INSTITUTO TECNOLÓGICO AUTÓNOMO DE MÉXICO

LABORATORIO: Sistemas Operativos

Práctica 8

Threads, programación concurrente

LosDos

Integrantes:

Amanda Velasco Gallardo - 154415
Carlos Octavio Ordaz Bernal - 158525

Fecha(s) de elaboración de la práctica:

22 de marzo de 2019

Introducción

El *program counter* es aquel que contra el flujo secuencial de ejecución que sigue el CPU. Así, un *hilo* o *thread* de ejecución es el recorrido secuencial de ejecución del CPU de principio a fin.

Los procesos que conocíamos con anterioridad se dividen en procesos pesados y ligeros. Por un lado, los primeros son aquellos que tienen un solo *thread* principal de ejecución conocido como *main*. Por otro lado, los segundos corren dentro del contexto del proceso pesado y toman ventaja de los recursos y del ambiente en que se encuentran instanciados.

Los *thread* son la unidad de planificación mientras que los procesos son sólo los contenedores donde los *hilos* se ejecutan. La principal ventaja de trabajar con *thread* es que se puede compartir información entre ellos, además resulta más barata su creación con respecto a la de los procesos concurrentes. Los *thread* comparten dentro del proceso: área de código, área de datos, y recursos del sistema operativo.

Los sistemas operativos actuales tienen la capacidad de trabajar con múltiples *hilos* de ejecución, a esta técnica se le conoce como *multi-threading*. Hay un *hilo* principal (*main*) por donde se inicia la ejecución del proceso, y es el encargado de crear y arrancar los demás *threads*.

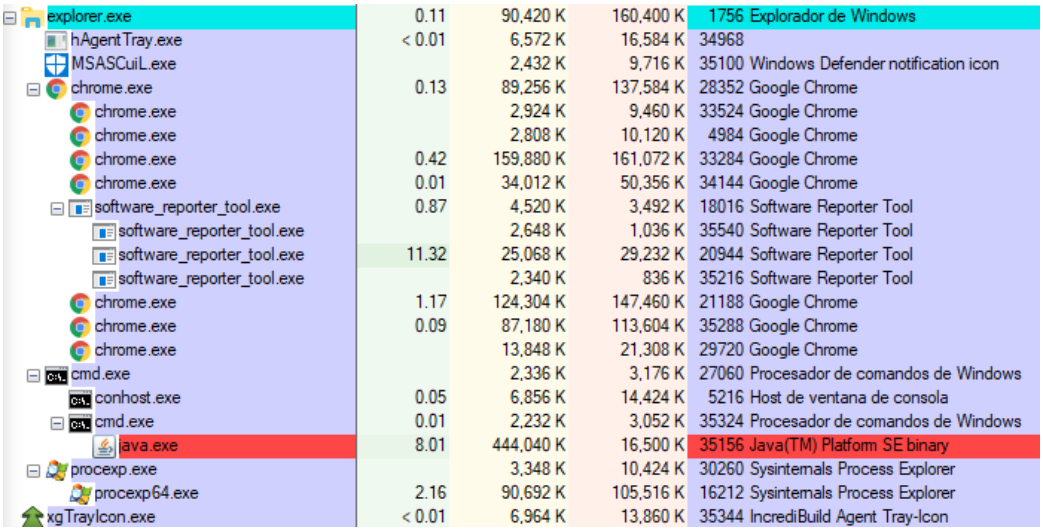
Desarrollo

Recolecte 3 tiempos de ejecución tanto de la aplicación concurrente con procesos pesados (Práctica BG07) como otros 3 de la actual práctica BH08. Tome muestras tanto del padre como de algún hijo.

- Práctica BG07: En tres corridas el padre tardó 1053, 748 y 1059 milisegundos respectivamente, mientras que el hijo tardó siempre 4 milisegundos.

- Práctica BH08: En tres corridas el padre tardó 14, 28 y 15 milisegundos respectivamente, mientras que el hijo tardó 4, 5 y 7 milisegundos.

Con *Process Explorer* capture un "pantallazo"de la ejecución de esta aplicación.
En la siguiente imagen 1, se puede observar solamente un proceso de `java`. Los *threads* no se muestran ya que ocurren dentro del mismo proceso padre.



explorer.exe	0.11	90,420 K	160,400 K	1756	Explorador de Windows
hAgentTray.exe	< 0.01	6,572 K	16,584 K	34968	
MSASCuiL.exe		2,432 K	9,716 K	35100	Windows Defender notification icon
chrome.exe	0.13	89,256 K	137,584 K	28352	Google Chrome
chrome.exe		2,924 K	9,460 K	33524	Google Chrome
chrome.exe		2,808 K	10,120 K	4984	Google Chrome
chrome.exe	0.42	159,880 K	161,072 K	33284	Google Chrome
chrome.exe	0.01	34,012 K	50,356 K	34144	Google Chrome
software_reporter_tool.exe	0.87	4,520 K	3,492 K	18016	Software Reporter Tool
software_reporter_tool.exe		2,648 K	1,036 K	35540	Software Reporter Tool
software_reporter_tool.exe	11.32	25,068 K	29,232 K	20944	Software Reporter Tool
software_reporter_tool.exe		2,340 K	836 K	35216	Software Reporter Tool
chrome.exe	1.17	124,304 K	147,460 K	21188	Google Chrome
chrome.exe	0.09	87,180 K	113,604 K	35288	Google Chrome
chrome.exe		13,848 K	21,308 K	29720	Google Chrome
cmd.exe		2,336 K	3,176 K	27060	Procesador de comandos de Windows
conhost.exe	0.05	6,856 K	14,424 K	5216	Host de ventana de consola
cmd.exe	0.01	2,232 K	3,052 K	35324	Procesador de comandos de Windows
java.exe	8.01	444,040 K	16,500 K	35156	Java(TM) Platform SE binary
procexp.exe		3,348 K	10,424 K	30260	Sysinternals Process Explorer
procexp64.exe	2.16	90,692 K	105,516 K	16212	Sysinternals Process Explorer
xgTrayIcon.exe	< 0.01	6,964 K	13,860 K	35344	IncrediBuild Agent Tray-Icon

Fig. 1: Captura del árbol de procesos en `ProcessExplorer`.

En la práctica 07, ¿cuántos y cuáles son los programas ordinarios que conforman el programa concurrente? En la misma práctica 08, ¿cuántos procesos ligeros (threads) y cuántos tradicionales o pesados fueron ejecutados?
En la práctica 7 hay dos programas ordinarios, `EdoCtaClientes` y `MovimientosCliente`. Sin embargo, en la práctica 8 ya solamente hay un proceso pesado (`EdoCtaClientesC`) que hace uso de otros 10 procesos ligeros (el thread `MovimientosClienteC`), uno por cada cliente.

De los resultados recolectados, ¿cómo justifica que este programa concurrente con *threads* es más barato en recursos y más rápido?
Porque el proceso principal no gasta tiempo extra en la creación de cada uno de los procesos hijos. En el caso de esta práctica los *threads* resultaron mucho más baratos bajando el tiempo de aproximadamente 1000 milisegundos a 15 milisegundos.

Conclusiones

El desarrollo de esta práctica nos permitió conocer de manera más detallada la forma en que se generan los *hilos* dentro del sistema operativo. El programar el código del archivo `EdoCtaClientesC.java` junto con su clase

privada `MovimientosClienteC` nos dejó profundizar en la manera en que se crean los *hilos* de un proceso para que estos distribuyan, de manera modular, la tarea principal del *hilo* principal. Pudimos comparar que, a diferencia de lo que ocurría con los procesos concurrentes, los *hilos* son mucho más rápidos y baratos. Por útlimo, utilizando `ProcessExplorer`, visualizamos la jerarquía que tienen los procesos dentro del sistema operativo, y la manera en que la máquina virtual de `Java` los aloja.

Referencias

- Ríos, J. (2019). Notas del curso de Sistemas Operativos. Recuperado el 19 de febrero de 2019, del sitio web: Comunidad ITAM.