Universidad Nacional de Rosario

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Sistemas Operativos 1

Trabajo Práctico N°1

Conway Game Of Life

Grupo

Cavagna, Lucas Nahuel

Demagistris, Santiago Ignacio

Abril 2021

Índice general

1 Congway Game Of Life

1.1. Enunciado

El trabajo consiste en la implementación de un simulador del Juego de la Vida de Conway. En particular se pide que la simulación se realice implementando un programa concurrente, tratando de optimizar la disponibilidad del hardware subyacente. Pero principalmente:

1. **Evitando estados inconsistentes del juego. La ejecución secuencial o paralela debería otorgar el mismo tablero final.**
2. **Evitando la aparición de deadlocks.**

Para esto pueden utilizar todos los recursos vistos en la primer practica. Implementados por ustedes, o los disponibles en la librería de **POSIX THREADS** (y semáforos).

Juego de la Vida

El Juego de la Vida fue planteado por el matemático John Horton Congway en 1970. Es un juego sin jugadores que planteado como un autómata celular. Esto significa que la única interacción está dada al comienzo del juego, planteando un tablero inicial, y que luego el sistema evolucionará siguiendo las reglas establecidas.

Reglas

El juego presenta un tablero bidimensional infinito compuesto por células. Las células pueden estar vivas o muertas. Al ser un tablero bidimensional e infinito, cada célula tiene 8 células vecinas. En cada momento, las células pasan a estar vivas o muertas siguiendo las reglas:

1. **Toda célula viva con 2 o 3 vecinos vivos sobrevive.**
2. **Toda célula muerta con exactamente con 3 vecinos vivo revive.**
3. **El resto de las células vivas mueren en la siguiente generación, como a su vez el resto de las células muertas se mantienen muertas.**

El **patrón inicial** del tablero se le suele llamar **semilla**. La primer generación es el resultado de aplicar las 3 reglas antes descriptas a todas las células de la semilla, las transiciones se dan de forma simultanea. Las reglas se siguen aplicando de la misma manera para obtener futuras generaciones.

2 Ensayo

2.1. Cambios de Game.h y Board.h

Decidimos utilizar un puntero a la estructura board/game (por eso está la función sarasa_create()) ya que nos sentimos más cómodos haciéndonos responsables de asginar y liberar memoria. Las operaciones eliminadas fueron reemplazadas por otras que creemos que cumplen la misma función. La decisión se basó en como fuimos encontrándonos con las dificultades/requerimientos. Si bien tomamos como punto de partida las funciones iniciales (init,boar_load, etc) una vez que terminamos con la lectura fuimos definiendo funciones a medida que las necesitabamos, por lo cual no nos guiamos al 100 % con el esqueleto.

Board.h

Operaciones nuevas

board_interchange
board_write
board_getCantFilaspar
board_getCantColumnas
board_create
board_cells_create
get_state
board_actual_set
board_proxGen_set

Operaciones actualizadas

board_show
board_load
board_init

Operaciones eliminadas

board_init_def
board_get_round

Game.h

Operaciones nuevas

game_writeBoard
game_destroy
game_show
game_load
game_init
game_vecino_lateral_d
game_vecino_lateral_i
game_vecino_inferior_d
game_vecino_inferior_i
game_vecino_inferior
game_vecino_superior_d
game_vecino_superior_i
game_vecino_superior
game_getCantColumnas
game_getCantFilas
game_set_value

Operaciones actualizadas

congwayGoL

Operaciones eliminadas

game_writeBoard
loadGame

2.2. Implementación de Game.c y Board.c

Board.c

La estructura _board se implemento de esta manera:

```
struct _board{
int** tableroActual;
int** tableroProxGen
int cantColumnas;
int cantFilas;
};
```

Decidimos implementar la estructura _board de la siguiente manera:

- Un tablero principal "tableroActual" que es un array bidimencional de enteros que contiene el patrón actual (estado de las células).
- Un tablero secundario "tableroProxGen" que es un array bidimencional utilizado para almacenar la información de la siguiente generación de acuerdo al patrón actual.
- La cantidad de filas y columnas que tienen ambos tableros para tener sus dimenciones en todo momento.

Decidimos utilizar arreglos de enteros ya que nos pareció más práctico a la hora de operar sobre ellos que tener que hacerlo sobre arreglos de chars.

Game.c[1]

La estructura _game se implemento de esta manera:

```
struct _game{
board_t* board;
int ciclos;
};
```

Decidimos implementar la estructura _board de la siguiente manera:

- Un puntero a un board_t llamado "board" que contendría toda la información pertinente del juego y serviría para guardar las generaciones nuevas del tablero.
- La cantidad de ciclos o generaciones çiclos" que se llevaran a cabo a partir de la información de "board

[1]Para la implemetnacion de Game.c se añadio la libreria Barreras.h

2.3. Toma de decisiones

1. Programamos desde la lectura del tablero hacia adelante. No nos preocupamos por el problema de la sincronización hasta que solucionamos todo lo subyacente. Fuimos avanzando en ese sentido y probando los módulos hasta que tuviesen el comportamiento deseado.

2. Utilizar 2 tableros, uno actual y uno próximo, ya que nos pareció más sencillo de trabajar que crear un tablero de dimensión 3 en el cual cada célula almancenaba el valor actual y el próximo. De esta forma cada hilo que trabaja sobre una célula en el tablero actual almacena el resultado de aplicar las reglas del juego en el tablero de próxima generación, así una vez que todos terminen de analizar cada célula tendríamos la próxima generación ya establecida. Una vez obtenida la próxima generación basta con intercambiar los punteros y así simplemente sobreescribir proxGen para el próximo ciclo.

3. El trabajo en concurrencia lo definimos como ir otorgando células por los índices desde el (0,0) hasta el (cantFilas-1,cantColumnas-1). Las ideas iniciales oscilaban entre un hilo por fila y un hilo por célula, la primer opción no era tan eficiente de acuerdo a la cantidad de filas y columnas como la que implementamos y la segunda opción no era viable de acuerdo a la pc en cuestión y la cantidad de células.

4. Utilizamos barreras y locks explícitos. De esta forma se nos hizo muy sencillo pensar al progama como estados en una cadena de producción.

5. Pensamos a game y a board como objetos y así toda interacción de game con el tablero debería pasar por una operación definida en board. De esta forma evitamos ingresar a información que no está definida implícitamente en el objeto en cuestión ya que el objeto game tiene un campo que es un objeto board. Por esto hay funciones que simplemente son una operación del game que llama a otra operación pero del board y esta última es la que realmente resuelve el requirimiento.

6. La lectura e inicialización decidimos hacerla .ªl simultáneo", en el sentido de que a medida que leemos los datos vamos completando donde corresponda. Esto nos llevo a tener que leer la primer línea 2 veces, una para obtener las dimensiones del tablero para crearlo y luego la inicialización propia del tablero actual se da en el objeto board.

7. Mostramos por pantalla el tablero actual con el número de generación correspondiente. Para avanzar a la siguiente generación se debe presionar enter

8. Definimos que el ingreso del patrón inicial o semilla deberá tener un salto de linea alfinal del archivo. De otra forma se tomará como archivo erróneo.

2.4. Concurrencia

Las regiones críticas son:

indiceFila

indiceColumna

actualizando

terminoCiclo

Si lo observamos como procesos, los procesos críticos en esta implementación son:

1. **Asignar célula (indiceFila, indiceColumna) a un dios**
2. **Intercambiar tableros**
3. **Reinicializar variables de la región crítica**

La idea de la concurrencia se basó en otorgar células a los dioses, para esto definimos variables globales (indiceFila, indiceColumna) las cuales irían señalando a cada célula del tablero. Todos los dioses van a buscar una célula pero aquí pusimos un lock, de esta forma sólo un dios puede obtener la célula que se está señalando actualmente por los índices. Esto se encuentra en un bucle que no termina hasta que todas las células hayan sido otorgadas a dioses (la cantidad de dioses está determinada por la cantidad de hilos que puede otorgar la pc en cuestión). Una vez que se otorgan todos las células y se ejecutan todos los juicios divinos sobre las mismas, una bandera, que se llama terminoCiclo, cambia su valor otorgando el paso al siguiente estado del proceso. Este estado es intercambiar los tableros y reinicializar las variables globales (con la bandera actualizando), utilizamos una barrera para asegurar que todos los dioses ejecutaron su juicio divino antes de intentar intercambiar los tableros. Luego por medio de un lock, el dios ganador de la compentencia será el que intercambie y reinicialice las variables. En este punto se puede considerar que ya el proceso estaría pipi cucú pero no es así ya que si dejásemos que los demas dioses continúen con la siguiente generación podríamos llegar a un estado de inconsistencia en el cual se modifique el tablero antes de intercambiarse o se intercambien varias veces. Para evitar esto utilizamos una barrera que en definitiva espera a que todos los dioses lleguen a ese punto para poder iniciar el siguiente ciclo.