



IIC2333 — Sistemas Operativos y Redes — 1/2019 Tarea 2

Viernes 5-Abril-2019

Fecha de Entrega: Lunes 15-Abril-2019 a las 14:00
Composición: Tarea individual

Objetivos

- Utilizar *syscalls* para construir un programa que administre el ciclo de vida de un conjunto de procesos.
- Comparar el uso de múltiples procesos respecto del uso de múltiples *threads*.
- Aprender a usar memoria compartida y señales entre procesos.

MapReduce

MapReduce es un modelo de programación usado para cómputo distribuido, ampliamente usado para procesar grandes cantidades de datos en paralelo. Este modelo consta de dos etapas que pueden verse como la aplicación de funciones a un conjunto de datos. En la etapa *map* se distribuyen las tareas a un vector y en la etapa *reduce* se agregan los resultados entregados por la etapa *map*.

Funcionalidad del programa

Su programa deberá utilizar el modelo MapReduce para procesar un texto cualquiera, de manera que se pueda obtener la siguiente información:

- (1) Cantidad de veces que se repite una palabra en el texto.
- (2) Palabras que se repiten la misma cantidad de veces, aplicando MapReduce al resultado en (1).
- (3) Cantidad de veces que se usan palabras del mismo largo, aplicando MapReduce al resultado en (1).

Multiprogramación

Deberá implementar un programa que permita resolver estas consultas utilizando el modelo MapReduce de dos maneras: (1) una versión que utilice múltiples procesos mediante las *syscalls* de manejo de procesos, y (2) una versión que utilice múltiples *threads* a través del uso de la biblioteca *pthread* (POSIX *threads*). Para cada implementación debe cumplirse que las tareas que ejecuten las operaciones de *map* se ejecuten **en paralelo**. Esto significa que debe utilizar múltiples *threads* o procesos que existan simultáneamente para ejecutar las operaciones. Por lo tanto no es correcto tener solo una operación de *map* ejecutando a la vez mediante un ciclo iterativo. Para el caso específico de múltiples procesos, se espera que éstos sean capaces de comunicarse entre sí a través de señales o memoria compartida cuando sea necesario.

Para lograr esto, se espera que tanto el proceso que corra *map* como su *reduce* correspondiente sean creados simultáneamente y, cuando *map* termine de hacer su trabajo, este le haga entrega de la información pertinente a *reduce*.

Ejecución



El programa será ejecutado por línea de comandos con las siguientes instrucciones:

```
./mapreduce <file> <output> <version> [<type>]
```

- `<file>` corresponderá a la ruta del archivo de *input*.
- `<output>` corresponderá a la ruta de un archivo CSV con las estadísticas de la simulación, que debe ser escrito por su programa.
- `<version>` corresponderá a la versión del programa: *threads* (*implementación con threads*) o *fork* (*implementación con múltiples procesos*).
- `[<type>]` es un parámetro que corresponderá a un entero entre 1 y 3 que describe lo que debe entregar el programa, según lo visto en la sección de funcionalidad del programa. El valor por defecto es 1.

Por ejemplo, algunas ejecuciones podrían ser las siguientes:

```
./mapreduce input.txt output.csv threads 3
./mapreduce input.txt output.csv fork 1
./mapreduce input.txt output.csv threads
```

Al presionar   se debe terminar la ejecución de MapReduce, esto es, que el programa termine su procesamiento. Sin embargo, debe guardar los datos calculados hasta ese momento tal como se describe en la sección *Output*. Los procesos que estén corriendo también deben dejar de hacerlo.

Archivo de entrada (*input*)

El archivo de entrada será un archivo de texto plano, en el que se encontrarán las palabras a procesar. Puede asumir que no hay signos de puntuación y que todas las palabras se encuentran separadas por al menos un espacio. No obstante lo anterior, en el archivo pueden existir saltos de línea.

Debe considerar que palabras como "Hola" y "hola" se consideran equivalentes, es decir, el programa no debe ser *case-sensitive*.

Un ejemplo de archivo de entrada es:

```
Lorem ipsum dolor sit amet consectetur adipiscing elit
Aenean sed dignissim elit nec hendrerit purus
Phasellus sit amet dictum mauris
Vivamus placerat sed justo lacinia commodo
Quisque venenatis quam eu.
```

Output

Al terminar el procesamiento, se debe escribir un archivo CSV de los datos obtenidos, los que dependen de lo que se pida procesar.

Es importante que el *output* se encuentre ordenado de menor a mayor frecuencia. En caso de empate, se debe desempatar con la segunda columna (*i.e.* orden lexicográfico de palabras). En caso de que la segunda columna tenga una lista, sus elementos deben también estar ordenados y separados por “;”, para considerar el primer elemento para el desempate.

Para ejemplificar el *output* de nuestro programa, se usará la siguiente frase:

```
amet sit dolor amet lorem sit amet dolor amet sit ipsum amet dolor
```

Para el primer caso de procesamiento, donde se pide la cantidad de veces que se repite una palabra en el texto, el *output* debe verse de la forma:

```
repeticiones,palabra
1,ipsum
1,lorem
3,dolor
3,sit
5,amet
```

Para el segundo caso, con la misma frase, el *output* debe verse de la forma:



```
repeticiones,palabras
1,[ipsum;lorem]
3,[dolor;sit]
5,[amet]
```

Finalmente, para el tercer y último caso, el *output* será:

```
cantidad,palabras
3,3
5,4
5,5
```

Es importante que se siga **rigurosamente** el formato antes descrito.

Notas

- Es importante que su programa funcione con MapReduce, es decir que haya un proceso o thread que actúe como **map** y otro como **reduce**.
- Para la comunicación entre procesos, se evaluará que se use memoria compartida con `shm_get()`, lo cual implica que, por ejemplo, hacer `fork()` de **map** para crear un reduce con la información necesaria **no será considerado válido**.
- Cuando se oprime   el programa **debe** cerrarse, junto con cualquier hijo que haya creado. El puntaje de memoria perfecta aplica en este caso.

Reporte

Uno de los objetivos de este trabajo es apreciar las diferencias de una tarea intensiva en cómputo al usar *threads* y al usar procesos. Para esto deberá efectuar una comparación entre ambas ejecuciones y reportar sus resultados en un informe breve en formato PDF con su nombre y número de alumno. Este reporte debe contener:

1. Comparación en cuanto a tiempo de ejecución, para la implementación usando procesos, y la implementación usando `pthread`s. Compare los tiempos: real (*wallclock time*), de usuario (*user time*), y tiempo de sistema (*sys time*).
2. Utilice un único archivo de entrada, de un tamaño suficientemente grande para hacer las comparaciones, y ejecútelas con distintas cantidad de procesos o *threads*. Muestre gráficamente sus resultados y explique a qué se deben. Incluya en esta explicación el ambiente utilizado (sistema operativo, versión de kernel, cantidad de memoria principal), y cualquier información que sea necesaria para comprender sus experimentos.

Si bien la correctitud del código se revisará en el servidor del curso y debe funcionar ahí, es posible efectuar los experimentos en una máquina distinta que use Linux, y que posea **al menos dos núcleos** para ejecutar. No importa en qué plataforma escriba su reporte (L^AT_EX, Word, Bloc de notas, Markdown, etc.) siempre y cuando se respete el formato de entrega solicitado (PDF).

Formalidades

A cada alumno se le asignó un nombre de usuario y una contraseña para el servidor del curso¹. Para entregar su tarea usted deberá crear una carpeta llamada T2 en el directorio principal de su carpeta personal y subir su tarea a esa carpeta. En su carpeta T2 **solo debe incluir el código fuente** necesario para compilar su tarea, además del reporte y un Makefile. Se revisará el contenido de dicha carpeta el día Lunes 15-Abril-2019 a las 14:00.

- La tarea debe ser realizada en forma individual.
- **NO debe incluir archivos binarios.** En caso contrario, tendrá un descuento de 0.5 puntos en su nota final.
- Su tarea **debe encontrarse** en la carpeta T2, compilarse utilizando el comando `make`, y generar un ejecutable llamado `life` en esa misma carpeta. Si su programa **no tiene** un Makefile, tendrá un descuento de 1 punto en su nota final.
- Es muy importante que su tarea corra dentro del servidor del curso. Si ésta **no compila o no funciona** (*segmentation fault*), obtendrán la nota mínima, teniendo como base 1 punto menos en el caso que soliciten corrección.

El no respeto de las formalidades o un código extremadamente desordenado podría originar descuentos adicionales. Se recomienda modularizar, utilizar funciones y ocupar nombres de variables explicativos. En el caso de no entregar en la carpeta especificada la tarea **no** se corregirá.

Evaluación

- **0.5 pts.** Lectura de `stdin`. Paso de argumentos. Construcción de `argc` y `argv`.
- **2 pts.** `MapReduce threads`
 - **0.5 pts.** Correcta implementación de `pthread`s.
 - **0.5 pts.** Correcta implementación de `MapReduce`.
 - **1.0 pts.** Output correcto.
- **3 pts.** `MapReduce fork`
 - **0.5 pts.** Correcta implementación de múltiples procesos.
 - **1.0 pts.** Comunicación entre procesos.
 - **0.5 pts.** Correcta implementación de `MapReduce`.
 - **1.0 pts.** Output correcto.
- **0.5 pts.** Manejo de memoria. Se obtiene este puntaje si `valgrind` reporta en su código 0 *leaks* y 0 errores de memoria en **todo caso de uso**².

Reporte: Cada pregunta del reporte sigue la siguiente distribución de puntaje:

- **2 pts.** La respuesta es correcta.
- **1 pts.** La respuesta se acerca a lo solicitado, pero posee aspectos incorrectos o no bien detallados. También aplica a las preguntas que no se sustenten en un resultado experimental, en el caso de ser solicitado.

¹`iic2333.ing.puc.cl`

²Es decir, debe reportar 0 *leaks* y 0 errores para todo *test*.

- **0 pts.** La respuesta está incorrecta o se deja en blanco.

La nota final de la evaluación es, entonces:

$$N_{T_2} = 0,7 \cdot N_S + 0,3 \cdot N_R$$

Donde N_S es la nota obtenida en la simulación y N_R la nota obtenida en el reporte.

Política de atraso

Se puede hacer entrega de la tarea con un máximo de 4 días de atraso. La fórmula a seguir es la siguiente:

$$N_{T_2}^{\text{Atraso}} = N_{T_2} - 0,75 \cdot d$$

Siendo d la cantidad de días de atraso.

Preguntas

Cualquier duda preguntar a través del [foro oficial](#).