Pontificia Universidad Javeriana

Departamento de Ingeniería de Sistemas
Sistemas Operativos, SIU 4085.
julio-diciembre 2019

Proyecto 1: Sort Concurrente

20%

Objetivo General del Proyecto

El objetivo de este proyecto es que el estudiante resuelva un problema en forma concurrente utilizando procesos e hilos. Una vez realizada la implementación, el estudiante comparará el desempeño de los dos tipos de entidades concurrentes. También tendrá la posibilidad de comparar su propia implementación de ordenamiento con un comando del sistema que realiza la misma función.

El problema a Resolver

El comando de UNIX ***sort*** toma los archivos que aparecen en su lista de parámetros de entrada y ordena sus líneas. La ordenación se realiza de acuerdo a los flags u opciones que acompañen al comando (consultar en el manual la función de los diferentes flags). El flag –r, por ejemplo, ordena a la inversa.

El comando se invoca de la siguiente forma:

```
$ sort [opciones] [archivos]
```

Donde:

- Las palabras entre corchetes son opcionales. Cuando no se especifican parámetros, el comando *sort* ordena lo que viene de la entrada estándar.
- **opciones** son los flags que indican cómo se va a hacer el ordenamiento.
- **archivos** son los distintos archivos a ordenar.

Ejemplos de invocación del comando son:

```
$ cat agenda.txt
  Juan López      555-4321
  Antonia Pérez   555-1234
  Rodolfo Ruiz    555-3214
  Ana Cohen       555-4321
```

Se le proveerá de un material que da cuenta de la dificultad de medir la utilización de los procesadores en las arquitecturas multicore con hyperthreading.

```
$ sort agenda.txt
 Ana Cohen      555-4321
 Antonia Pérez  555-1234
 Juan López     555-4321
 Rodolfo Ruiz   555-3214
```

Tomado de: https://es.wikipedia.org/wiki/Sort_(Unix)

$ sort

$ sort file1.txt file2.txt file3.txt

$ sort –r file2.txt

Para este proyecto Ud. deberá realizar **tres implementaciones concurrentes del comando sort**, cada una de las cuales realizará el ordenamiento de varios archivos que se recibirán como entrada desde la línea de comandos, y producirá un único archivo de salida. Únicamente será válido (y opcional) para cada implementación el flag –r. A continuación se explica la sintaxis y semántica de cada uno de los procesos ejecutables o comandos que Uds. van a desarrollar.

csortp: este comando realiza el ordenamiento a través de procesos concurrentes, es decir aquellos que se crean usando la llamada al sistema *fork*.

Cómo será invocado:

$ csortp [-r] archivoinput1 [……archivoinputN] archivooutput

Donde:

-r indica que se realizará un orden inverso en todos los archivos y en consecuencia así será el orden del archivo resultante. El flag es opcional, si no se especifica, el ordenamiento se realiza según el orden alfabético.

archivoinput1….archivoinputN: son los archivos de entrada que se van a ordenar. **Debe colocarse al menos un archivo como parámetro.** De colocarse varios archivos, el número de archivos nunca será mayor a 10, es decir 1 <= N <= 10.

archivooutput: es el nombre del archivo donde se colocará el resultado.

Por ser un sort concurrente, este comando creará tantos procesos hijos como archivos **archivoinput*i*** haya recibido. Si sólo se recibe un archivo de entrada no se crearán procesos adicionales y el ordenamiento lo hará el mismo proceso **csortp**. Cada proceso hijo ordenará el archivo correspondiente y una vez que todos terminen, el padre tomará todos los archivos ordenados producidos por los hijos y generará un único archivo de salida ordenado. En las figuras 1 y 2 se puede observar la jerarquía de procesos, la

Se le proveerá de un material que da cuenta de la dificultad de medir la utilización de los procesadores en las arquitecturas multicore con hyperthreading.

distribución de archivos entre procesos y, para 3 archivos de entrada (*log1, log2 y log3*), cuál es el archivo de salida resultante. El ***algoritmo de ordenamiento lo deben implementar los estudiantes (nota: si utilizan y adaptan un algoritmo de internet debe citar la fuente).***

csortpexec: este comando realiza el ordenamiento a través de procesos concurrentes, es decir aquellos que se crean usando la llamada al sistema *fork*. No obstante, el ordenamiento en este caso *se realiza usando el comando sort del sistema operativo.*

Cómo será invocado:

$ *csortpexec [-r] archivoinput1 […..archivoinputN] archivooutput*

En este caso el significado de los argumentos de entrada es el mismo que en el caso anterior.

De igual forma que se realizó con **csortp**, este comando creará tantos procesos hijos como archivos *archivoinputi* haya recibido como argumentos de entrada. Cada proceso hijo ordenará el archivo correspondiente y, una vez que todos terminen, el padre tomará todos los archivos resultantes y producirá un único archivo de salida ordenado. La diferencia con el comando anterior es que los procesos hijos utilizarán alguna de las llamadas al sistema pertenecientes a la familia ***exec,*** de Linux para invocar al comando **sort** que provee el sistema operativo. Como en el caso anterior, una vez que todos los procesos terminen y hayan ordenado los archivos, el padre toma estos archivos resultantes y produce el archivo *archivooutput* ordenado**.**

csorth: este comando realiza el ordenamiento a través de hilos.

Cómo será invocado:

$ csorth [-r] archivoinput1 […..archivoinputN] archivooutput

El significado de los argumentos de entrada es el mismo que en los casos anteriores.

En este caso, el comando creará tantos hilos o *threads* como archivos ***archivoinputi*** haya recibido. Si se recibe un único archivo de entrada, se creará un único hilo para realizar el ordenamiento. Cada hilo creado ordenará el archivo correspondiente y una vez que todos terminen, el padre tomará la salida ordenada de cada uno de los hilos y producirá un único archivo de salida ordenado: **archivooutput.** Las tareas concurrentes en este comando serán threads de la librería POSIX. Los estudiantes pueden usar para **csorth,** el mismo algoritmo de ordenamiento que usaron en **csortp**.

Ejemplo de Ejecución

Suponga que se invoca uno de los comandos de la siguiente manera:

$ csortp log1 log2 log3 salida

Se le proveerá de un material que da cuenta de la dificultad de medir la utilización de los procesadores en las arquitecturas multicore con hyperthreading.

Las figura 1 y 2 muestran cómo es la distribución del trabajo entre los procesos y cuáles son los archivos de entrada y salida. También muestra la columna del archivo por la cual se realiza el ordenamiento: 4ta columna.

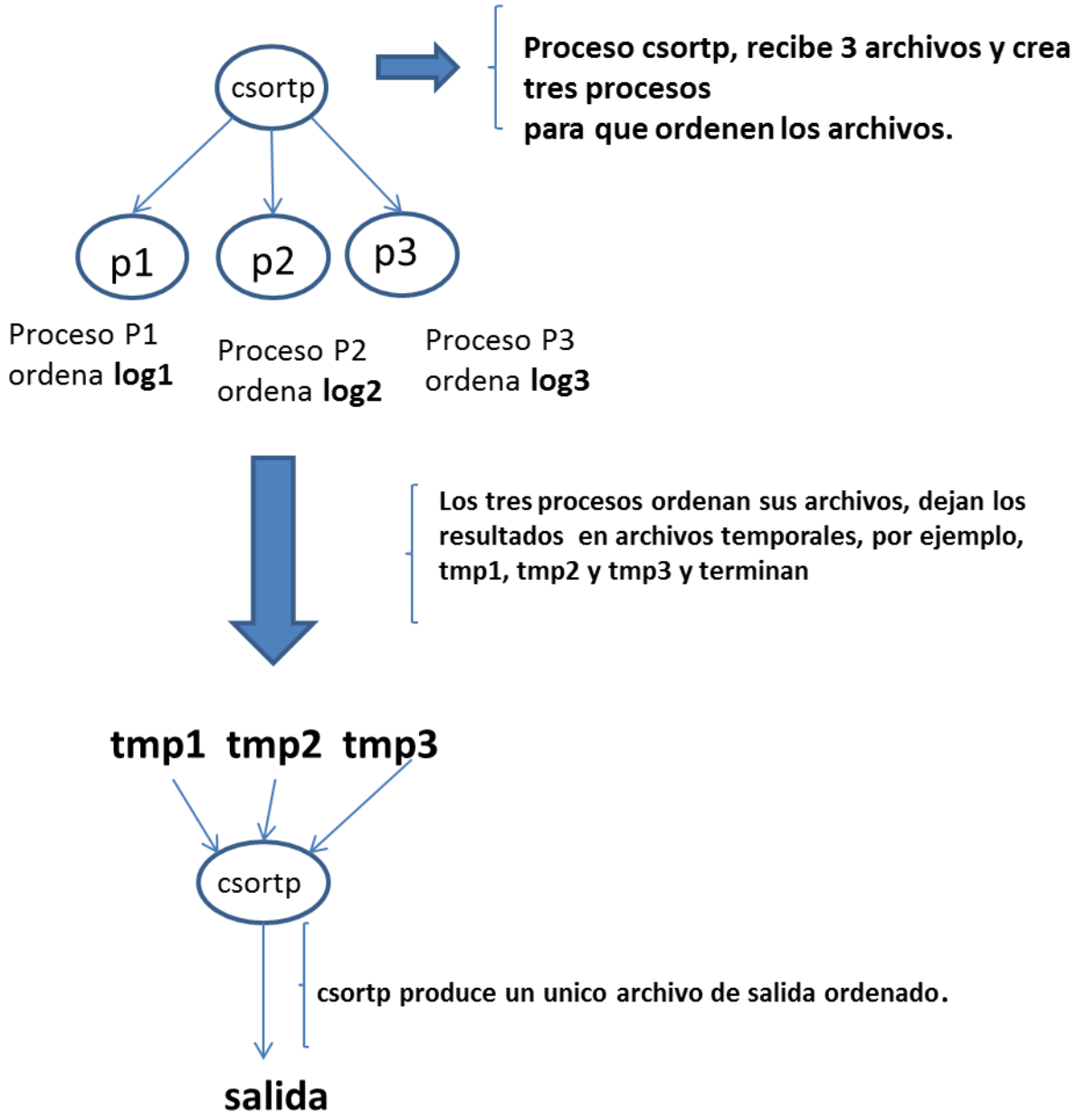

Figura I: Creación de procesos y distribución del trabajo para la ejecución del comando: **csortp log1 log2 log3 salida**

Se le proveerá de un material que da cuenta de la dificultad de medir la utilización de los procesadores en las arquitecturas multicore con hyperthreading.

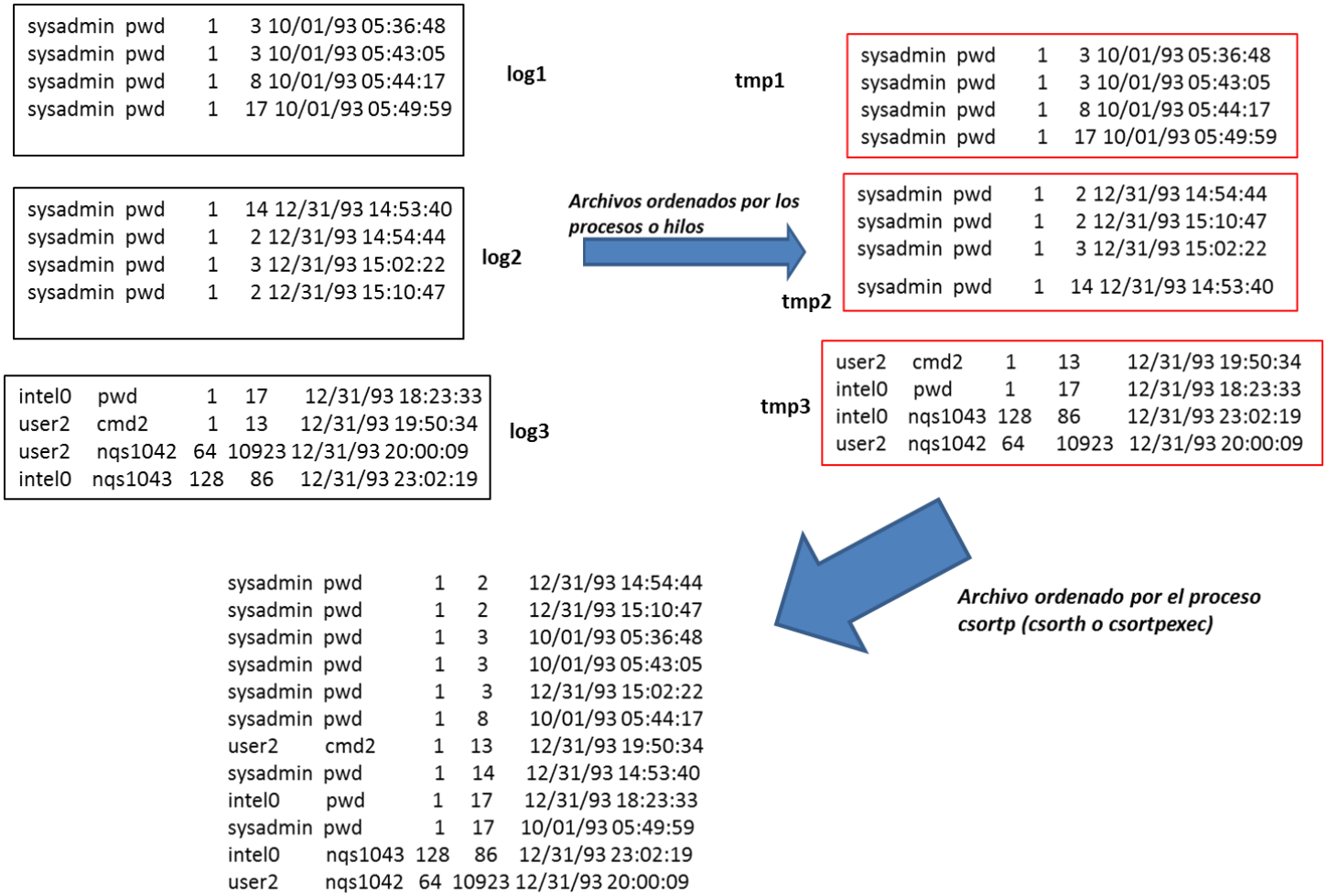

Figura II: Ejemplo de tres archivos de entrada y el archivo de salida resultante después del proceso de ordenamiento, para el comando: **csortp log1 log2 log3 salida**

Formato de los archivos de entrada y salida

Los archivos de entrada y salida tendrán el formato que se muestra en la figura II. Es decir, tendrán 6 columnas, separadas por espacios en blanco. Estos archivos forman parte de un log registrado por un sistema operativo para multiprocesadores. La primera columna es el usuario, la segunda el comando ejecutado, la tercera el número de procesadores, la cuarta el tiempo total de ejecución del comando en milisegundos, la quinta columna muestra el año de ejecución del comando y, finalmente, se muestra la hora.

Se le proporcionarán archivos de ejemplo para que pruebe sus programas y el día de la sustentación se le proveerán los archivos con los que se ejecutarán los tres programas.

Noten que el ordenamiento se realiza por defecto según el valor de la columna 4. Si los números de la columna 4 son iguales, se ordenan según la columna 5 y si éstos también son iguales se discrimina por la columna 6.

Se le proveerá de un material que da cuenta de la dificultad de medir la utilización de los procesadores en las arquitecturas multicore con hyperthreading.

Tamaño de los archivos de entrada

El tamaño máximo de cada uno de los archivos de entrada será de 100000 líneas. Todos los archivos que se reciben como argumento del programa no son necesariamente del mismo tamaño como pasa en el ejemplo de la figura 2.

Algoritmos de ordenamiento

En el caso de los comandos **csortp** y **csorth** Uds. deben implementar el algoritmo de ordenamiento de los procesos hijos, y el proceso padre debe implementar un *mergesort*. En el caso del **csortpexec** los procesos invocan al comando sort del sistema operativo con los flags adecuados. Nota: deben usar cualquiera de las llamadas **exec**. No deben usar **System** ya que es más ineficiente y se quiere que aprendan el uso del *exec*.

Detalles de la Implementación

La implementación se realizará en lenguaje C (ansi C). Para la creación de los hilos se utilizará la librería POSIX. Deben seguir en su código las recomendaciones de la Guía de Programación en C publicada en UVirtual (archivo estiloC.pdf, en la carpeta de información sobre el curso. Deben usar el compilador gcc con el flag –ansi para garantizar que el código es C estándar.

Puede usar archivos temporales para pasar archivos ordenados de los hijos a los padres. No obstante, el uso de archivos para guardar los datos temporalmente no es obligatorio y debería evitarse cuando las entidades concurrentes usan memoria compartida para comunicarse.

Evaluación del Rendimiento de los Comandos

Una vez desarrollados los tres comandos Uds. deberán medir el rendimiento de los comandos utilizando dos métricas: a) el tiempo que tarda el comando completo en ejecutarse (pueden medirlo usando el comando time) y b) la utilización del CPU durante la ejecución. Con las medidas realizadas deberá realizar gráficos y tablas que permitan comparar el rendimiento de los programas para los diferentes tamaños en los archivos de entrada y un número fijo de procesos o hilos. La medición se realizará de la siguiente forma:

1. Se le darán 9 archivos, 3 de 1000 líneas, 3 de 10000 líneas y 3 de 20000 líneas.
2. Para cada uno de estos tamaños realice mediciones de los tres comandos utilizando, en cada invocación, tres archivos de igual tamaño.
3. Todos los grupos deben realizar las mediciones en la Sala de BD, que es donde serán las clases del laboratorio.
4. Registre los tiempos totales de los comandos en cada caso y la utilización del CPU. Registre datos del computador (cpus, memoria, etc) y del SOP.

Se le proveerá de un material que da cuenta de la dificultad de medir la utilización de los procesadores en las arquitecturas multicore con hyperthreading.

5. Una vez realizadas las medidas, debe colocar los resultados en un informe que contenga no más de 5 páginas y los siguientes puntos:

    a. Identificación, título, etc.
    b. Descripción del problema planteado.
    c. Qué algoritmos de ordenamiento implementó.
    d. ¿Cómo utilizó la llamada al sistema exec, con qué parámetros o flags? En cuáles procesos: hijos, padre? (solo para el comando csortpexec)
    e. Tablas de datos: con una celda para cada una de las medidas realizadas. La tabla debe tener 9 celdas con los tiempos de ejecución (tabla X mostrada más abajo) y 9 celdas con las utilizaciones del CPU. Gráficos donde se pueda observar el comportamiento de las soluciones (la figura 3 es un ejemplo de gráfico con datos ficticios, no los compare con los resultados que Uds. deben obtener). Describa cómo realizó la medición de la utilización del CPU.
    f. Análisis de los resultados obtenidos. Los resultados son consistentes con los conceptos de la clase teórica? Cuál de las tres soluciones concurrentes es más eficiente?
    g.

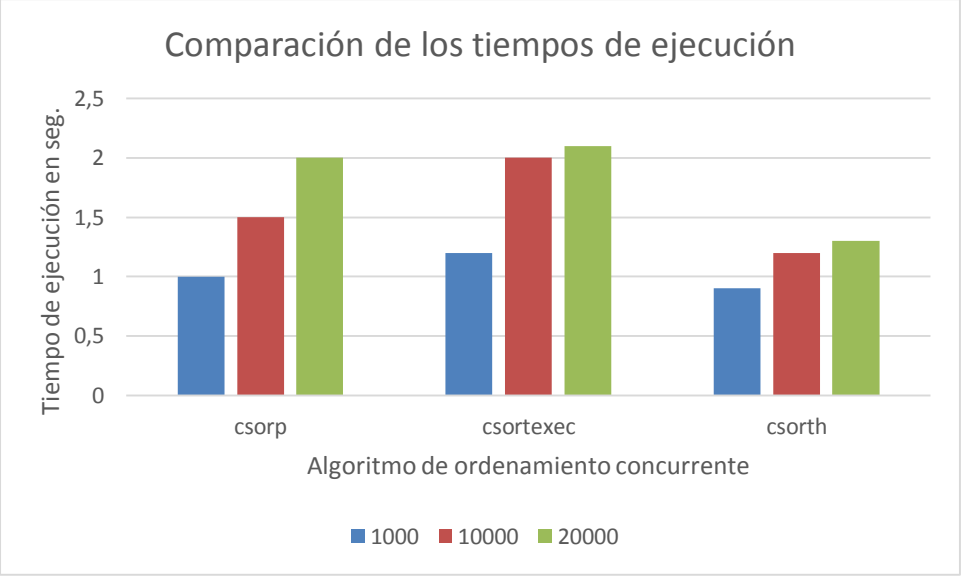

Figura 3: Ejemplo de Grafico donde se muestran los tiempos de los tres comandos para diferentes tamaños en los archivos de entrada. Todas las corridas se hicieron con N = 3

Se le proveerá de un material que da cuenta de la dificultad de medir la utilización de los procesadores en las arquitecturas multicore con hyperthreading.

Tamaño de los logs	csorp	csortexec	csorth
1000	1	1,2	0,9
10000	1,5	2	1,2
20000	2	2,1	1,3

Tabla X. tiempo de ejecución, en segundos, al ejecutar cada comando con tres logs del tamaño indicado en la primera columna.

¿Qué debe entregar y cómo debe hacerlo?

- El proyecto lo deben entregar el martes de la semana 8 (martes 17 de septiembre) antes de las 12 mm, por UVirtual. La sustentación se realizará en las horas de práctica. Si el proyecto no está en UVirtual a las 12 mm no se corregirá. Deberán colocar en UVirtual:
    - Un Makefile que permita producir los ejecutables a partir de sus archivos fuentes.
    - Los archivos fuente de su programa.
    - El informe.

      *Coloque estos archivos en un .tar o .rar que tenga el nombre de los integrantes del equipo.

- Cada archivo fuente debe estar debidamente identificado y cada función debidamente documentada. Deben validarse las llamadas al sistema y los argumentos de la entrada, pero pueden suponer que los archivos están en el formato correcto.

Observaciones Adicionales

- Revise las llamadas al sistema: *fork*, *wait*, *waitpid*, exec, f*read*, f*write*,f*open*, f*close* y aquellas relacionadas con la manipulación de hilos. Los más curiosos pueden revisar mmap().
- El proyecto lo deben realizar en grupos de 2 ó 3 estudiantes. No se admiten proyectos realizados por un único estudiante.
- Recuerde que si se sospecha de copia entre grupos, el caso será elevado al Decano de la Facultad.

Nota: Cualquier duda sobre el enunciado del proyecto debe consultarla con la profesora en forma oportuna. La comprensión del problema y su correcta implementación, según lo indica el enunciado, es parte de lo que se está evaluando.

Suerte

Prof. Mariela Curiel

Se le proveerá de un material que da cuenta de la dificultad de medir la utilización de los procesadores en las arquitecturas multicore con hyperthreading.

Se le proveerá de un material que da cuenta de la dificultad de medir la utilización de los procesadores en las arquitecturas multicore con hyperthreading.