TP Juego de la Vida de Conway

Cassinerio Marcos - Cerruti Lautaro

Abril 19, 2021*

*Updated April 21, 2021

1 Introducción

El objetivo de este trabajo fue implementar un simulador del Juego de la Vida de Conway que funcionara de forma concurrente.

2 Compilado y ejecución

Para compilar el proyecto abrimos una terminal, y una vez ubicados en el directorio del proyecto, ejecutamos el comando `make`. Esto nos generará el ejecutable del simulador.

El mismo lo corremos con:

```
./simulador FilePath/Nombre.game
```

3 Organizacion de los archivos

El programa se divide en 4 partes: Board, Barrier, Game y Simulador
Por un lado tenemos la implementación y declaración de Board en los archivos `board.c` y `board.h` respectivamente.
Por otro lado tenemos Barrier hecho de la misma manera, en los archivos `barrier.c` y `barrier.h`.
Luego tenemos al Game que hace uso de las dependecias board y barrier. Su implementación y declaración se encuentra en los archivos `game.c` y `game.h`.
Finalmente tenemos en el archivo `simulador.c` que hace uso de Game para poder correr el juego.

4 Implementaciones y estructuras

4.1 Board

La implementación de Board esta basado en un array bidimensional de chars, definido de la siguiente manera:

```
struct _board {
  char** cell;
  unsigned int col;
  unsigned int row;
};

typedef struct _board board_t;
```

En su cabecera declaramos las funciones:

- `board_init`
- `board_get`
- `board_get_round`
- `board_set`
- `board_load_row`
- `board_show`
- `board_destroy`

La función `board_get` nunca es utilizada, pero para que la implentación sea general se vio la necesidad de hacerla.
Todas las implementaciones se encuentran en `board.c`.

4.2 Barrier

La declaración de la barrier es la siguiente:

```
struct cond_barrier {
  unsigned int count;
  unsigned int waiting;
  pthread_mutex_t mutex;
  pthread_cond_t cond;
};

typedef struct cond_barrier barrier_t;
```

En su cabecera declaramos las siguientes funciones:

- `barrier_init`
- `barrier_wait`
- `barrier_destroy`

Sus implentaciones se encuentran en `barrier.c`, junto con la implementacion de la funcion:

- `digits_of_int`

4.3 Game

El Game se encuentra definido de la siguiente manera:

```
struct _game {
  unsigned int cycles;
  board_t *board;
};

typedef struct _game game_t;
```

En su archivo cabecera se encuentran declaradas las siguientes funciones:

`loadGame`
`writeBoard`
`conwayGoL`
`destroyGame`

Sus implementaciones se encuentran en el archivo `game.c`, junto con las implementaciones de las funciones:

`distribute`
`getNextState`
`threadFoo`

4.4 Simulador

En el simulador se encuentra el main del programa, este se encarga de ejecutar las funciones de Game.

5 Desarrollo y complicaciones

Comenzamos implementando el board mientras pensabamos ideas de como implementar la concurrencia. Luego de implementarlo con barriers para sincronizar todos los threads al finalizar la escritura, comenzamos a notar algunos problemas y errores.
Uno de estos errores fue que en el manual de `pthread_cond_wait` dice que hace un unlock del mutex que se le pasa por parámetro, sin embargo debuggueando notamos que no lo estaba haciendo, provocando que todos los otros threads esperaran a que este lo soltara pero el thread que lo había lockeado estaba dormido, lo que resultaba en deadlock.
En barrier se nos presento otra complicación que fue por un error nuestro, cuando era el último thread en llegar, hacía el unlock del mutex antes de poner el contador de waiting en 0. Esto provocaba que los threads que estaban esperando el mutex se fueran a dormir antes de haber reiniciado el contador, y luego el thread lo reiniciaba a 0 cuando en realidad ya había threads durmiendo, provocando que en un momento todos se fueran a dormir y el programa quedara en deadlock.
Otra complicación que se nos presentó fue que al pasar los argumentos a cada thread estabamos usando una referencia a la misma variable de estructura de argumentos con una de sus variables cambiada en cada iteración. Esto causo que en diferentes threads se tuviera el mismo valor en dicha variable, calculando erroneamente el tablero.

Una de las decisiones de desarrollo que consideramos importante mencionar es la de implementar una funcion `distribute` para optimizar las divisiones del tablero entre los threads. Ya que en un principio pensamos en hacer divisiones solamente por rows, pero nos dimos cuenta que en un caso como cuando tenemos un tablero de 3 filas por 20 columnas y 4 unidades de procesamiento, no estariamos aprovechando una de esas unidades. Entonces implementamos la funciónantes mencionada que basandose en la cantidad de filas y columnas, dividiera el tablero por cual fuera la mayor de esas cantidades.

6 Bibliografia

https://es.wikipedia.org/wiki/Juego_de_la_vida
https://pubs.opengroup.org/onlinepubs/7908799/xsh/pthread_cond_wait.html