Laboratorio 3: Sistemas Operativos

Profesor: Viktor Tapia
Ayudante de cátedra: Martín Rodriguez y Tomas Rebolledo
Ayudante de Tarea: Javiera Cárdenas y Nicolás Toro

Mayo 2024

1 Reglas Generales

Para la siguiente tarea se debe realizar un código programado en lenguaje Java. Se exigirá que los archivos se presenten de la forma más limpia y legible posible. Deberá incluir un archivo README con las instrucciones de uso de sus programas junto a cualquier indicación que sea necesaria y un archivo MAKE para poder ejecutar el programa

2 Enunciado

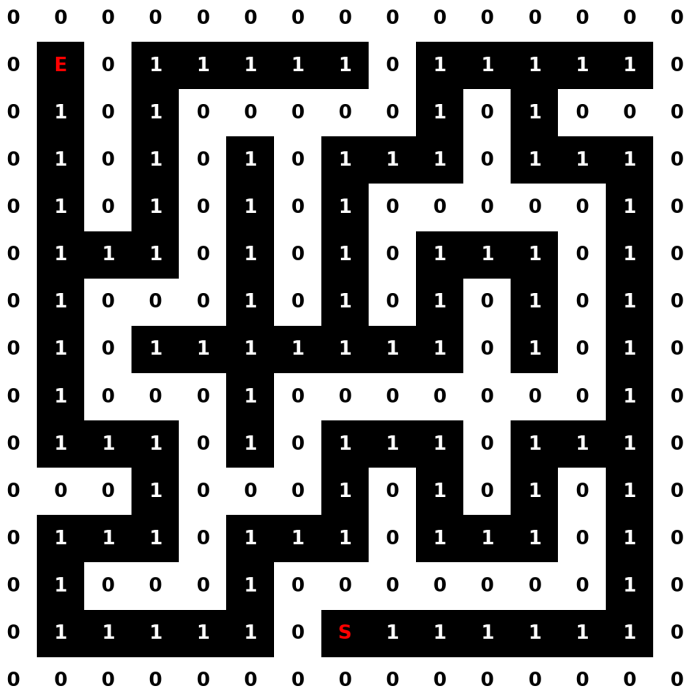
En el bullicioso centro de una gran ciudad, se encuentra la concurrida estación de Metro Baquedano. Durante la hora punta, la estación se convierte en un laberinto de pasillos, escaleras y andenes abarrotados. Los pasajeros, desesperados por llegar a sus destinos, se enfrentan al desafío de encontrar la salida correcta entre la multitud.

Un grupo de programadores decidió utilizar sus habilidades en programación paralela para ayudar a los pasajeros a encontrar la salida de la estación de manera eficiente. Dividieron la tarea en múltiples procesos y hilos, explorando diversas rutas simultáneamente para encontrar la salida más rápida.

Se le solicita resolver el laberinto dos veces una utilizando **Threads** y otra con **Fork/Join** de manera recursiva como principales herramientas.

2.1 Instrucciones

El laberinto será una matriz cuadrada de tamaño variable (por ejemplo, 10x10, 12x12, etc.) con el siguiente formato:



Ejemplo de Laberinto 15x15

El laberinto se proporcionará en un archivo .txt llamado `laberinto.txt` que contendrá lo siguiente:

1. Primera línea: Dimensión del Laberinto (NxN).
2. Segunda línea: Coordenadas donde se inicia el laberinto $[x, y]$.
3. Desde la tercera línea en adelante: Representación del laberinto.
    - **1**: Camino.
    - **0**: Muro.
    - **E**: Entrada.
    - **S**: Salida.

Deberán crear un programa en Java que utilice dos enfoques para resolver el laberinto: la clase `Multithreading` de manera recursiva y la clase `ForkJoinPool` junto con `RecursiveTask`. Para ello, deberán seguir los siguientes pasos:

- Iniciar la exploración del laberinto con el proceso principal.
- Cuando se encuentre una intersección (dos o más caminos posibles), se deben generar procesos suficientes para explorar todos los caminos posibles.
- Repetir el paso anterior hasta encontrar la salida.

- Cuando un proceso llegue a un camino sin salida, debe imprimir en pantalla su número de proceso y su ubicación en coordenadas $[x, y]$.
- Si un proceso logra encontrar la salida, debe imprimir en pantalla su número de proceso, la ubicación de la salida, y los procesos que estuvieron involucrados, incluyendo las coordenadas en las que se generaron más procesos.

Por ejemplo, si se tiene un Laberinto de 15x15 (como el caso del ejemplo), deberían existir 12 procesos, contando el proceso padre y se deberían tener las siguientes impresiones por pantalla:

```
P3- [7,1]
P5- [11,1]
P4- [7,13]- Salida
P1- [1,5]
P2- [9,13]
```

Finalmente, también se debe entregar un **excel** que incluya un breve análisis sobre si es eficiente o no la utilización de las herramientas entregadas para este ejercicio probando múltiples escenarios, tales como medir y comparar el tiempo de ejecución en [ms] con hebras, Fork/Join, agregando al menos un gráfico simple sobre los tiempos de ejecución de varios casos de prueba. Para esto, también se le solicita incluir la marca, el modelo y frecuencia del procesador de su computador junto con funcionalidades extras que este tenga (Ejemplos: Hyperthreading, Turbo boost o Overclocking), debido a que cada procesador va a presentar un tiempo de ejecución distinto. El formato del excel a entregar se encuentra en Aula y presenta todas las áreas con los datos a incluir.

2.2 Consideraciones

Se deben tener en cuenta las siguientes consideraciones a la hora de hacer la tarea

1. El laboratorio debe ejecutar solo un laberinto.
2. Los procesos solo pueden recorrer de forma vertical u horizontal el laberinto. (No en diagonal).
3. En el caso que un proceso encuentre la salida, y aun existen procesos activos, una vez estos encuentren otra intersección o terminen, se debe terminar el programa .
4. En caso de asumir algo con respecto al laboratorio, mientras cumpla las reglas, agregarlo al README.

3 Archivo de prueba

Se adjunta con el enunciado 10 archivos de prueba. Sin embargo, considere que al momento de que su tarea sea corregida, el o los archivos serán distintos (en contenido, no en formato).

4 Presentación Aleatoria

Para cada tarea, se seleccionarán grupos al azar para presentar su tarea frente a ayudantes y eventualmente profesor, recibiendo una ponderación del 80% y 20% entre tarea y presentación

respectivamente. Si su grupo presentó en una tarea, no volverá a salir nuevamente. Se comunicará días antes que grupos presentarán.

5 README

Debe contener como mínimo:

- Nombre, Rol y Paralelo de los integrantes.
- Especificación de los nombres de los archivos. (Cual corresponde a cada sección de la tarea)
- Instrucciones generales de compilación y uso.

6 Consideraciones Generales

- Se deberá trabajar de a pares. Se deberá entregar en Github en el repositorio de su grupo a mas tardar el **día 6 de Junio de 2024 a las 23:59 hrs.** Se descontarán 10 puntos por cada hora o fracción de atraso. Las copias serán evaluadas con nota 0 en el promedio de las tareas.
- La tarea debe ser hecha en Java. Se asume que usted sabe programar en este lenguaje, ha tenido vivencias con el, o que aprende con rapidez.
- Pueden crear todas las funciones auxiliares que deseen, siempre y cuando estén debidamente comentadas.
- Las tareas serán ejecutadas en **Linux**, cualquier tarea que no se pueda ejecutar en dicho sistema operativo, partirá de nota máxima 60.
- Los archivos deberán ser subidos a la carpeta correspondiente del repositorio.
- Las preguntas deben ser hechas por Aula. De esta forma los demás grupos pueden beneficiarse en base a la pregunta.
- Si no se entrega README o MAKE, o si su programa no funciona, la nota es 0 hasta la recorrección.
- Se descontarán 50 puntos por:
  - Mala implementación del Makefile.
  - No respetar el formato de entrega.
  - Solicitar edición de código para el correcto funcionamiento del Laboratorio.