INFORME TRABAJO PRÁCTICO 1

Joaquin Arroyo – Ignacio Cainelli

El juego de la vida de Conway se lleva a cabo sobre un tablero donde cada casilla(box) simboliza una célula. Dicha célula puede estar viva o muerta según sus vecinos y las reglas que se apliquen para dicho juego. Es decir, por cada turno estas células pueden cambiar su estado.

En la implementación dada, la estructura *box_t* contiene a los datos *int state* y *char val,* es decir, el state tendrá el estado de la célula y el val tendrá el valor que se leerá y se mostrará en salida.

Ahora, teniendo un tablero con *r* filas y *c* columnas, si iteramos sobre cada casilla de este tablero, se podrán seguir los siguientes pasos para llevar a cabo el juego:

```
for(i ; i < r ; i++){
     for(j ; j < c ; j++){
          contar_vecinos(tablero[i][j]);
          aplicar_reglas(tablero[i][j]);
          cambiar_estado(tablero[i][j]);
     }
}
actualizar_valores(tablero);
```

La función `contar_vecinos` cuenta la cantidad de células vivas adyacentes (se fija en su valor y no en su estado), la función `aplicar_reglas` aplica las reglas del juego de la vida según la cantidad de células vecinas vivas adyacentes y `cambiar_estado` cambia el estado (no el valor) de la célula si esta lo requiere. Al final de la iteración (final del turno) se actualizan los valores de todas casillas del tablero.

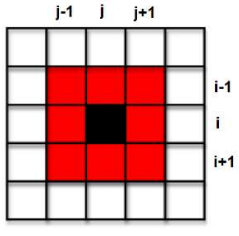
Este sería nuestro programa sin utilizar concurrencia, ahora, a la hora de plantear la concurrencia, necesitamos encontrar cual iba a ser nuestro recurso compartido y de qué manera lo íbamos a distribuir entre los distintos procesos.

Recurso compartido: el tablero y los ciclos

¿Cómo dividimos el tablero?

Para empezar, cada célula depende de sus células vecinas, es decir, el valor de una célula no debe actualizarse hasta que se actualicen los estados de las células vecinas.

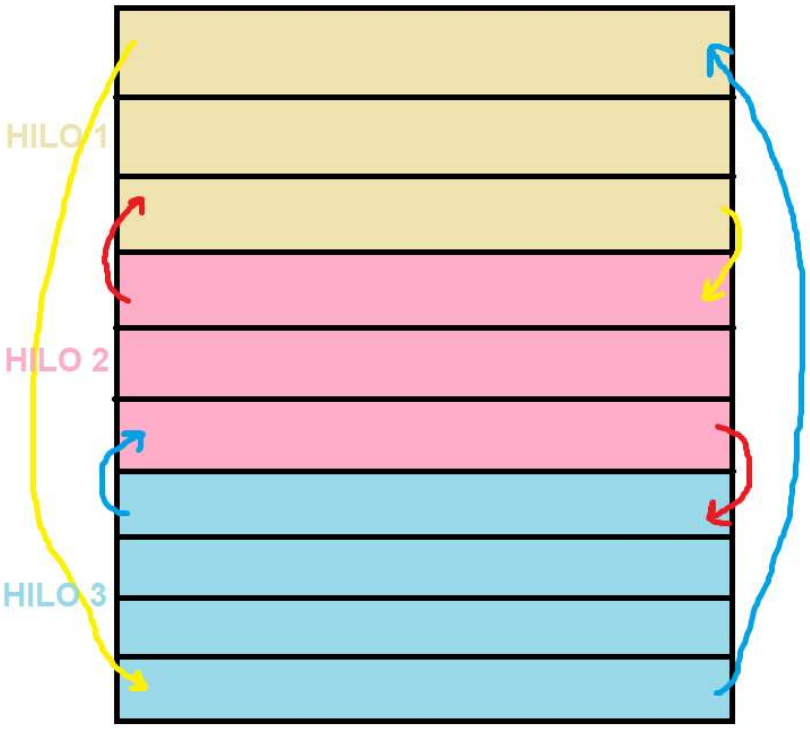
La célula negra depende del valor de las células rojas para actualizar su estado



Sabiendo esto, podemos decir que cada fila depende de su fila inferior y

superior para actualizarse, es decir, podemos separar tablero en filas

para luego asignarle un conjunto de filas a cada hilo, donde la dependencia entre las filas de cada sector depende de la siguiente manera:



Ahora, podemos ejecutar concurrentemente nuestro programa donde cada hilo itera un bloque de filas.

Estructura que usamos para la concurrencia: **Semoforos Posix**

Esta decisión se debe a que, al ejecutarse los hilos de manera concurrente, un hilo no puede ejecutarse de manera continua sin tener en cuenta lo que estén haciendo sus hilos adyacentes, esto se debe a que, la ultima fila de un bloque correspondiente al hilo i, depende de la primera fila del hilo i+1 para poder aplicar las reglas del juego (tal como se ve en la figura de arriba).

Por lo tanto, para poder actualizar los valores del tablero, debemos esperar a que todos los hilos finalicen su turno. Por lo tanto, lo que hacemos es dormir el proceso hasta que todos los demás finalicen (utilizando semáforos), así se actualizan los valores y estos se vuelven a ejecutar.

Una vez completado todos los ciclos, el juego se dará por finalizado.

A la hora de crear los hilos, nos vimos con el problema de que debíamos pasarle muchos argumentos a la función `vida()`, por lo cual creamos una estructura llamada args donde guardamos todos los argumentos que entran en esa función, para que así se pueda crear el hilo.

Para compilar el archivo se provee un archivo makefile, con el cual en la consola al ingresar `make simulador` se compilara.