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

IIC2333 — Sistemas Operativos y Redes — 1/2020

Tarea 1

Viernes 27-Marzo-2020

Fecha de Entrega: Lunes 06-Abril-2020 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.

`CRVID-20`

Debido a la aparición de una extraña nueva enfermedad en el DCC llamada CRVID-20 (**C**ristian **R**uz **VI**rus **20**20), la cual tiene como síntoma principal la inhabilidad de programar en cualquier lenguaje que no sea **C**, el departamento le ha pedido al profesor **Cristian Ruz**, el descubridor de la enfermedad, que simule las posibles configuraciones que podría tomar el virus por medio del **Juego de la Vida**.

Sin embargo, existe una dificultad importante. Dado que pueden darse infinitas combinaciones distintas, debido tanto a las condiciones iniciales como a los parámetros de simulación, es necesario que el programa ejecute múltiples simulaciones de manera **paralela**.

Ya que en el presente el profesor de encuentra en cuarentena, este le pide ayuda a los alumnos del gran ramo **`Sistemas Operativos y Redes`** para que lo ayuden a crear su programa.

Juego de la vida

El **juego de la vida** es un juego diseñado por John Conway en el año 1970 [1], que consiste en un **autómata celular** donde se simula el nacimiento y muerte de pequeñas "células" en tiempos discretos. Para la simulación se tendrá un espacio acotado de dimensiones $\texttt{D} \times \texttt{D}$ dividido en cuadrados, que llamaremos "tablero". Los cuadrados del tablero pueden estar "vacíos" cuando no poseen una célula en ellos, o "llenos" cuando sí poseen una. Las reglas para determinar si en una casilla nace, muere o continúa viviendo una célula son las siguientes:

- Una célula nace en un espacio vacío si y solo si a su alrededor hay `A` células vivas.
- Una célula se mantiene con vida solo si a su alrededor hay entre `B` y `C` células vivas.

Considere que `A`, `B` y `C` son variables entre 1 y 8 (inclusive), con $\texttt{B} \leq \texttt{C}$, y $\texttt{D} \geq 5$.

Simulación

Debido a la naturaleza del juego, existen situaciones en las que podemos llegar a un *loop* infinito, y queremos evitar que nuestro programa corra por siempre. Describiremos tres condiciones para terminar nuestra simulación. Basta con que alguna de éstas se cumpla para finalizar. Estas condiciones son:

[1] En este link puedes encontrar más información

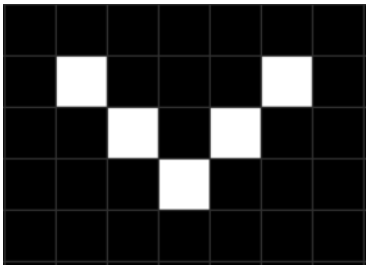

Figura 1: El juego en un estado inicial.

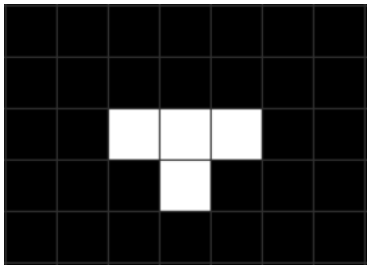

Figura 2: El mismo juego para `A` = 3, `B` = 2 y `C` = 3.

- No quedan células vivas dentro del tablero.
- Se alcanzó un tiempo máximo de simulación `t`.
- Por medio de una señal externa, se le pidió a la simulación que terminase.

Estructura de Procesos

Su programa (o sus programas, dependiendo de como modelen el problema) deberá poder cumplir con las siguientes funciones:

- Modelar un tablero del Juego de la vida.
- Crear procesos hijos.

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:

- En el caso de un proceso que simule al juego de la vida, este deberá:
  (1) Leer la estructura del tablero desde un archivo de texto.
  (2) Simular el juego de la vida hasta el término del juego.
  (3) Escribir en un archivo CSV las estadísticas del juego terminado.
  (4) Capturar correctamente las señales enviadas por su padre.
- En el caso de un proceso generador, este deberá:
  (1) Leer los procesos que debe crear desde un archivo de texto.
  (2) Crear los subprocesos correspondientes.
  (3) Controlar los tiempos de ejecución de sus procesos hijos.
  (4) Capturar y, de ser necesario, propagar correctamente las señales enviadas por su padre.
  (5) Escribir en un archivo CSV las estadísticas de todos sus procesos hijos.
  (6) Enviar las señales correspondientes.

Ejecución

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

```
./crvid <input> <line>
```

Donde:

- `<input>` es la ruta de un archivo de entrada, el cual posee la lista de todos los programas externos a ejecutar.
- `<line>` es línea del archivo de entrada donde debe empezar a leer nuestro programa principal.

Un ejemplo de ejecución podría ser el siguiente:

```
./crvid input.txt 0
```

Archivo de entrada (*input*)

El archivo de entrada será un archivo de texto plano, en el cual cada línea representará un proceso. Los dos tipos de línea son:

- Línea de proceso generador:

  ```
  tipo tiempo n linea_1 ... linea_n
  ```

  Todos los valores son enteros. En específico `tipo` representa que tipo de proceso es (los procesos generadores son `0`), `tiempo` representa el tiempo máximo en segundos que el proceso puede correr tras el cual debe detener a sus procesos hijos enviandoles la señal `SIGINT`, `n` representa la cantidad de sub-procesos que se van a crear y finalmente, los siguientes `n` números representan las líneas del archivo donde se puede encontrar cada sub-proceso a ejecutar.

- Línea de proceso simulador:

  ```
  tipo iteraciones A B C D tablero
  ```

  Todos los valores son enteros. En específico `tipo` representa que tipo de proceso es (los procesos simuladores son `1`), `iteraciones` representa la cantidad máxima de iteraciones antes de que se pare el proceso, `A`, `B`, `C` y `D` son los parámetros del juego de la vida y finalmente, `tablero` representa la línea del archivo `tableros.txt` donde se puede encontrar el tablero a utilizar.

Un ejemplo de archivo de entrada es el siguiente:

```
0 5 2 1 4
1 7 5 8 10 13 1
0 2 1 3
1 1 3 7 9 18 2
0 5 1 2
```

Cosas a tener en consideración

- Toda linea del archivo es un punto de partida válido para el programa.
- Para todos los *input* la línea 0 estará conectado a todos los procesos.
- No habrán dos procesos generadores que compartan un proceso hijo.

Tableros

Junto con el archivo de input, existirá el archivo `tableros.txt` donde se encuentran las posibles configuraciones iniciales de células. Cada línea describirá el posicionamiento de las células iniciales en cada tablero, con las posiciones `x` e `y` comenzando en 0 desde la esquina **superior izquierda**.
Cada linea del archivo tendrá el siguiente formato:

```
n x_1 y_1 x_2 y_2 ... x_n y_n
```

Donde `n` indica la cantidad de células iniciales en el tablero. Todas las posiciones `x`, `y` serán posiciones válidas y únicas dentro del tablero. Un ejemplo de archivo de *tableros.txt* sería lo siguiente: [2]

```
3 1 0 2 0 4 3
5 5 5 1 1 2 2 3 3 4 4
1 1 0
2 2 0 3 4
```

Output

Al momento de terminar un proceso deberá escribir información en un archivo de nombre `<line>.csv`, siendo `<line>` la línea que representa al proceso. El formato del output serpá distinto entre procesos generadores y procesos simuladores, en especifico, el formato será el siguiente:

- Formato proceso simulador:

  Al terminar, un proceso simulador deberá escribir una sola linea en su csv siguiendo el siguiente formato:

  ```
  cantidad_células, iteraciones, razón_término
  ```

  Donde *cantidad_células* La **razón** por la cual la simulación termina, puede ser

  1. `NOCELLS`, si el tablero termina por falta de células.
  2. `NOTIME`, si el tablero termina debido al tiempo máximo de simulación.
  3. `SIGNAL`, si el tablero es terminado manualmente.

- Formato proceso generador:

  Al terminar, un proceso generador deberá unir todos los csv creados por sus procesos hijos en uno.

Interrupciones

Todos los procesos creados deben ser capaces 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 proceso **no debe** finalizar, sino que debe propagar esta señal y esperar al término de todos hijos actualmente instanciados. Posterior a esto, el proceso debe escribir las estadísticas correspondientes en su archivo de salida y finalizar su ejecución. Es importante notar que cuando un proceso generador cumple con su tiempo máximo de ejecución, debe enviar la señal `SIGINT` a sus hijos.

[2] Notarás que no se especifica el tamaño del tablero. Este se puede encontrar en el archivo de entrada, es el valor `D`.

Modelación

En esta tarea se pide que creen múltiples procesos. Para lograr esto les recomendamos tres posibles modelaciones:

- Crear un solo programa que dentro tenga definido los dos tipos de procesos [3] y haga uso de `fork` para crear los nuevos procesos.
- Crear un solo programa que dentro tenga definido los dos tipos de procesos y haga uso de `exec` para crear los nuevos procesos.
- Crear tres programas, uno por cada tipo de proceso y uno que llame al proceso inicial. Luego, se puede utilizar `exec` para crear los nuevos procesos.

Estas solo son propuestas, no son las únicas opciones, pero los invitamos a seguir alguna de estas si les parece más cómodo.

Formalidades

A cada alumno se le asignó un nombre de usuario y una contraseña para el servidor del curso[4]. 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 y un `Makefile`. Se revisará el contenido de dicha carpeta el día Lunes 06-Abril-2020 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 `crvid` 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, los cuales quedarán a discreción del ayudante corrector. 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`.
- **0.5 pts.** Correcta lectura de archivos de entrada.
- **1.0 pts.** Correcta implementación de proceso generador.
- **1.0 pts.** Correcta implementación de proceso simulador.
- **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**[5].

[3] Como funciones distintas, o en archivos distintos, por ejemplo.
[4] `iic2333.ing.puc.cl`
[5] Es decir, debe reportar 0 *leaks* y 0 errores para todo *test*, sin importar si este termina normalmente o por medio de una interrupción.

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:

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

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. El uso de días de atraso no implica días extras para alguna tarea futura, por lo que deben usarse bajo su propio riesgo.

Preguntas

Cualquier duda preguntar a través del foro oficial.