PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

IIC2333 — Sistemas Operativos y Redes — 2/2019

Tarea 1

Viernes 16-Agosto-2019

Fecha de Entrega: Miércoles 28-Agosto-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.
- Comunicar múltiples procesos por medio del uso de señales.

`CRexecuter`

¡El DCC está en peligro! El gran **`Cruz`** necesita hacer un programa que paralelice la ejecución de múltiples programas, a modo de poder correr sus amados sistemas distribuidos que le dan conexión a todo el DCC. Sin embargo, **`Cruz`** es una persona muy ocupada, siendo profesor de 11 ramos. Desesperado, recurre a sus secuaces **`Germy`** y **`Richi`**, sin embargo ellos también se encuentran muy ocupados respondiendo *mails* y creando discos, respectivamente.
Por este motivo, se le pide ayuda a los alumnos del gran ramo **`Sistemas Operativos y Redes`** para que lo ayuden.

Funcionalidad del programa

Su programa deberá ser capaz de ejecutar múltiples programas concurrentemente mediante la creación de procesos, y controlar que dichos programas no superen un tiempo de ejecución determinado. Las tareas que debe cumplir su programa son:

(1) Recibir los argumentos entregados a su programa por medio de la línea de comandos.

(2) Leer un archivo.

(3) Ejecutar un conjunto de programas externos **de forma paralela** mediante la creación de múltiples procesos.

(4) Terminar la ejecución de todos los programas externos que excedan $\langle max \rangle$ segundos, por medio de señales.

(5) Capturar señales enviadas por el usuario, evitando el término del programa principal y haciendo que todos los programas externos en ejecución terminen.

(6) Recolectar estadísticas de los programas externos ejecutados y entregarlas en un archivo en formato `.csv`.

Ejecución

El programa principal será ejecutado por línea de comandos con las siguiente sintaxis:

```
./crexe <input> <output> <amount> [<max>] [-t]
```

Donde:

- `<input>` es la ruta de un archivo de entrada, el cual posee la lista de todos los programas externos a ejecutar.
- `<output>` es la ruta de un archivo de texto con las estadísticas de la ejecución, en formato `.csv`, que debe ser escrito por su programa.
- `<amount>` corresponde a la cantidad máxima de programas externos que pueden estar corriendo de manera concurrente.
- `<max>` es un parámetro opcional que es un entero que indica la cantidad máxima de segundos que puede demorarse en correr un programa antes de que sea terminado. Si no se entrega este parámetro, se debe considerar que los programas tienen tiempo ilimitado para ejecutar.
- `-t` es un *flag* opcional que permite ejecutar el programa utilizando *threads* en lugar de procesos. Para más detalles, ver la sección **Bonus**.

Por ejemplo, algunas ejecuciones podrían ser las siguientes:

```
./crexe input.txt output.csv 10
./crexe hola/input.txt output.csv 10 10
./crexe input.txt hola/output.csv 5 -t
./crexe input.txt output.csv 5 10 -t
```

Archivo de entrada (*input*)

El archivo de entrada será un archivo de texto plano, en el que la primera línea contendrá el número `N` de programas externos a ejecutar. Cada una de las `N` líneas que vienen a continuación contendrá: (i) el número de argumentos; (ii) el nombre del ejecutable (programa externo); y (iii) los argumentos para su ejecución. Todos los elementos estarán separados por espacios, como se muestra en el siguiente archivo de ejemplo:

```
5
2 ./hola hola.csv chao.csv
2 python3 fibonacci.py 15
1 python3 hello_world.py
1 ./crfs simdiskfilled.bin
0 ./hello
```

- Puede existir una cantidad de argumentos igual a cero.
- La cantidad de argumentos indicada en cada línea siempre será correcta.

Output

Al terminar la ejecución de todos los programas externos, se debe escribir un archivo `.csv` con los resultados finales. Este *output* debe contener `N` líneas, una por cada programa externo indicado en el archivo de *input*, y donde cada línea debe seguir **estrictamente** el siguiente formato:

```
nombre_ejecutable,tiempo_ejecucion,resultado_ejecucion
```

Donde:

- `nombre_ejecutable` es el nombre del programa externo ejecutado.

- `tiempo_ejecucion` se refiere al tiempo de ejecución del programa externo. Si fue interrumpido, es el tiempo que alcanzó a ejecutar.
- `resultado` se refiere al resultado de la ejecución del programa externo. Debe ser 1 si la ejecución es exitosa y 0 en otro caso. Entenderemos por ejecución exitosa la que **no es terminada de manera externa** (ver sección "Interrupciones") y su código de salida es cero[1].

Interrupciones

Además de la ejecución de programas de forma concurrente, el programa principal debe ser capaz de **capturar** la señal de interrupción `SIGINT`, la que se genera al oprimir Ctrl C. En el contexto de esta tarea, al recibir esta señal el programa principal **no debe** finalizar, sino que debe terminar el funcionamiento de los hijos actualmente instanciados, es decir, debe **interrumpir** todos los programas externos que estén siendo ejecutados concurrentemente en ese instante. Posterior a esto, el programa principal debe escribir las estadísticas en el archivo de salida y finalizar su ejecución. Es importante notar que, naturalmente, esto incidirá tanto en las **estadísticas de tiempo** como en la cantidad de casos **anormales** en la ejecución.

Bonus - *Threads*

Se le otorgarán 5 décimas de bonus si implementan, además, la ejecución concurrente de los programas a través de *threads*[2], la que se debe llevar a cabo si y solo si se incluye la *flag* `-t`.
Nótese que esto **sólo es un bonus**, lo cual significa que de querer implementar el programa por medio de *threads*, también se debe implementar por medio únicamente de procesos.

Formalidades

A cada alumno se le asignó un nombre de usuario y una contraseña para el servidor del curso[3]. Para entregar su tarea usted deberá crear una carpeta llamada `T1` en el directorio principal de su carpeta personal y subir su tarea a esa carpeta. En su carpeta `T1` **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 Miércoles 28-Agosto-2019 a las 14:00.

- La tarea debe ser realizada en forma individual.
- La tarea deberá ser realizada en el lenguaje de programación **C**. Cualquier tarea escrita en otro lenguaje de programación no será revisada.
- **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 `T1`, compilarse utilizando el comando `make`, y generar un ejecutable llamado `crexe` 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 0,5 puntos menos en el caso que soliciten recorrección.

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.

---

1 Recordar que, por *default*, el código de retorno 0 indica que un programa terminó sin contratiempos.
2 Notar que para ejecutar los distintos programas, es necesario usar procesos, sin embargo el bonus busca que usen *threads* para controlar el tiempo de vida de los procesos que crean.
3 `iic2333.ing.puc.cl`

Evaluación

- **0.5 pts.** Lectura de `stdin`. Paso de argumentos. Construcción de `argc` y `argv`.
- **0.5 pts.** Correcta lectura de archivos de entrada.
- **2.0 pts.** Correcta implementación de múltiples procesos paralelos.
- **1.0 pts.** Comunicación entre procesos por medio de señales, implementación del término por Ctrl C.
- **1.0 pts.** *Output* correcto.
- **1.0 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**[4].

La nota final de la evaluación es, entonces:

$$N_{T_1} = N_P + b$$

Donde N_P es la nota obtenida en el programa y b las décimas de bonus.

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:

$$N_{T_1}^{\text{Atraso}} = \text{mín}(N_{T_1}, 7{,}0 - 0{,}75 \cdot d + b)$$

Siendo d la cantidad de días de atraso. Notar que esto equivale a un descuento *soft*, es decir, cambia la nota máxima alcanzable y no se realiza un descuento directo sobre la nota obtenida.

Preguntas

Cualquier duda preguntar a través del foro oficial.

[4] Es decir, debe reportar 0 *leaks* y 0 errores para todo *test*, sin importar si este termina normalmente o por medio de una interrupción.