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

IIC2333 — Sistemas Operativos y Redes — 2/2018

Tarea 2

Martes 4-Septiembre-2018

Fecha de Entrega: Martes 11-Septiembre-2018 a las 23:59
Composición: Tarea individual

Objetivo

Desarrollar un programa utilizando múltiples procesos y controlando su ciclo de vida mediante *syscalls*.

Juego de la vida

El **juego de la vida** es un juego diseñado por John Conway en el año 1970, que consiste en un **autómata celular** donde se simula el nacimiento y muerte de pequeñas "células" en tiempos discretos[1]. Para la simulación se tendrá un espacio acotado de dimensiones `D`×`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 $\texttt{D} \geq 5$.

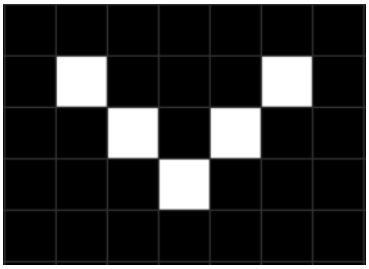

Figura 1: El juego en un estado inicial.

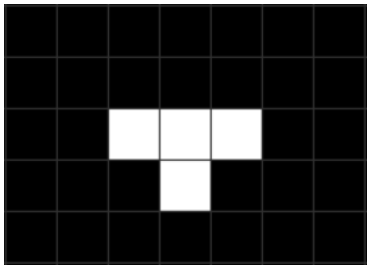

Figura 2: El mismo juego para `A` = 3, `B` = 2 y `C` = 3.

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:

- No quedan células vivas dentro del tablero.
- Se alcanzó un tiempo máximo de simulación `t`.
- La simulación ha entrado en un *loop*. Definiremos que una simulación ha entrado en un *loop* cuando vuelve a estar en uno de los estados que tenía hace 4 o menos tiempos.

[1] *Disclaimer*: Todos los ayudantes han cursado o están cursando matemáticas discretas.

Input

El archivo de texto que será entregado como *input* tendrá en la primera línea los parámetros necesarios para realizar una simulación. Cada línea siguiente describirá el posicionamiento de las células iniciales en cada tablero, con las posiciones `x` e `y` comenzando en 0 desde la esquina **inferior izquierda**. El archivo tendrá el siguiente formato:

```
<número de tableros> <A> <B> <C> <D>
<nombre tablero 1> <cel_1> <x1> <y1> <x2> <y2> ... <xcel_1> <ycel_1>
...
<nombre tablero n> <cel_n> <x1> <y1> <x2> <y2> ... <xcel_n> <ycel_n>
```

Donde `cel_i` indica la cantidad de células iniciales en el tablero `i`. Todas las posiciones `x`, `y` serán posiciones válidas y únicas dentro del tablero. El siguiente es un ejemplo de archivo de *input*:

```
3 3 2 3 10
t1 5 3 5 4 4 5 3 6 4 7 5
t2 3 2 3 3 3 4 3
t3 4 1 1 1 2 2 1 2 2
```

Output

Al terminar cada tablero, se debe imprimir el nombre de éste y sus condiciones de término, es decir, si la simulación termina por no quedar células vivas, por alcanzar el tiempo máximo, por entrar en un *loop* o por término inesperado del proceso. Finalmente, debe indicar el último tiempo de simulación y cantidad de células vivas en ese último tiempo. El siguiente es un ejemplo de *output*:

```
t1 Término por loop. Tiempo simulación: 6. 3 Células
t2 Término por falta de células. Tiempo simulación: 120. 0 Células
t3 Término por loop. Tiempo simulación: 20. 12 Células
```

Una vez que el programa haya terminado, su programa deberá escribir un archivo `CSV` con los siguientes datos por tablero:

- Nombre del tablero.
- Tiempo final de simulación.
- Cantidad de células que quedan cuando el tablero termina.
- La **razón** por la cual la simulación termina, la cual puede ser
    1. `LOOP`, si el tablero termina por encontrarse en un *loop*.
    2. `NOCELLS`, si el tablero termina por falta de células.
    3. `NOTIME`, si el tablero termina debido al tiempo máximo de simulación.
    4. `SIGNAL`, si el tablero es terminado manualmente.

Es importante que siga **rigurosamente** el siguiente formato:

```
nombre_tablero_i,tiempo_final_i,cantidad_celulas_i,razon_termino_i
nombre_tablero_j,tiempo_final_j,cantidad_celulas_j,razon_termino_j
...
```

Ejecución

El juego será ejecutado por línea de comandos con la siguiente instrucción:

```
./life <file> <t>
```

Donde `life` es el programa compilado, `<file>` es la ruta del archivo con la descripción de los tableros y `t` el tiempo máximo de ejecución por tablero, compartiendo todos este mismo límite.
Al presionar Ctrl C se debe terminar la ejecución de `life`, esto es, que el programa termine las simulaciones. Sin embargo, debe imprimir los datos de cada tablero hasta ese momento tal como se describe en la sección *output*. Los procesos que estén corriendo también deben dejar de hacerlo.

Tenga en cuenta

1. Cada tablero debe ser ejecutado por un proceso diferente (no por *threads* de un mismo proceso).
2. El programa principal no podrá terminar hasta que **todos** los tableros terminen su ejecución.
3. Deberá usar las *syscalls* de manejo de procesos `fork`, `exec`, `wait`, `kill` (no *threads*).
4. Deberá construir adecuadamente `argc` y `argv` para pasarlos a los comandos que deberá ejecutar.

Reporte

Además de su programa, deberá incluir un reporte en formato `PDF` con su nombre y número de alumno. En este reporte, debe responder las siguientes preguntas:

1. ¿Qué ventajas y desventajas podría tener la implementación de esta tarea con `threads` en vez de el uso de procesos separados?
2. Ejecute su programa de tal manera que haya 1, 2, 4, 8 y 16 simulaciones ejecutando simultáneamente. Para cada ejecución mida su tiempo de ejecución usando `time`[2]. Explique el comportamiento de los tiempos `real`, `user`, y `sys`.

No importa en qué plataforma escriba su reporte (LaTeX, 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 curso[3]. 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 Martes 11-Septiembre-2018 a las 23:59.

- 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 recorrección.

[2] Para esto puede ejecutar `time ./life <file><t>`
[3] `iic2333.ing.puc.cl`

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

- **0.25 pts.** Formalidades. Esto incluye cumplir las normas de la sección formalidades.
- **5.75 pts.** `life`
  - **0.75 pts.** Lectura de `stdin`. Paso de argumentos. Construcción de `argc` y `argv`.
  - **1.0 pts.** Correcta implementación del **juego de la vida**.
  - **1.0 pts.** Estadísticas de ejecución.
  - **3.0 pts.** Implementación de procesos.

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.
- **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 2 días de atraso. La fórmula a seguir es la siguiente:

$$N_{T_2}^{\text{Atraso}} = N_{T_2} - 1{,}5 \cdot d$$

Siendo d la cantidad de días de atraso.

Bonus 1(+0.5 pts): manejo de memoria perfecto

Se aplicará este bonus si *valgrind* reporta en su código 0 *leaks* y 0 errores de memoria, considerando que los programas funcionen correctamente. El bonus a su nota se aplica solo si la nota correspondiente es $\geq 3{,}95$.

Bonus 2(+1.0 pts): visualización gráfica del juego

Se aplicará este bonus si su programa, además de generar el archivo de estadísticas, imprime en consola el estado inicial y final de cada tablero en la consola. Para esto, pueden ayudarse con los íconos de Unicode UTF-8 [4]: ❑ (hexa: 25A0) y ■ (hexa: 25A1). El bonus a su nota se aplica solo si genera el *output* solicitado con los resultados de la simulación y si la nota correspondiente es $\geq 3{,}95$.

Preguntas

Cualquier duda preguntar a través del foro.

[4] https://www.w3schools.com/charsets/ref_utf_geometric.asp