Principios de Sistemas Operativos

Proyecto 1

Laberinto Threads

Profesora:

Ing. Erika Marín Schumann

Estudiantes:

Dayron Padilla Cubillo

Dylan Torres Walker

Instituto Tecnológico de Costa Rica

Primer Semestre del 2024

Índice

1. Introducción

El documento sirve de documentación para el primer proyecto del curso de Sistemas Operativos del primer semestre del 2024, la parte de programación pretende crear una aplicación en C para Linux que simule el recorrido de hilos sobre un laberinto. La aplicación funcionará leyendo un laberinto desde un archivo de texto, generando hilos para explorar cada bifurcación, y mostrará en pantalla el movimiento de los hilos y los espacios recorridos. A través de este proyecto, se explorará la implementación práctica de hilos en sistemas operativos y se ofrecerá una experiencia interactiva y desafiante para los estudiantes involucrados.

Este proyecto, además, tiene como meta poner en práctica los conocimientos teóricos adquiridos en el curso, mediante la implementación y exploración del funcionamiento de hilos en sistemas operativos, en este caso en Linux.

2. Estrategia de Solución

La estrategia de solución implementada se basa en la exploración del laberinto utilizando hilos. Cuando un hilo se encuentra en una posición dentro del laberinto, primero marca esa posición como visitada y la muestra en la salida estándar, representando visualmente el recorrido del hilo. Luego, verifica si esa posición es la salida del laberinto. Si lo es, el hilo termina su ejecución exitosamente. De lo contrario, el hilo crea nuevos hilos para explorar las posiciones adyacentes que aún no han sido visitadas. Este proceso se repite hasta que un hilo alcanza la salida o no hay más posiciones por explorar.

En términos generales, la estrategia de solución podría considerarse una búsqueda en profundidad (DFS) sobre el laberinto, utilizando múltiples hilos para explorar las diferentes ramas del laberinto de manera concurrente.

Para solucionar el problema de saber por dónde ha pasado algún hilo, se utiliza una matriz llamada visitado que registra las celdas visitadas por cada hilo. Esta matriz se inicializa con ceros al principio y se marca con un uno cada vez que un hilo visita una celda del laberinto. Esto permite a los hilos verificar si una celda ya ha sido visitada antes de

moverse a ella, evitando así que los hilos sigan el mismo camino más de una vez y garantizando que cada hilo explore nuevas áreas del laberinto.

Mediante el uso de mutex: Se utiliza un mutex para garantizar la exclusión mutua al modificar la matriz maze y el arreglo visitado. Esto evita que múltiples hilos intenten modificar la misma posición en el laberinto al mismo tiempo, asegurando la coherencia de los datos.

Para evitar la exploración infinita o en ciclado de los hilos, cuando un hilo alcanza una pared o el límite de la matriz, se finaliza su ejecución utilizando pthread_exit(NULL). Esto asegura que los hilos no continúen explorando más allá de los límites del laberinto y evita que queden atrapados en un bucle infinito.

Además, para mejorar la visualización se usa un system("clear") cada segundo y se pinta la matriz actualizada con los cambios: Se utiliza system("clear") para limpiar la pantalla y mostrar el laberinto actualizado con los cambios después de cada movimiento de hilo. Esto mejora la visualización del recorrido del laberinto en la salida estándar y proporciona una representación clara y actualizada del estado del laberinto mientras se realiza la exploración.

3. Análisis de Resultados:

Funcionalidad	Porcentaje de realización	Justificación
Leer laberinto de un archivo txt	100%	No aplica
Imprimir correctamente el laberinto	100%	No aplica
Actualizar el laberinto con los cambios cada segundo	100%	No aplica
Se genera el primer hilo y recorre la matriz	100%	No aplica
Cuando se dan las condiciones el hilo crea nuevos hilos	100%	No aplica

Los hilos interpretan por donde pueden o no pueden ir	100%	No aplica
Los hilos terminan su ejecución cuando se topa con la pared, meta o espacio ya recorrido.	100%	No aplica
Cuando se crea un hilo el hijo padre le pasa la dirección y la cuenta de pasos dados	100%	No aplica
Cuando los hilos terminan su ejecución imprimen un mensaje de finalización.	100%	No aplica

4. Lecciones aprendidas:

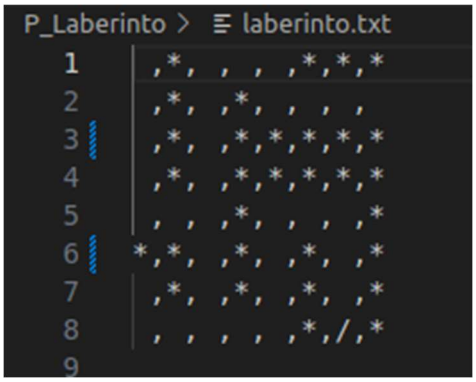
1- Un aspecto que se aprendió durante la realización de este proyecto, fue la utilización de GCC para la compilación de los archivos .c, tanto desde Linux como desde Mac, además, personalmente (Dayron) conocí o me di cuenta que este compilador GCC además de venir nativamente en Linux, también está presente en Macos de forma nativa, porque al buscar como instalarlo en Mac, fue algo innecesario ya que ya lo tenía desde un principio y hasta el desarrollo de este proyecto fue que conocí este punto.

2- Se aprendió sobre el uso de hilos en C, y también del uso de mutex como forma de sincronización entre los hilos.

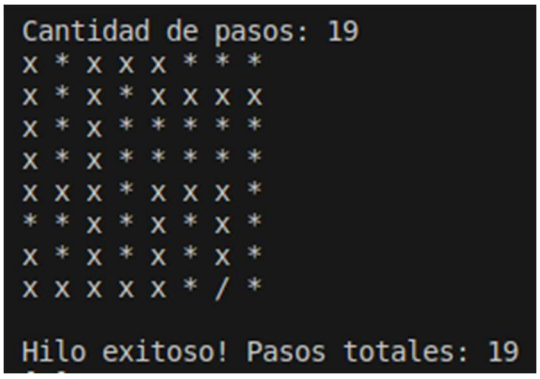
3- Otro punto que parecería una trivialidad, pero es interesante de mencionar, fue el uso de system(“clear”). Buscando formas de mejorar la visualización de la ejecución sobre el laberinto, se encontró que se puede utilizar el system(“clear”) para simular un refrescado de la pantalla.

5. Casos de pruebas:

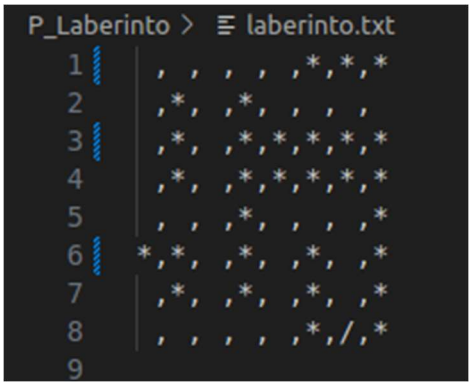
1. **Caso 1**: Ejemplo dado en el enunciado



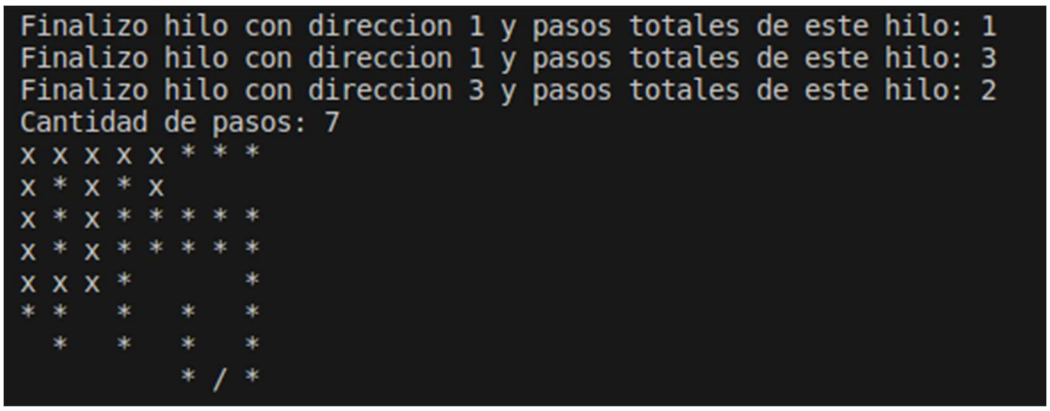
Caso dado por el enunciado del trabajo, los hilos se crean hasta encontrar la meta



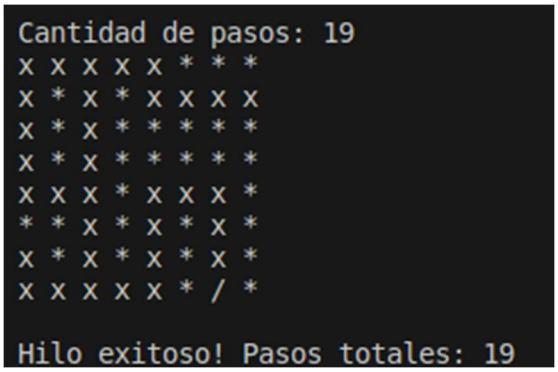
2. **Caso 2**: Doble en la ruta inicial, con intensión de provocar choque hilos



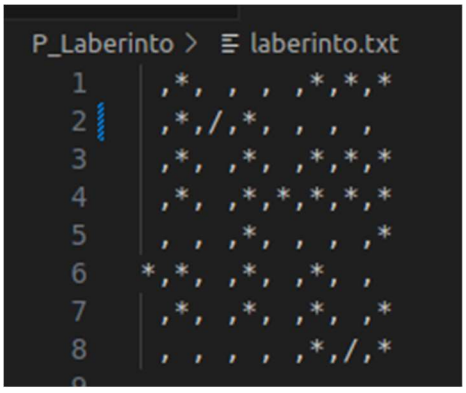
Se generan múltiples hilos desde un inicio uno recorre hacia abajo y otro hacia la derecha, choque ocasionado por los hilos hace que estos de detengan sin fallar



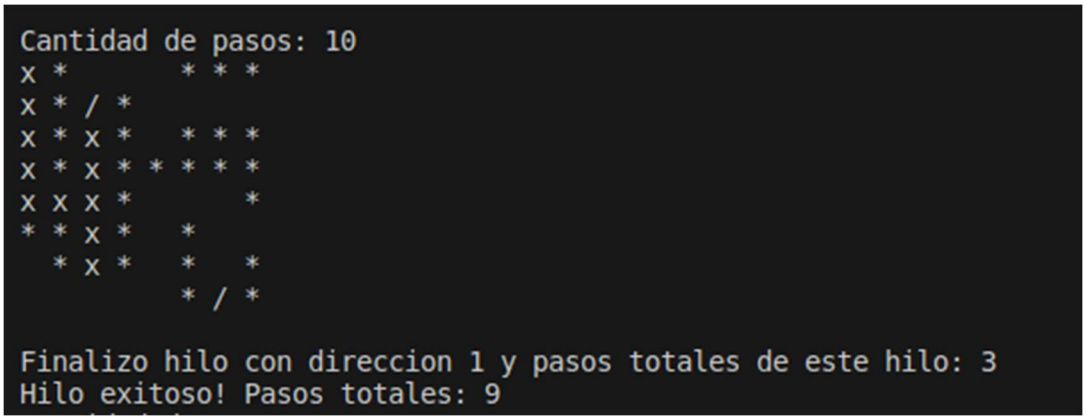
Luego continúan hasta buscar el caso de éxito



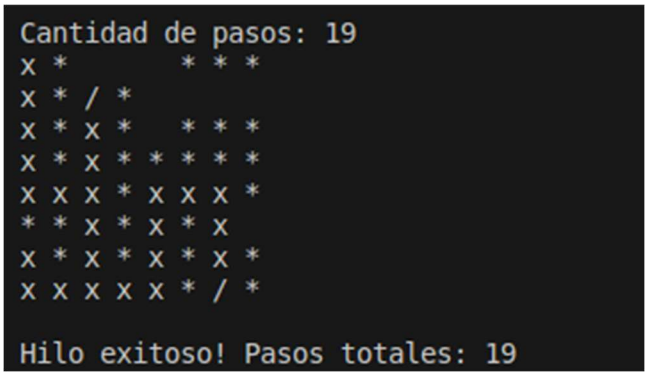
3. **Caso 3**: Doble condición de éxito



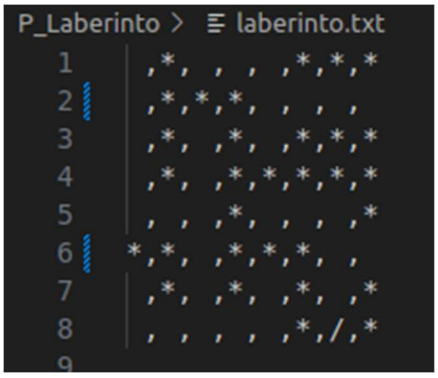
En una condición de doble éxito el mensaje de que encontró la condición se da dos veces



En nuestra solución encontrar un caso de éxito detiene el hilo que la encontró pero los demás continuar con la intensión de buscar una posible segunda solución



4. **Caso 4**: Sin camino a la salida



En este caso no presentamos un camino el cual lleve al caso de éxito por ende una vez los hilos no encuentran un camino, ellos terminan y dan por finalizada la ejecución

```
Finalizo hilo con direccion 3 y pasos totales de este hilo: 2
Finalizo hilo con direccion 2 y pasos totales de este hilo: 2
Cantidad de pasos: 12
x *       * * *
x * * *
x * x *   * * *
x * x * * * * *
x x x *       *
* * x * * *
  * x *   *   *
x x x x x * / *

Finalizo hilo con direccion 0 y pasos totales de este hilo: 1
Finalizo hilo con direccion 0 y pasos totales de este hilo: 1
```

5. **Caso 5**: Posición (0,0) ocupada antes de iniciar

```
P_Laberinto > laberinto.txt
1  *,*, , , ,*,*,*
2   ,*, ,*, , , ,
3   ,*, ,*, ,*,*,*
4   ,*, ,*,*,*,*,*
5   , , ,*, , , ,*
6  *,*, ,*, ,*, ,
7   ,*, ,*, ,*, ,*
8   , , , , ,*,/,*
9
```

En este caso el código simplemente termina debido a que no cumple con el requisito de iniciar en la posición (0,0) como disponible

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

[1] + Done                    "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm
stef@UbuntuNew:~/Desktop/Git/SO_S1_2024$
```

6. Comparación:

Si se hubieran utilizado forks en lugar de hilos para abordar el problema, habría algunas diferencias significativas en la implementación y el comportamiento del programa:

Primero, en lugar de utilizar pthread_create() para crear nuevos hilos, se utilizaría fork() para crear procesos hijos. Basándose en los expuesto por Napster, (2023), cada proceso hijo tendría su propia copia del espacio de memoria del proceso padre, lo que significa que tendrían su propia copia de las variables locales, incluida la matriz del laberinto y la matriz de visitados. Esto aumentaría el uso de memoria, ya que cada proceso hijo tendría su propia copia de los datos.

Los procesos creados mediante fork() no comparten memoria, por lo que no se puede acceder directamente a la misma matriz de laberinto o matriz de visitados entre los procesos. Se necesitaría algún mecanismo de comunicación para compartir datos entre los procesos, como pipes. (Déniz et al, 2006)

Los procesos tienen un mayor costo de creación y gestión en comparación con los hilos. Además, la coordinación y sincronización entre procesos puede ser más compleja y costosa en términos de rendimiento que con hilos, debido a la necesidad de la comunicación entre procesos y la gestión de recursos adicionales. Esto a su vez le hubiera sumado más complejidad al código.

Por último, si se hubieran utilizado forks en lugar de hilos, se habría requerido un enfoque diferente para la comunicación entre los procesos creados, y para la sincronización porque usando procesos no se podría haber utilizado mutex por ejemplo, y se tendría que haber optado por usar semáforos para sincronizar los procesos por mencionar una opción. Otra referencia consultada para esta comparativa fue: Universidad de Murcia, (2012).

7. Manual de usuario:

Usando una terminal y estando en la raíz de la carpeta P_Laberinto, además de estar en un entorno Linux con GCC, usamos el comando:

gcc main.c -o main

Este comando compilara el código c y genera un archivo llamado main que podremos ejecutar simplemente escribiendo en la terminal:

./main

Ojo: No confundir main con el archivo main.c, main.c es el código en c del programa, main es el binario generado luego de compilar, y a este archivo se le puede poner cualquier nombre, simplemente cambiando “main” por el nombre deseado el comando:

gcc main.c -o <nombreDeseado>

En caso de que se desee ejecutar con alguno de los archivos .txt adicionales para las pruebas bastara con editar la línea 23 del código main.c con el .txt deseado a utilizar y volvemos a ejecutar las instrucciones de ejecución anteriores.

8. Bitácora

Fecha	**Tareas Realizadas**	**Problemas Encontrados**	**Decisiones Tomadas**
02/04/2024	- Configurar el repositorio - Probar la ejecución de un hilo sencillo en C, en Linux - Diseñar y plantear la solución		- Dylan se encargará de programar la función para leer el laberinto desde un archivo. - Dayron se encargará de programar el comportamiento de un hilo en el laberinto

04/04/2024	- Verificar el leer el laberinto desde un archivo - Crear hilos hijos a partir de uno inicial - Actualizar el laberinto en pantalla con los cambios	- Cuando el hilo llegaba a la pared exterior se detenía y no creaba hilos en ninguna dirección	- Dayron va a arreglar el problema de que el hilo genere hijos al chocar con las paredes exteriores - Dylan va a buscar la forma de comunicar entre los hilos por donde ya se ha pasado
06/04/2024	- Hacer que los hilos terminen cuando deben y mostrar mensajes cuando terminan su ejecución - Verificar que los hilos saben si otro ya paso por donde va a pasar	- el hilo primero sobrescribía la matriz y luego preguntaba si hay un “/”, entonces nunca encontraba la meta. - Hay un conflicto para mostrar los mensajes de finalización y el system(“clear”)	- Entre los dos se van a solucionar los detalles y errores generales.
07/04/2024	- Terminar los detalles y problemas menores		

9. Bibliografía y fuentes digitales utilizadas

Déniz, O., Alexis, S., Arencibia, Q., & Santana Pérez, F. (2006). *Sistemas Operativos: Programación de Sistemas Curso 2006-07 Tema 5 : Comunicación entre Procesos mediante Tuberías*. http://sopa.dis.ulpgc.es/progsis/material-didactico-teorico/tema5_1transporpagina.pdf

Napster. (2023, January 17). *Forking Vs Threading, Thread Vs Process Python & Java - Geek Ride.* Geek Ride. https://geekride.com/fork-forking-vs-threading-thread-linux-kernel/

Universidad de Murcia. (2012). *Procesos e Hilos en C*. https://www.um.es/earlyadopters/actividades/a3/PCD_Activity3_Session1.pdf