Universidad Simón Bolívar

Departamento de Computación y T.I.

Taller de Sistemas de Operación I

Septiembre-Diciembre 2014

Proyecto 1: Programación Concurrente (15%)

I. Objetivos

- Utilizar las primitivas fork, wait, exec y exit para el manejo de procesos concurrentes.
- Utilizar las primitivas de hilos (threads) para el manejo de procesos concurrentes.
- Desarrollar un mecanismo de comunicación de de memoria compartida para la comunicación entre los hilos
- Manejar primitivas que permitan medir el tiempo de ejecución de los procesos/hilos.
- Manejar primitivas para el manejo de archivos binarios y texto.

II. Enunciado

Ud. deberá implantar en C dos versiones (una usando procesos y otra usando hilos) de un programa que ordenará una larga secuencia de enteros almacenados en un archivo. El ordenamiento de los enteros se realizará con un árbol binario de procesos/hilos a n niveles. Los procesos/hilos hojas ordenan y los procesos/hilos intermedios mezclan las secuencias ordenadas de sus procesos/hilos hijos. Es decir, cada proceso/hilo hoja recibe una secuencia de enteros desordenada, la ordena, y entrega a su proceso/hilo justo superior (padre inmediato) la secuencia ordenada. Este proceso/hilo intermedio recibe las dos secuencias ordenadas de sus procesos/hilos hijos, las mezcla, entrega a su padre inmediato y así sucesivamente. Al final, el proceso/hilo raíz recibe las últimas dos secuencias ordenadas de enteros, las mezcla manteniendo el orden, y guarda el resultado en un archivo final. La línea de comandos se realizará de la siguiente forma:

```
$ ordenArchivo-p NumEnteros NumNiveles ArchivoEnterosDesordenado
ArchivoEnterosOrdenado
```

(para la versión usando procesos)

```
$ ordenArchivo-t NumEnteros NumNiveles ArchivoEnterosDesordenado
ArchivoEnterosOrdenado
```

(para la versión usando hilos)

Donde:

- **NumEnteros:** es la cantidad total de enteros a ordenar, es decir el número de elementos en el archivo ArchivoEnterosDesordenado.
- **NumNiveles:** es la profundidad del árbol de procesos/hilos
- **ArchivoEnterosDesordenado:** es el nombre del archivo que almacena los enteros en forma desordenada.
- **ArchivoEnterosOrdenado:** es el nombre del archivo donde al final del programa se almacenarán los enteros ordenados.

III. Salida del Programa

La salida del programa se compone de un único archivo (ArchivoEnterosOrdenado) donde se almacenarán todos los enteros ya ordenados. Además, todos los procesos/hilos no hojas deben imprimir en pantalla el tiempo que tardó realizando su trabajo.

IV. Detalles de Implementación

Para la implementación con procesos:

Inicialmente el proceso raíz reparte el trabajo entre sus dos hijos en partes iguales siempre que sea posible. En cualquiera caso, a cada proceso hijo le tocará una porción de enteros a ordenar, ni. Sucesivamente cada proceso realiza la misma división entre sus dos hijos con sus respectivos ni enteros, hasta llegar a las hojas. Cada proceso recibirá (heredará) de su proceso padre inmediato los siguientes valores:

- El nombre del archivo donde se encuentran los enteros desordenados.
- El número de enteros a ordenar: ni
- A partir de qué elemento del archivo comienzan sus elementos: inicio.

Ejemplo: Suponga que el archivo inicial tiene 18 elementos a ordenar y se indicó 3 niveles de procesos. El proceso raíz indicará a su primer hijo, ni=9 e inicio=1, mientras que a su segundo hijo le indicará, ni=9 e inicio=10. El primer hijo del segundo nivel, asignará a su primer hijo, ni=4 e inicio = 1 y a su segundo hijo ni=5 e inicio=5. Mientras que el segundo hijo del segundo nivel, asignará a su primer hijo, ni=4 e inicio = 10 y a su segundo hijo ni=5 e inicio=14. Note que 2 procesos hojas ordenarán 4 elementos y los otros dos ordenarán 5 elementos.

Asegúrese de que cada proceso realiza los cálculos necesarios para que al crear a sus hijos, éstos hereden los valores de ni e inicio que le correspondan.

Una vez que los procesos hojas terminen de ordenar su parte, crean un archivo (cuyo nombre será su PID.txt) que contenga los elementos ordenados, imprimen por salida estándar el tiempo que tardaron realizando su trabajo y terminan.

Los procesos intermedios leen los archivos correspondientes a sus hijos, mezclan las secuencias de enteros ordenados usando el algoritmo merge sort, crean sus correspondientes archivos (PID.txt), eliminan los archivos generados por sus hijos, imprimen por salida estándar el tiempo que tardaron realizando su trabajo y terminan. Finalmente, todos los enteros ordenados los escribirá el proceso raíz en el archivo ArchivoEnterosOrdenado que se recibe como argumento de llamada.

En http://www.ldc.usb.ve/~yudith/docencia/ci-3825-taller/Ejemplo_Tiempo.c encontrará un ejemplo del uso de rutinas para tomar tiempos.

- **Los procesos hojas**

El código de los procesos hojas debe estar escrito en el archivo **hoja.c (y se debe generar un ejecutable distinto a ordenArchivo que es el ejecutable del proceso raíz).** Cada uno de los procesos hojas abrirá el archivo inicial (cuyo nombre recibe de su proceso padre inmediato), tomará los enteros que les corresponda (revise la función de librería fseek que le permitirá desplazarse dentro del archivo) y los ordenará, en la memoria, utilizando el algoritmo quick sort. Una vez ordenados los enteros en la memoria, crea un archivo (cuyo nombre será su PID.txt) que contenga los elementos ordenados, imprime por salida estándar el tiempo que tardó realizando su trabajo y termina.

Para la implementación con hilos:

La implementación con hilos tiene algunas diferencias respecto a la implementación con procesos:

- La comunicación debe hacerse a través de pase de parámetros al momento de crear los hilos y de estructuras de datos compartidas entre los hilos, haciendo uso eficiente de la memoria (recuerde que el número de hilos crece exponencialmente). Note que en la versión de hilos no se deben crear archivos intermedios.
- Para la creación de hilos se sugiere crear los 2n – 2 hilos (donde n es el número de niveles y el hilo del main representa el nodo raíz del árbol) desde el hilo principal distinguiendo los hilos intermedios de los hilos hojas.
- Inicialmente el hilo raíz (hilo del main) reparte el trabajo entre los 2(n-1) hilos hojas en partes iguales de ser posible. En cualquiera caso, a cada hilo hoja, le tocará una porción de enteros a ordenar ni y un inicio. Estos dos valores los establece el hilo raíz antes de crear todos los hilos. Ejemplo: Suponga que el archivo inicial tiene 18 elementos a ordenar y se indicó 3 niveles de hilos. Esto significa que tendremos 4 hilos hojas (23-1), de los cuales tres ordenarán ni=4 enteros y un hilo ordenará el resto, es decir, ni=6 enteros. Esto implica que un hilo hoja cargará con los enteros que resten de la repartición si la división no es exacta. Los valores de inicio serán, inicio=1, inicio=5, inicio=9 e inicio=13, respectivamente.
- El árbol binario será simulado de acuerdo a los join que deberán realizar los hilos intermedios. Los hilos hojas usarán el algoritmo quick sort para ordenar.

V. Formato de los Archivos

El archivo de entrada (ArchivoEnterosDesordenado), estará en formato binario. En este tipo de archivos la información está almacenada por registros contiguos, todos de un mismo tipo de dato, que se pueden acceder aleatoriamente. El contenido de estos archivos no puede verse desde su terminal con los comandos type, more, etc.

En http://www.ldc.usb.ve/~figueira/Cursos/ci3825/taller/material/Ejemplos_C/Archivos/Archivos.c , encontrarán un ejemplo donde se generan, leen y escriben registros a un archivo binario. En el ejemplo, los registros son estructuras, en el caso de este proyecto los elementos no son estructuras sino enteros. Noten que tienen que utilizar fwrite para generar su archivo de prueba.

El archivo de salida que genera el proceso raíz (ArchivoEnterosOrdenado) si estará en formato texto . Es decir este último archivo se generará con las rutinas de librería: fprintf, fputc, fputs, putchar, etc; y su contenido podrá observarse desde la cónsola haciendo, por ejemplo:

```
$ cat ArchivoEnterosOrdenado
```

El acceso a este archivo es únicamente de forma secuencial. Esto último es fundamental para efectos de corrección.

VI. Observaciones

- El archivo de entrada se supone correcto (sólo contiene enteros).
- Deben escribir un makefile para la compilación óptima.
- Deben estructurar modularmente el programa usando archivos .h y manejar correctamente los posibles errores de las llamadas al sistema.
- Recuerde validar las llamadas al sistema con perror() y verificar los argumentos de entrada; por ejemplo que NumEnteros y NumNiveles deben se enteros positivos, que NumEnteros sea mayor al número de hojas, etc.

VII. Entrega

La entrega del proyecto está pautada para el martes de la semana 7. Ese día Ud. deberá:

- Subir a Aula Virtual un archivo tar.gz que contenga únicamente los fuentes (archivos .c y .h) de su proyecto y un archivo Makefile para generar los ejecutables.
- Un informe de no más de 3 páginas explicando los algoritmos utilizados (no detallar ni quick sort ni merge sort) más bien explique el manejo de la memoria compartida, parámetros de llamada a los procesos/hilos, comunicación de procesos/hilos, etc. Compare hilos y procesos en función a la memoria utilizada, tiempo de ejecución, facilidad de programación, etc. Agregar al informe una tabla de tiempo promediando al menos 10 ejecuciones, para comparar las dos versiones del proyecto. Calcule tiempo de tal manera que justifique las respuestas anteriores y sus conclusiones. Compare las ejecuciones con archivos que contengan 25 y 215 enteros, con 5 y 15 niveles. Especifique la arquitectura que usó para realizar las ejecuciones (Intel dual core, dual processor, etc.).