Laboratorio 3: Sistemas Operativos

Profesor: Viktor Tapia
Ayudantes de cátedra: Muryel Constanzo y Nicolás Schiaffino
Ayudantes de Laboratorio: Ian Rossi y Luciano Yevenes

21 de octubre de 2025

1. Reglas Generales

Para la siguiente tarea se debe realizar un código programado en lenguaje C o C++ y 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 y ejecución de sus programas junto a cualquier indicación que sea necesaria, y un archivo MAKE para poder ejecutar el programa.

2. Tarea

Ahora que tienen repartidas las tareas, viene la parte difícil, escalar el volcán en el centro de la isla para que los pueda ver el equipo de rescate que (esperan) que llegué por aire. Escalar una ruta no explorada anteriormente es un proceso extremadamente peligroso, por lo cual no hay tolerancia a errores. Deben pensar muy bien en que ruta elegir y planearla correctamente. La idea es crear un programa que pueda analizar todas las rutas posibles y elegir la mejor.

Como analizar todas las rutas posibles y sus combinaciones puede ser un proceso caro, deberían implementarlo usando threads o forks. Luego de una sangrienta pelea entre los miembros del equipo acerca de si es mejor Java o C/C++ (aunque todos sabemos cual es el mejor realmente), el líder del equipo decide que tendrán que implementar ambas versiones, con threads para Java y forks para C/C++ y luego comparar los resultados con el primer tramo que los sacará de la playa. Con esto podrán decidir que versión del software usar durante todo el resto del trayecto.

2.1. Especificaciones

Tienen que analizar grafos que representan puntos en la pared de la montaña, y obtener el camino mas corto que los lleva desde el inicio hasta el final.

Se solicita implementar dos versiones del programa:

- **Versión en C o C++:** Esta implementación debe hacer uso de *forks* para la creación de procesos hijos, y *pipes* para la comunicación entre ellos.
- **Versión en Java:** Debe emplear *threads* para realizar la paralelización..

El objetivo final es comparar el rendimiento de ambas versiones utilizando archivos de prueba provistos o generados, enfocándose principalmente en métricas de tiempo de ejecución. Con base en esta comparación, se deberá elaborar un informe que analice cuál de las dos alternativas resulta más eficiente bajo las condiciones de la tarea.

2.2. Formato del Archivo de Entrada

El programa debe recibir como entrada un archivo de texto plano (`.txt`) que contiene un grafo especificando los nodos y sus conexiones. Este archivo debe seguir el siguiente formato:

- Línea 1: Un solo número entero, que es la cantidad total de nodos en el grafo.
- Línea 2: Un solo número entero, que es el nodo de inicio.
- Línea 3: Un solo número entero, que es el nodo final.
- Líneas siguientes: Cada línea representa una arista y contiene tres números enteros separados por espacios: `nodo_origen nodo_destino peso`.

A continuación se muestra un ejemplo de archivo de entrada:

```
12
5
9
0 8 45
0 11 15
1 2 78
1 6 33
...
```

Este grafo se interpretaría como:

- El grafo tiene 12 nodos (numerados del 0 al 11).
- La ruta más corta debe encontrarse desde el nodo 5...
- ...hasta el nodo 9.
- Las líneas siguientes describen las conexiones: hay una arista entre el nodo 0 y el 8 con un peso de 45, una entre el 0 y el 11 con peso 15, y así sucesivamente.

2.3. Implementación con Forks y Pipes

Para la versión desarrollada en *C/C++*, se utiliza la llamada al sistema `fork()` para crear procesos hijos, los cuales se encargan de realizar parte del cálculo del camino más corto. Dependiendo de la estrategia del algoritmo, cada proceso hijo puede ser responsable de revisar y relajar un subconjunto de las aristas del grafo. Esto permite aprovechar la ejecución concurrente en sistemas multiprocesador para explorar el grafo de manera distribuida.

La comunicación entre el proceso padre y sus hijos se realiza mediante *pipes*. En cada iteración del algoritmo, los hijos informan al padre sobre las actualizaciones de distancias que han encontrado. El proceso padre se encarga de sincronizar los procesos y consolidar los resultados de cada iteración. Una vez que el algoritmo converge, el padre reconstruye la ruta más corta y la escribe en un archivo de salida llamado `salidaFork.txt`.

2.4. Implementación en Java con Threads

La implementación en Java emplea *threads* para paralelizar la búsqueda del camino más corto. A cada *thread* se le asigna la tarea de procesar un subconjunto de los nodos o aristas del grafo. Por ejemplo, cada hilo calcularía las distancias tentativas desde sus nodos asignados hacia sus vecinos.

Para la sincronización y el manejo del acceso a estructuras compartidas, como el arreglo de distancias mínimas, se utilizan mecanismos de concurrencia de Java. Esto puede incluir el uso de bloques *synchronized* o estructuras de datos atómicas (como *AtomicIntegerArray*) para garantizar la coherencia de los resultados y evitar condiciones de carrera al actualizar las distancias.

Al finalizar el trabajo de todos los hilos (lo cual se puede coordinar con una barrera o *CountDownLatch*), el hilo principal se encarga de determinar el resultado final. La ruta más corta encontrada se almacena en un archivo de salida llamado `salidaThread.txt`.

2.5. Algoritmo de búsqueda a usar

Se **recomienda** que implementen el algoritmo *Bellman-Ford* ya que este es simple de paralelizar, pero están en libertad de usar otro si desean. Es necesario que especifiquen en el README cual algoritmo están usando. También es importante que ambos programas usen el mismo algoritmo para que la comparación sea justa.

2.6. Informe de Comparación y Análisis

El informe debe incluir un análisis comparativo entre las versiones desarrolladas en `C/C++` (usando *forks* y *pipes*) y en `Java` (usando *threads*). Este análisis debe basarse en pruebas reales realizadas por ambos integrantes del grupo, en sus respectivos computadores personales.

Especificaciones de los equipos de prueba

Se deben incluir las siguientes características técnicas para cada computador:

- Modelo del procesador (incluyendo cantidad de núcleos e hilos)
- Memoria RAM disponible
- Sistema operativo utilizado
- Arquitectura del sistema (32 o 64 bits)

Preguntas de análisis

El informe debe responder, como **mínimo**, a las siguientes preguntas:

1. ¿Cuál de las dos implementaciones tuvo un mejor rendimiento en términos de tiempo de ejecución? ¿A qué crees que se debe esto?
2. ¿Se observó alguna diferencia significativa en el uso de recursos del sistema (CPU, RAM) entre ambas soluciones?
3. ¿En qué escenarios consideras más adecuado el uso de procesos (`forks`) frente a hilos (`threads`)?
4. ¿Qué estrategias de optimización aplicarías al programa con menor rendimiento para que iguale o supere la eficiencia del programa mejor evaluado?

Las respuestas deben ser reflexivas y estar fundamentadas en la experiencia real durante la ejecución de las pruebas.

3. README

Debe contener como mínimo:

- Nombre, RoI y Paralelo de los integrantes.
- Especificación de los algoritmos y desarrollo realizado.
- Instrucciones de como compilar y correr el código.
- Supuestos utilizados.

4. Consideraciones Generales

- El uso de librerías está estrictamente limitado a aquellas esenciales para la implementación del laboratorio. Se permite el uso de:
  - Bibliotecas estándar básicas como `<stdio.h>`, `<stdlib.h>`, `<string.h>`, `<math.h>` en C, o `<iostream>`, `<vector>`, `<string>` en C++.
  - `unistd.h` para manejo de procesos y pipes.
  - `pthread.h` para la implementación de hilos.
  - `time.h` o bibliotecas estándar equivalentes para la medición de tiempos de ejecución.
  - Cualquier otra biblioteca de sistema necesaria exclusivamente para la gestión de `fork()`, `pipes` o `threads`.

  No está permitido utilizar librerías externas, de alto nivel o científicas que resuelvan automáticamente operaciones como la multiplicación de matrices, la paralelización o la comunicación entre procesos.

  En caso de requerir alguna librería adicional no listada explícitamente, su uso debe ser consultado previamente con los ayudantes y autorizado formalmente.
- Se deberá trabajar **OBLIGATORIAMENTE** en parejas.
- Deberá estar subido al Github correspondiente a mas tardar el día 4 de Noviembre a las 23:59 horas.
- 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 el lenguaje C o C++ y Java. Se asume que usted sabe programar en este lenguaje, ha tenido vivencias con el, o que aprende con rapidez.
- El código debe ser entregado en forma de **2 archivos**, uno `.cpp` o `.c` y otro `.java`, ambos nombrados en base al formato `"LAB1_ApellidoIntegrante1_ApellidoIntegrante2"`
- Los códigos serán pasados por un software anti-plagio, en caso de ser detectada copia o uso de una IA para la totalidad del Laboratorio, se pondrá nota 0, hasta que se realice una reunión con los grupos involucrados.

- 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.
- Las preguntas deben ser hechas por Aula a través del foro o servidor de Discord. De esta forma los demás grupos pueden verse beneficiados también.
- Si no se entrega README o MAKE, o si su programa no funciona, la nota es 0 hasta la recorrección.
- Se descontarán hasta 50 puntos por:
    - Mala implementación del Makefile.
    - No respetar el formato de entrega.
    - No respetar el formato del README.
    - Solicitar edición de código al momento de revisar.
    - Código poco prolijo y mal estructurado (ausencia de indentación adecuada, falta de consistencia en el estilo, nombres de variables confusos o poco descriptivos, comentarios insuficientes o irrelevantes).
- **Una vez publicadas las notas tendrán 5 días para apelar con el corrector que les revisó, después de este plazo las notas no tendrán ningún tipo de cambio**.