Universidad Católica Andrés Bello

Escuela de Ingeniería Informática

Sistemas Operativos

**Informe del Proyecto #1**

Integrantes:

Miguel Ordoñez

César Bonadío

Las mediciones mostradas a continuación están expresadas en microsegundos.

Mediciones para los hilos

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **C.Hilos** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|  | 240 | 399 | 909 | 464 | 1149 | 818 | 1322 | 1144 | 1698 | 1248 |
|  | 269 | 476 | 684 | 725 | 664 | 793 | 655 | 847 | 1280 | 1217 |
|  | 261 | 420 | 415 | 469 | 761 | 766 | 1228 | 1148 | 1056 | 1287 |
|  | 263 | 411 | 384 | 771 | 709 | 768 | 919 | 1084 | 1211 | 1130 |
|  | 303 | 466 | 827 | 651 | 666 | 896 | 791 | 989 | 1261 | 1022 |
|  | 260 | 412 | 511 | 556 | 661 | 763 | 868 | 780 | 1184 | 1368 |
|  | 274 | 424 | 469 | 646 | 829 | 1116 | 973 | 1030 | 1266 | 1262 |
|  | 277 | 525 | 523 | 455 | 550 | 942 | 782 | 1348 | 1205 | 1221 |
|  | 260 | 441 | 504 | 668 | 1022 | 970 | 980 | 1135 | 1166 | 1445 |
|  | 266 | 450 | 800 | 750 | 1003 | 895 | 900 | 1100 | 1200 | 1250 |
| **Promedio** | 267,3 | 442,4 | 602,6 | 615,5 | 801,4 | 872,7 | 941,8 | 1060,5 | 1252,7 | 1245 |

Mediciones para los procesos

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **C.Procesos** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|  | 1084 | 1788 | 2870 | 1544 | 2514 | 4033 | 4442 | 4493 | 4766 | 5626 |
|  | 1299 | 1070 | 2135 | 2066 | 1400 | 2364 | 3313 | 4609 | 5566 | 5644 |
|  | 1403 | 1491 | 1166 | 2580 | 2345 | 2940 | 3364 | 5311 | 4628 | 3373 |
|  | 1672 | 1652 | 1306 | 1595 | 1206 | 3610 | 3876 | 4475 | 6742 | 5678 |
|  | 1569 | 1162 | 1013 | 2348 | 1506 | 3655 | 4095 | 4572 | 4723 | 4168 |
|  | 1654 | 1792 | 1277 | 2318 | 2320 | 3237 | 4604 | 4526 | 4339 | 5622 |
|  | 1958 | 1028 | 2014 | 1295 | 4653 | 3553 | 4031 | 4224 | 5297 | 6253 |
|  | 1044 | 1640 | 1160 | 2937 | 2903 | 3139 | 3995 | 4737 | 5474 | 6106 |
|  | 1831 | 1984 | 1880 | 3803 | 2354 | 4306 | 3975 | 4065 | 5454 | 6497 |
|  | 1503 | 1510 | 1725 | 2150 | 2744 | 3532 | 4200 | 4623 | 4256 | 5642 |
| **Promedio** | 1501,7 | 1511,7 | 1654,6 | 2263,6 | 2394,5 | 3436,9 | 3989,5 | 4563,5 | 5124,5 | 5460,9 |

Gráficas

Leyenda:

* Procesos
* Hilos

Microsegundos en función de N

Conclusiones sobre las mediciones

Se realizaron mediciones de tiempo para la ejecución del programa con procesos e hilos. Para realizar dichas mediciones de tiempo se utilizó un archivo de entrada “entrada.txt” con 150 números.

Como se puede ver en la gráfica, se realizaron las medidas del tiempo de ejecución variando la cantidad de hilos o procesos entre el rango permitido de entrada (1-10), para cada número de procesos e hilos se realizaron 10 medidas y se calculó el promedio por el simple hecho de que los resultados pueden variar a grandes rangos por tratarse de operaciones que se hacen con gran rapidez y que son medidas en microsegundos, además influyen ciertos factores más como el estado del sistema operativo o las políticas del planificador. A pesar de que existen mediciones de una misma cantidad de hilos o procesos con un amplio rango de diferencias, el promedio de 10 medidas refleja la diferencia que hay entre ejecutar el programa con hilos o con procesos.

Como los hilos comparten memoria y son creados por un proceso padre, tienden a durar mucho menos que los procesos, esto se debe a que los procesos tienen autonomía de ejecución y su proceso padre debe esperar a que los demás terminen de ejecutarse, en los hilos no, en los hilos el proceso principal del programa va creando los hilos, estos luego se unen al proceso principal y finaliza el programa. Los hilos a su vez, al igual que los procesos, aumentan su tiempo de ejecución conforme se aumente N, claro está, que al ejecutar el programa con N hilos el tiempo va a ser menor que con el mismo N procesos.

También se puede observar que mientras mayor cantidad de procesos e hilos, mayor diferencia va a haber en su tiempo de ejecución individual y total. Además, mientras más números tengan que ser procesados, por tratarse de unidades de tiempo medidas en microsegundos, la diferencia va a ser más notoria.

Otro factor que también afecta el tiempo de ejecución de los hilos, es que en los hilos es necesario asignar un espacio de memoria a la estructura que contiene los intervalos de lectura e interpretación que le corresponde a cada hilo, esto se debe a que para llamar una función ejecutada por hilos, el parámetro que se le pasa debe ser una dirección de memoria o un puntero y no se le puede asignar una estructura por referencia porque como cada hilo se ejecuta en paralelo , si no se pasa por valor, cada hilo modifica desordenadamente el intervalo que le corresponde.

Código

***En el problema 1, 2 y 3*** se crearon funciones separadas para los hilos y procesos. Los hilos hacían una función distinta a la de los procesos, ya que los parámetros deben ser distintos. Esto se debe a las primitivas de inicialización de cada uno.

Para asignar intervalos de lectura a cada hilo o proceso es usada una estructura global con tres datos, un entero desde, un entero hasta y el número de hilo o proceso. Para ello, en caso de los procesos, solo es asignados sus valores a la estructura y pasados por parámetro a la función de procesos, sin embargo, en los hilos es completamente distinto porque es necesario asignar un espacio de memoria y pasarla como parámetro a la función de hilos porque el parámetro que te pide la creación de hilos es necesario que sea una dirección de memoria o cualquier tipo de puntero.

Las mediciones realizadas durante la entrega del proyecto fueron las siguientes:

* Con 5 hilos, 11345 microsegundos. Con 5 procesos 12456 microsegundos.
* Con 10 hilos, 16254 microsegundos. Con 10 procesos 14256 microsegundos.

Como se puede ver, durante la presentación del proyecto, los tiempos de ejecución variaron y se determinó que los hilos no presentaban concurrencia, en cambio los procesos sí se presentó concurrencia. Esto se debe a que el join de los hilos está dentro del ciclo donde se crean estos y por lo tanto espera la terminación de cada uno de manera secuencial, evitando así que haya **concurrencia**. Por eso se puede ver reflejado en el caso de los 10 hilos y 10 procesos, donde se supone que el resultado de los hilos debería ser menor que el de los procesos (porque hay un cambio de contexto más liviano), sin embargo, esto no ocurre, debido a que los procesos se están ejecutando de manera concurrente y los hilos de manera secuencial, limitando el gran beneficio de los hijos, el cual es el cambio de contexto rápido.

***En el problema 4*** (semáforos), se hizo uso de 3 semáforos con el objetivo de que los 3 procesos creados (**BisLeft, BisRigth y Sendero**) se ejecutaran siempre en un *mismo orden.* Por ejemplo: BisRigth, BisLeft, Sendero, BisRigth, BisLeft, Sendero, y así sucesivamente.

**“BisLeft”** se encarga de tomar la hora de demanda de una bicicleta cuyo sentido es el izquierdo. Cabe destacar que este proceso tiene un variable global “posI” con la que lleva el control para obtener una bicicleta en ese sentido.

**“BisRigth”** se encarga al igual que **BisLeft** de tomar la demanda de una bicicleta, a diferencia que esta tiene un sentido contrario, es decir, derecha. También usa de una variable global “posD” para llevar un control de la bicicleta que se obtiene.

**“Sendero”** se encarga de hacer las respectivas validaciones y comparaciones entre la hora de la bicicleta izquierda y derecha, obtenidas respectivamente por **BisLeft, BisRigth** indicando cuál de las dos bicicletas tiene mayor prioridad permitiendo su paso por el sendero.

Este orden de ejecución de logra con un uso de par de estos 3 semáforos, es decir, 2 de 3 semáforos en cada proceso. En la tabla a continuación se puede observar mejor la explicación.

Semáforo\_A = 1 Semáforo\_B = 0 Semáforo\_C = 0

|  |  |  |
| --- | --- | --- |
| down(Semáforo\_A) | down(Semáforo\_B) | down(Semáforo\_C) |
| BisRigth | BisLeft | Sendero |
| up(Semáforo\_B) | up(Semáforo\_C) | up(Semáforo\_A) |

De esta manera se asegura que solo un proceso esté en ejecución y que siempre se repita el orden anteriormente descrito (BisRigth, BisLeft y Sendero), debido que cada proceso depende del proceso anterior, explicado de una mejor manera:

Cuando se ejecuten los tres procesos, tanto el proceso **BisLeft** “down(Semáforo\_B)” como **Sendero** “down(Semáforo\_C)”, van a disminuir su semáforo quedando en un valor negativo y en consecuencia durmiendo al proceso, quedando como el único en ejecución el proceso **BisRigth**, debido que al disminuir su semáforo “down(Semáforo\_A)” este queda en 0 y por lo tanto continua su ejecución; este al finalizar se encarga de despertar al proceso **BisLeft** con el up(Semáforo\_B) y el proceso **BisRigth** se duerme, debido que se vuelve a repetir y esta vez pasando el valor de su semáforo de 0 a -1. Esta misma lógica, se repite en el proceso **BisLeft**, pero despertando al proceso Sendero y este último lo mismo, pero, despertando al proceso **BisRigth**, repitiendo nuevamente este ciclo.

Es importante destacar que para poder llevar a cabo las diferentes comparaciones como lo fue la comparación entre las horas de demanda obtenidas por los dos procesos que obtienen las bicicletas (una en cada sentido), se necesitó del uso de memorias compartidas con el fin de que una “misma variable” pudiera ser escrita y leída desde cualquier proceso, y verse reflejado cualquier cambio en TODOS los procesos.