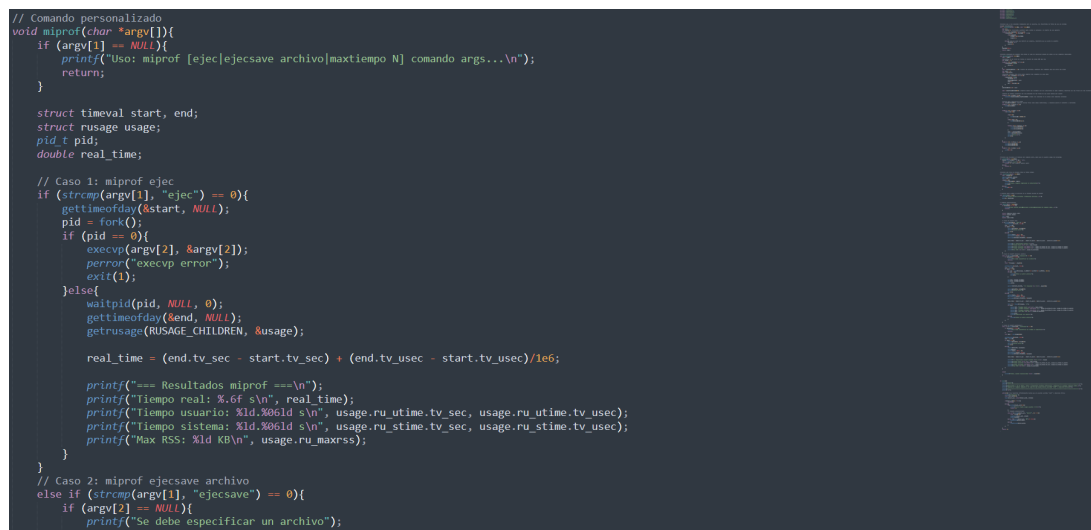
3. Diseño de la solución

Para poder construir la shell, se fueron creando funciones gradualmente para obtener el resultado deseado:

1. `main` → inicia la shell, recibe comandos, originalmente solo detectaba si se ingresaba un texto o no, luego se modificó para detectar si son `exit`, `miprof`, pipes o comandos comunes, y los ejecuta llamando a otra función.
2. `creaProceso` → crea un proceso hijo (por medio de `fork`) que ejecuta un comando normal, a través de `execvp()`. Para lograr esto último, el string ingresado debe tener un formato exacto.

3. `separaCmd` → divide el texto ingresado en palabras para que `execvp()` las entienda como comando y argumentos.
4. `exitCmd` → detecta si el usuario escribió `exit` para cerrar la shell.
5. `manejoPipe` → reconoce si el usuario ingresó alguna pipe dentro del comando, luego se cambian todas las "|" detectadas por "\"0" para separar el comando original en subcomandos, los cuales son entregados a `separaCmd` para dejarlos en el formato deseado. Después de eso, se ejecutan los comandos de forma ordenada por medio de las pipes.
6. `miprof` → comando personalizado que mide tiempos, memoria y permite guardar resultados o limitar la ejecución de procesos.
7. `handler` → función encargada de finalizar un proceso cuando se supera un tiempo máximo, utilizado en uno de los casos de `miprof`.

Dentro del código se realizaron anotaciones escritas al momento del desarrollo, en donde se explica qué hacen ciertas líneas.



```
// Comando personalizado
void miprof(char *argv[]){
    if (argv[1] == NULL){
        printf("Uso: miprof [ejec|ejecsava archivo|maxtiempo N] comando args...\n");
        return;
    }

    struct timeval start, end;
    struct rusage usage;
    pid_t pid;
    double real_time;

    // Caso 1: miprof ejec
    if (strcmp(argv[1], "ejec") == 0){
        gettimeofday(&start, NULL);
        pid = fork();
        if (pid == 0){
            execvp(argv[2], &argv[2]);
            perror("execvp error");
            exit(1);
        }else{
            waitpid(pid, NULL, 0);
            gettimeofday(&end, NULL);
            getrusage(RUSAGE_CHILDREN, &usage);

            real_time = (end.tv_sec - start.tv_sec) + (end.tv_usec - start.tv_usec)/1e6;

            printf("=== Resultados miprof ===\n");
            printf("Tiempo real: %.6f s\n", real_time);
            printf("Tiempo usuario: %ld.%06ld s\n", usage.ru_utime.tv_sec, usage.ru_utime.tv_usec);
            printf("Tiempo sistema: %ld.%06ld s\n", usage.ru_stime.tv_sec, usage.ru_stime.tv_usec);
            printf("Max RSS: %ld KB\n", usage.ru_maxrss);
        }
    }

    // Caso 2: miprof ejecsava archivo
    else if (strcmp(argv[1], "ejecsava") == 0){
        if (argv[2] == NULL){
            printf("Se debe especificar un archivo");
            return;
        }
    }
}
```

Figura 1: Pantallazo de una parte del código de la shell.

4. Pruebas y resultados

Tras la finalización de la escritura de cada función, se comprobaba si funcionaba correctamente. Normalmente, al compilar el archivo, saltaba algún error por un mal manejo de punteros o un error en los condicionales de `if`, los cuales se solucionaban con facilidad. Otros errores destacables al mencionar serían los relacionados con el manejo de pipes, que no eran detectadas por el sistema o en uno de los ciclos `for` para crear las pipes, se creaba un número menor o mayor al necesario para el comando deseado. Tras solucionar eso, el código no mostró ningún otro error que impidiera su correcto funcionamiento, compilándose y ejecutándose con normalidad. Algo a destacar es que la shell solo soportaría un número limitado de caracteres en los comandos; si se sobrepasa este número podría descartar ciertas partes del comando ingresado por el usuario.

```

=====
Bienvenido a la Mi Shell, tarea 1 asignatura Sistemas Operativos, impartida el segundo semestre 2025.
Ingrese el comando deseado o en caso de que desee salir, escriba 'exit' o precione Ctrl+C
=====

mishell:$ ls -l
total 36
-rwxrwxrwx 1 gonzalo gonzalo 2659 Sep 27 20:49 README.md
drwxrwxrwx 1 gonzalo gonzalo 512 Sep 24 18:35 f
-rwxrwxrwx 1 gonzalo gonzalo 21496 Sep 27 20:25 mishell
-rwxrwxrwx 1 gonzalo gonzalo 7819 Sep 27 20:25 mishell.c
-rwxrwxrwx 1 gonzalo gonzalo 462 Sep 27 20:46 salida.txt
-rwxrwxrwx 1 gonzalo gonzalo 4 Sep 24 19:22 text.txt
mishell:$ cat text.txt hola
holacat: hola: No such file or directory
mishell:$ ls -l | sort -m -w 5
sort: invalid option -- 'w'
Try 'sort --help' for more information.
mishell:$ ls -l | sort
-rwxrwxrwx 1 gonzalo gonzalo 4 Sep 24 19:22 text.txt
-rwxrwxrwx 1 gonzalo gonzalo 462 Sep 27 20:46 salida.txt
-rwxrwxrwx 1 gonzalo gonzalo 2659 Sep 27 20:49 README.md
-rwxrwxrwx 1 gonzalo gonzalo 7819 Sep 27 20:25 mishell.c
-rwxrwxrwx 1 gonzalo gonzalo 21496 Sep 27 20:25 mishell
drwxrwxrwx 1 gonzalo gonzalo 512 Sep 24 18:35 f
total 36
mishell:$ |

```

Figura 2: Ejecución final de la shell, probando algunos comandos simples

```

mishell:$ miprof ejec ls -l
total 36
-rwxrwxrwx 1 gonzalo gonzalo 2659 Sep 27 20:49 README.md
drwxrwxrwx 1 gonzalo gonzalo 512 Sep 24 18:35 f
-rwxrwxrwx 1 gonzalo gonzalo 21496 Sep 27 20:25 mishell
-rwxrwxrwx 1 gonzalo gonzalo 7734 Sep 28 00:57 mishell.c
-rwxrwxrwx 1 gonzalo gonzalo 462 Sep 27 20:46 salida.txt
-rwxrwxrwx 1 gonzalo gonzalo 4 Sep 24 19:22 text.txt
=== Resultados miprof ===
Tiempo real: 0.067103 s
Tiempo usuario: 0.013031 s
Tiempo sistema: 0.037138 s
Max RSS: 3200 KB
mishell:$ |

```

Figura 3: Ejecución de la shell, probando el comando personalizado

5. Conclusiones

El objetivo fundamental de la tarea fue completado, logrando desarrollar una shell funcional, implementando las características principales de un intérprete de comandos en Linux, incluyendo manejo de procesos, pipes y un comando personalizado de medición de desempeño. En el transcurso, el equipo fortaleció su comprensión de las llamadas al sistema y la gestión de procesos en C. Futuras mejoras podrían incluir el manejo de un número ilimitado de caracteres y la optimización de la ejecución de comandos con múltiples pipes, consolidando así la eficiencia y robustez de nuestra shell.