

Proyecto 2 de Sistemas Operativos

Concurrencia/sincronización y uso de profiler

Cecilia Hernández

December 11, 2020

Fecha inicio: Viernes, 11 de Diciembre, 2020.

Fecha entrega: Viernes, 8 de Enero, 2021 (a mediodía).

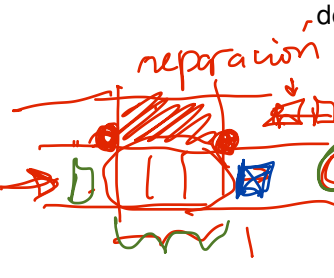
1. Objetivos

- Uso de concurrencia y mecanismos de sincronización.
- Uso de herramienta de profiling *linux perf* para análisis de memoria e identificar las porciones del código con mayor consumo de tiempo.

2. Metodología: Trabajo en grupo de 2 o 3 alumnos.

3. Descripción

- (a) Construya un simulador para el problema de control de tráfico en una ruta que está en proceso de reparación en una de sus vías. El uso de la vía disponible debe ser controlado para permitir el paso de un N autos en una dirección. Implemente un monitor tipo MESA (o sea que utiliza `while()` y no `if()` para esperar en variables de condición) para simular el control de tráfico. Para su implementación tenga en consideración los siguientes puntos:

- 
- Asuma que su monitor usa dos métodos, *PuntoEntrada(direccion)* y *PuntoSalida()*. Un auto llama a *PuntoEntrada(direccion)* al llegar a la entrada de la reparación y llama a *PuntoSalida()* cuando sale.
 - Potencialmente, un auto que sale de la sección de la ruta en reparación puede habilitar a un auto esperando entrar.
 - Utilice un solo tipo de hebra para simular los autos.
 - La implementación debe estar libre de bloqueo mortal y condiciones de carrera.

Se pide:

- Proporcionar el algoritmo del monitor.
- Analizar el algoritmo y determinar si tiene algún problema bajo algún escenario.
- Implementar el simulador usando `pthread` o `threads` de C++.

- (b) Desarrolle la multiplicación de matrices con dimensión N mas simple con algoritmo $O(N^3)$ y luego implemente dos alternativas adicionales también $O(N^3)$. Utilice memoria dinámica para almacenar las matrices y opciones de optimización `O3` para compilar. Luego mida las siguientes características con *linux perf*. Realice sus experimentos con N variando en potencias de 2 desde 2 a 2048.

- `cpu-cycles`
- `instructions`
- • `L1-dcache-loads`
- • `L1-dcache-load-misses`
- • `L1-dcache-stores`
- • `LLC-load-misses`

$1 \ll \text{exp}$

`perf stat -e`

`g++ -O3 nexec prog.cpp -O3`

`man g++`

\$ perf stat -e cpu-cycles, instructions ./main 10
2'10 2'10 x 2'10

- • LLC-store-misses
- • dTLB-load-misses
- • dTLB-store-misses

Considere el siguiente algoritmo clásico de multiplicación de matrices.

```
for i=0 to N
  for j=0 to N
    for k=0 to N
      C[i][j] += A[i][k]*B[k][j]
```

baseline

- Construya gráficos que permitan comparar los resultados de las tres implementaciones. Si considera que algunas de las métricas pueden coexistir en un mismo gráfico para hacer una mejor discusión siéntase libre de hacerlo.

sds (4) Ahora utilice linux perf para averiguar las principales funciones que ocupan el mayor porcentaje de uso de CPU del internet browser que utilice. Además, compute el número de major faults and minor faults ejecutando el browser por primera vez, y luego ejecutando una vez mas el browser. Investigue representen los dos tipos de fallos (minor y major).
perf record firefox

Evaluación

La evaluación final consiste en una nota ponderada con 80% funcionamiento, 10% estructura del código y 10% informe. Su informe debe incluir sus algoritmos, análisis y discusión.

perf stat -e minor-faults, major-faults firefox &
dejar abierto firefox
perf stat -e minor-faults, major-faults firefox &