Informe de Laboratorio: Programa psinfo

Curso: Sistemas Operativos

Nombre: Juan Sebastian Loaiza Mazo, Sulay Gisela Martínez Barreto, Mateo Herrera Londoño

1. Estructura general del programa

El programa `psinfo` fue desarrollado en lenguaje C y está dividido en múltiples archivos para mantener una estructura modular y ordenada. A continuación se describe la funcionalidad principal de cada archivo:

- **psinfo.c**: Archivo principal que contiene la función `main`, gestiona la lectura de argumentos desde la línea de comandos y la lógica general del programa (manejo de banderas `-l`, `-r`, y lectura de PID).
- **infoPid.c**: Contiene la función `leerInformacionProceso`, que es responsable de leer y parsear los datos relevantes del archivo `/proc/[pid]/status` y almacenarlos en una estructura definida para cada proceso.
- **infoFile.c**: Implementa la funcionalidad para generar archivos de reporte, según la opción `-r`. Construye el nombre del archivo y escribe en él los datos previamente recolectados de cada proceso.
- **funciones.h**: Archivo de cabecera que contiene las declaraciones de funciones y estructuras utilizadas por los otros archivos.

2. Reporte de uso de IA generativa

Para el desarrollo de este laboratorio, se utilizó ChatGPT como herramienta generativa de código.

Partes generadas por la IA:

- Estructura base de la función `main`.
- Parte de la lógica para abrir y leer archivos dentro de `/proc/[pid]/status` en el `InfoFile.c`.
- Estructura para almacenar los datos del proceso en el `InfoPid`.

En sí, la base del código fue generada por IA, pero tuvo modificaciones.

Main:

```c
// Generated by: chatgpt
// Date: 2023-10-04
// Description: This program retrieves and displays information about a process in Linux using the /proc filesystem.
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "funciones.h" // Incluye las definiciones de las funciones de infoPid.c

int main(int argc, char *argv[]) {
    if (argc < 2) {
        fprintf(stderr, "Usos:\n %s [-l] <PID> <PID> ...\n %s [-r] <PID> <PID> ...\n %s <PID> <PID> ...\n", argv[0], argv[0], argv[0]);
        return 1;
    }else if (strcmp(argv[1], "-l") == 0) {
        if (argc < 3) {
            fprintf(stderr, "Error: Debe ingresar al menos un PID después de -l. \nEjemplo: ./psinfo -l 2093\n");
            return 1;
        }
        print_multiple_pids(argc - 2, &argv[2]); // Procesar múltiples PIDs
        return 0;
    }else if (strcmp(argv[1], "-r") == 0) {
        if (argc < 3) {
            fprintf(stderr, "Uso: %s -r <PID1> <PID2> ...\n", argv[0]);
            return 1;
        }
        generar_reporte_archivo(argc - 2, &argv[2]); // Genera el archivo del reporte
        return 0;
    } else if (argc == 2) {
        // Manejar un solo PID
        ProcInfo info;
        if (get_proc_info(argv[1], &info) == 0) {
            print_proc_info(&info); // Imprimir información del proceso
        } else {
            fprintf(stderr, "Error: No se pudo obtener información para el PID %s.\n", argv[1]);
        }
        return 0;
    } else {
        // Mostrar error si hay más de un PID sin usar -l o -r
```

InfoFile.c

```
void generar_reporte_archivo(int count, char *pids[]) {
    for (int i = 0; i < count; ++i) {
        if (!es_pid_valido(pids[i])) {
            printf("Error: El PID %s es inválido o no existe.\n", pids[i]);
            return; // Salir si algún PID es inválido
        }
    }

    // Calcular el tamaño necesario para el nombre del archivo
    size_t len_total = strlen("psinfo-report") + 1;
    for (int i = 0; i < count; ++i) {
        len_total += strlen(pids[i]) + 1; // para '-' y el PID
    }
    len_total += strlen(".info") + 1;

    char *filename = malloc(len_total);
    if (!filename) {
        perror("Error al asignar memoria para el nombre del archivo");
        return;
    }

    strcpy(filename, "psinfo-report");
    for (int i = 0; i < count; ++i) {
        strcat(filename, "-");
        strcat(filename, pids[i]);
    }
    strcat(filename, ".info");

    FILE *file = fopen(filename, "w");
    if (!file) {
        perror("No se pudo crear el archivo de reporte");
        free(filename);
        return;
    }

    for (int i = 0; i < count; ++i) {
        fprintf(file, "Pid: %s\n", pids[i]);
        FILE *f;
        char path[64], line[256];
        snprintf(path, sizeof(path), "/proc/%s/status", pids[i]);
        f = fopen(path, "r");
        if (!f) {
            fprintf(file, "Error: No se pudo abrir /proc/%s/status\n\n", pids[i]);
            continue;
        }
```

InfoPid.c

```c
/* Función genérica para extraer valores de líneas específicas */
void extraer_valor(const char* linea, const char* prefijo, const char* formato, void* destino) {
    if (strncmp(linea, prefijo, strlen(prefijo)) == 0) {
        sscanf(linea + strlen(prefijo), formato, destino);
    }
}

/* Función para obtener información del proceso */
int get_proc_info(const char* pid, ProcInfo* info) {
    char path[64];
    snprintf(path, sizeof(path), "/proc/%s/status", pid);

    FILE *f = fopen(path, "r");
    if (!f) {
        fprintf(stderr, "Error: No se pudo abrir el archivo para el PID %s. Razón: %s\n", pid, strerror(errno));
        return -1;
    }
```

Modificaciones realizadas:

- Se realizaron ajustes en los nombres de campos y variables para mejorar la legibilidad y coherencia del código.
- Se incluyó lógica adicional que permite presentar la información del proceso de manera más clara y alineada con los requerimientos del programa.
- Se implementaron validaciones adicionales para evitar errores en caso de que un proceso finalice antes de que se complete la lectura de sus datos.
- Se mejoró el control de errores mediante mensajes personalizados y la gestión adecuada de códigos de salida.
- El código fue adaptado a una estructura modular, distribuyendo sus funcionalidades en archivos separados (`infoPid.c`, `infoFile.c`, y `funciones.h`), facilitando así su mantenimiento y escalabilidad.

3. Reflexión técnica

El uso de ChatGPT permitió acelerar el desarrollo inicial, siendo útil para estructurar funciones básicas, generar plantillas de código y sugerir correcciones sintácticas. Sin embargo, no reemplazó el razonamiento técnico, ya que fue necesario validar manualmente los campos de `/proc/[pid]/status` utilizando documentación oficial y pruebas reales desde la terminal.

Durante el desarrollo se comprendió mejor el formato del archivo `/proc/[pid]/status`, así como el uso eficiente de `struct` en C para representar datos. También se reforzaron buenas prácticas en el manejo de errores, como la detección temprana de fallos, el uso de mensajes informativos y códigos de salida adecuados.

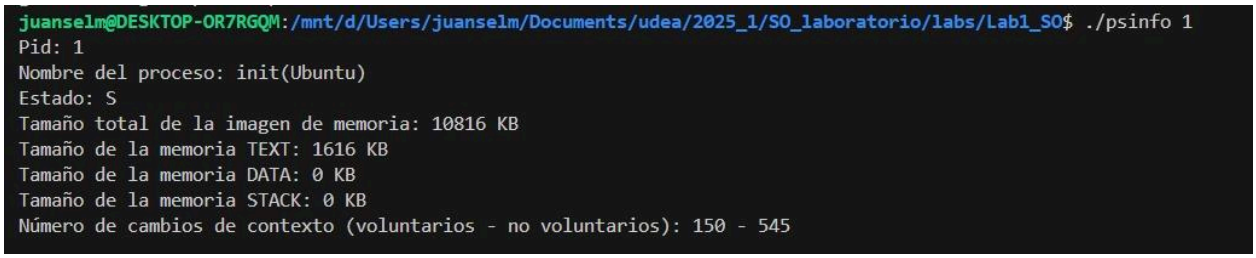
Además, se afianzó el valor de la modularización en C, mejorando la organización y mantenibilidad del código. Esta experiencia permitió integrar herramientas de IA como apoyo, sin dejar de aplicar juicio crítico y validación rigurosa en cada etapa. Esta experiencia no solo fortaleció mis habilidades en C y en entornos Linux, sino que también me dio herramientas prácticas para futuros desarrollos de sistemas que interactúen directamente con el sistema operativo.

4. Pruebas y debugging

Caso 1: PID válido

Comando: `./psinfo 1`

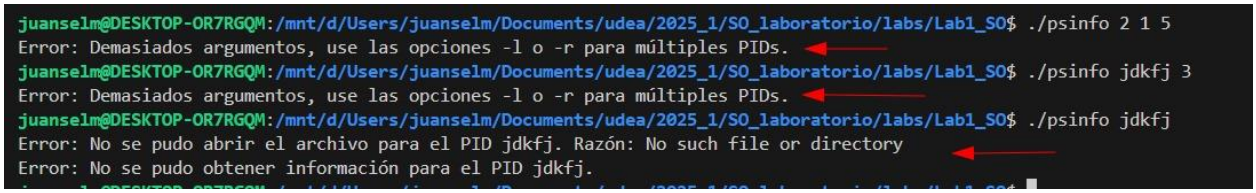
Salida esperada: Datos del proceso como nombre, estado, y memorias.



Caso 2: PID inexistente

Comando: `./psinfo 1 2 5 - ./psinfo jdkfj 3 - ./psinfo jdkfj`

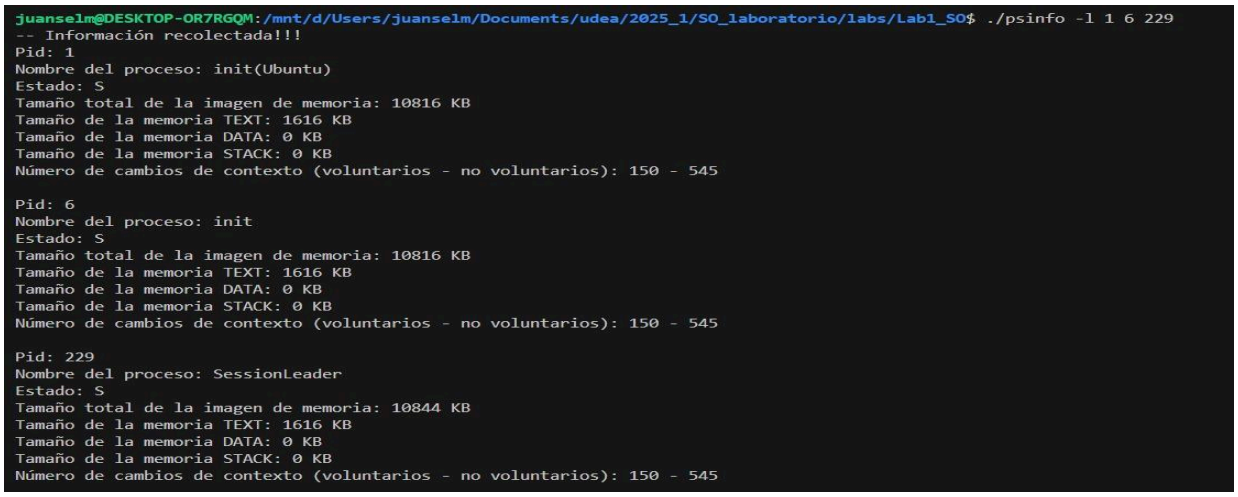
Salida: Mensaje de error.



Caso 3: Uso de opción -l con múltiples PIDs

Comando: `./psinfo -l 1 6 229`

Salida: Información recolectada de ambos procesos.



Caso 4: Uso incorrecto de opción -l con múltiples PIDs

Comando: `./psinfo -l - ./psinfo -l 1 1930 - ./psinfo -l 1 ajd`

Salida: Mensaje de error.

```
juanselm@DESKTOP-OR7RGQM:/mnt/d/Users/juanselm/Documents/udea/2025_1/SO_laboratorio/labs/Lab1_SO$ ./psinfo -l
Error: Debe ingresar al menos un PID después de -l.
Ejemplo: ./psinfo -l 2093
juanselm@DESKTOP-OR7RGQM:/mnt/d/Users/juanselm/Documents/udea/2025_1/SO_laboratorio/labs/Lab1_SO$ ./psinfo -l 1930
-- Información recolectada!!!
Pid: 1930
Error: El PID no es válido o no existe.

No se encontró información válida para los PIDs proporcionados.
juanselm@DESKTOP-OR7RGQM:/mnt/d/Users/juanselm/Documents/udea/2025_1/SO_laboratorio/labs/Lab1_SO$ ./psinfo -l ajd
-- Información recolectada!!!
Pid: ajd
Error: El PID no es válido o no existe.

No se encontró información válida para los PIDs proporcionados.
```

Caso 5: Uso de opción -r con múltiples PIDs

Comando: `./psinfo -r 1 6 229`

Salida: Archivo generado con la información solicitada.

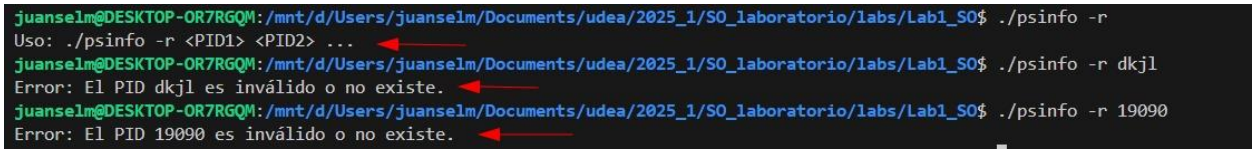
psinfo-report-1-6-229.info

```
Pid: 1
Nombre del proceso: init(Ubuntu)
Estado: S
Tamaño total de la imagen de memoria: 10816 KB
Tamaño de la memoria TEXT: 1616 KB
Tamaño de la memoria DATA: 0 KB
Tamaño de la memoria STACK: 0 KB
Número de cambios de contexto (voluntarios - no voluntarios): 150 - 545

Pid: 6
Nombre del proceso: init
Estado: S
Tamaño total de la imagen de memoria: 10816 KB
Tamaño de la memoria TEXT: 1616 KB
Tamaño de la memoria DATA: 0 KB
Tamaño de la memoria STACK: 0 KB
Número de cambios de contexto (voluntarios - no voluntarios): 150 - 545

Pid: 229
Nombre del proceso: SessionLeader
Estado: S
Tamaño total de la imagen de memoria: 10844 KB
Tamaño de la memoria TEXT: 1616 KB
Tamaño de la memoria DATA: 0 KB
Tamaño de la memoria STACK: 0 KB
Número de cambios de contexto (voluntarios - no voluntarios): 150 - 545
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS SQL HISTORY TASK MONITOR

```
juanselm@DESKTOP-OR7RGQM:/mnt/d/Users/juanselm/Documents/udea/2025_1/SO_laboratorio/labs/Lab1_SO$ ./psinfo -r 1 6 229
Archivo de salida generado: psinfo-report-1-6-229.info
juanselm@DESKTOP-OR7RGQM:/mnt/d/Users/juanselm/Documents/udea/2025_1/SO_laboratorio/labs/Lab1_SO$
```

Caso 6: Uso Incorrecto de opción -r con múltiples PIDs
Comando: `./psinfo -r - ./psinfo -r dkj1 - ./psinfo -r 19090`
Salida: Mensaje de error.



Errores detectados y solucionados:

- **Error de segmentación al usar PIDs inexistentes**: Se corrigió validando si el archivo `/proc/[pid]/status` se puede abrir antes de leer.
- **Formato incorrecto del nombre del archivo en la opción `-r`**: Se ajustó el código para concatenar correctamente los PIDs.
- **Mensajes de error genéricos o poco claros**: Se mejoraron los mensajes de error para hacerlos más informativos, facilitando la depuración y brindando mejor retroalimentación al usuario final.
- **Inconsistencias al mostrar información**: Al mostrar los datos por consola, en algunos casos no se alineaban correctamente o se omitían. Se revisó el formato de impresión para que todos los campos aparecieran ordenadamente y con títulos adecuados.

5. Sección desarrollada sin IA

La siguiente parte del código fue completamente desarrollada manualmente y no se utilizó inteligencia artificial generativa para su creación:

Código de `funciones.h`

```c
// Nueva función para leer info de un PID en una estructura
int get_proc_info(const char* pid, ProcInfo* info);

// Nueva función para imprimir la info de un ProcInfo
void print_proc_info(const ProcInfo* info);

// Nueva función para manejar la opción -l
void print_multiple_pids(int count, char* pids[]);

void generar_reporte_archivo(int count, char *pids[]);

// Función inline para validar si un PID es válido
static inline int es_pid_valido(const char *pid) {
    char path[64];
    snprintf(path, sizeof(path), "/proc/%s", pid);
    return access(path, F_OK) == 0; // Verifica si el directorio /proc/<pid> existe
}
```

El archivo `funciones.h`, que define las funciones necesarias para manejar la información de los procesos, también fue **escrito manualmente**. Incluye las declaraciones de las funciones para obtener la información de un proceso (`get_proc_info`), imprimir dicha información (`print_proc_info`), y manejar la opción de varios PIDs (`print_multiple_pids`). Además, incluye la validación de un PID con la función `es_pid_valido`, que verifica la existencia del directorio correspondiente al proceso en `/proc/<pid>`

Código de impresión de información de un proceso

.
```c
/* Función para imprimir información de un proceso */
void print_proc_info(const ProcInfo* info) {
    printf("Pid: %s\n", info->pid);
    printf("Nombre del proceso: %s\n", info->name);
    printf("Estado: %s\n", info->state);
    printf("Tamaño total de la imagen de memoria: %ld KB\n", info->vmSize);
    printf("Tamaño de la memoria TEXT: %ld KB\n", info->vmExe);
    printf("Tamaño de la memoria DATA: %ld KB\n", info->vmData);
    printf("Tamaño de la memoria STACK: %ld KB\n", info->vmStk);
    printf("Número de cambios de contexto (voluntarios - no voluntarios): %ld - %ld\n",
           info->voluntary, info->nonvoluntary);
    printf("\n");
}
```

+
```c
/* Función para imprimir información de múltiples procesos */
void print_multiple_pids(int num_pids, char* pids[]) {
    printf("-- Información recolectada!!!\n");
    int has_valid_pid = 0;

    for (int i = 0; i < num_pids; ++i) {
        if (!es_pid_valido(pids[i])) {
            printf("Pid: %s\n", pids[i]);
            printf("Error: El PID no es válido o no existe.\n\n");
            continue;
        }

        has_valid_pid = 1; // Indicar que al menos un PID es válido
        ProcInfo info;
        if (get_proc_info(pids[i], &info) == 0) {
            print_proc_info(&info);
        } else {
            printf("Pid: %s\n", pids[i]);
            printf("Error: No se pudo leer información del proceso.\n\n");
        }
    }
```

Esta sección incluye las funciones para imprimir información de un proceso específico (`print_proc_info`) y de múltiples procesos (`print_multiple_pids`). Fueron escritas sin

el apoyo de la inteligencia artificial, asegurando que cada parte de la lógica fuera desarrollada manualmente y ajustada a las necesidades del laboratorio.

6. Conclusiones

El laboratorio permitió integrar conceptos teóricos con práctica real sobre el manejo de procesos en Linux, el lenguaje C y el uso de herramientas de IA. Se logró un programa funcional, con buena estructura y control de errores. Además, la experiencia de trabajar con herramientas de IA generativa nos permitió optimizar y agilizar tareas,, pero también reforzó la importancia de la validación manual y el juicio crítico en el desarrollo de software. La modularización y el manejo adecuado de errores fueron esenciales para garantizar la robustez y mantenibilidad del código.

7. Enlace al repositorio: https://github.com/juanselm/Lab1_SO