Computer Architecture and System Programming Laboratory - 2016/Spring

www.cs.bgu.ac.il |

Assignment 3

Further proficiency in assembly language. Understanding stack manipulations, as needed to create a co-routine (equivalently thread) management scheme.

Program Goal

Simulating a variant of Conway's Game of Life (for those who are interested, the original game is described here), along with many of its variants.

In the variant to be implemented in **this assignment**, every organism is a cell in a grid made of **hexagonal close-packed disks**, so it has 6 neighbors (see an example here). The LIFE board is toroidal.

The program begins by reading the initial state configuration of the organisms managed by the co-routines, the number of generations to run, and the printing frequency (in steps).

The program initializes an appropriate mechanism, and control is then passed to a scheduler co-routine which decides the appropriate scheduling for the co-routines. The states of the organisms are managed by the co-routines: each co-routine is responsible for **one** organism (a cell in the array).

The cell organisms change according to the following rules: if the cell is currently alive, then it will remain alive in the next generation if and only if exactly 3 or 4 of its neighbors are currently alive. Otherwise it dies. A dead cell remains dead in the next generation, unless it has exactly 2 living neighbors, in which case we say that an organism is born here. These rules are called the "B2, S34" rules in the "game of life" webpage

A specialized co-routine called the **printer** prints the organism states for all the cells as a two dimensional hexagonal grid.

User Command Line Input

Your program will be run using the following command-line formats:

```
> ass3 <filename> <length> <width> <t> <k>
> ass3 -d <filename> <length> <width> <t> <k>
```

where:

-d (debug) option
If used, you should print at the beginning of the program the input, in a manner that Printer co-routine

does, and the values of the command line arguments.

Example:

```
>ass3 inputExample.txt 5 5 10 3
length=5
width=5
number of generations=10
print frequency=3
0  0  0  1  0  0
  0  1  0  0  1  0
0  0  0  0  0  0
  0  1  0  0  1  0
0  0  0  1  0  0
```

Initial Configuration

Filename is the name of the file containing the initial state. The file include just space ' ' (one space between cells' values) ,'0' (dead), '1'(alive) and newline character after every width*2 characters. Width and length are the dimensions of the co-routine array, you may assume that both of them are even. Maximum value of length and width is 60. Space is the first character in each even line and appears before the newline character in each odd line. (This strange setup is done so that when viewed as normal text, the 2D configuration will appear hexagonal, see inputExample.txt, which you can use as an input example.)

This file determines the initial state of the organisms. You should allocate a global array called **"state"**, which contains all the organisms' state. Every co-routine will be able to access this array for reading and writing. After reading the arguments and file you need to allocate space and initialize the co-routine structures for the co-routine controlling each cell properly. Two additional co-routines must be initialized: the scheduler and the printer, as described later. Number of generations:
The **fourth** command-line argument, *t*, is the number of generations for the scheduler to iterate.

Printing frequency

Next, *k*, is the number of ***steps*** to be done by the scheduler between calls to the printer, i.e. after each *k* "resumes" of cell co-routines, there needs to be one to the printer.

You may assume that the inputs to the program are correct.

Cell

The cell co-routine's main function receives its x, y coordinates as unsigned int (4 bytes each) arguments, using the C calling convention. Each living organism is represented by 0 (dead) or 1 (alive), stored the appropriate location in a globally accessible array. In any given "time slice", a cell co-routine tries to move its cell from generation j to j+1. It does so in two stages:
(1) read the current state (its own cell, and its neighbors' cells) and compute the new state, but do not store it in the array yet.
(2) Update the cell to the new state in the global cell array.
In our simulation, we act as if there is insufficient time to do both in one "time slice", so it must resume the scheduler after step 1 and after step 2. In other words, a cell co-routine loops (forever) over: Step (1) –> resume scheduler –> Step (2) –> resume scheduler
Each generation thus consists of the above two stages.

Printer

Once in a while the scheduler transfers control to the printer co-routine. In each time slice the printer prints the **current states** of all cells as a (width*2) * length table in the same format of the initialization file (use the newline character at the end of every line).

Scheduler

After initialization the "main" program transfers control to a scheduler co-routine. The scheduler co-routine must be named "scheduler" and be declared global. All co-routines are implemented in assembly language.

The scheduler receives two arguments: the number of generations, and *k* (as given by the user). The scheduler transfers control to each one of the co-routine cells, and after each *k* transfers, it gives "time" to the printer co-routine.

Every cell co-routine must be visited **i times** before the scheduler resumes another cell co-routine for the **i+1 time**. That is, the scheduling of the cell co-routines is "round robin". The scheduler returns to the main program when the last generation is reached, after giving one last "time slice" to the printer.

Important Notes:

- The scheduler co-routine MUST be exclusively written in a separate file, the actual control transfer (context switch) should be done with the resume mechanism. Hence, label **resume** would be declared as extern to the scheduler, and register ebx would be used to transfer control.
- The relevant argument needs to be pushed onto the scheduler's stack during the initialization phase, so that they appear as if the scheduler is called with arguments using the C calling convention. Likewise for the cell co-routine cell coordinates.
- You are **required to implement** a simple scheduler which will iterate ***t*** times through the cells (other co-routines) in a round-robin manner.
  Every *k* "resumes" of cell co-routines, the scheduler transfer control to the printer.
- Different implementations of the scheduler **will** be examined by the checker.

Naming conventions:
Some global variables are required (to reduce the number of function arguments, for your convenience):

- As assembly language does not support array sizes, you are required to have global variables named **WorldWidth** and **WorldLength** to hold the number of cells in the simulation. Make sure you initialize it (based on the first command-line argument) before you call the scheduler to start iterations.
- An "array" **cors**, is actually a pointer to ***WorldWidth * WorldLength + 2*** stack tops of: scheduler,printer,cell_1,cell_2,…,$\text{cell}_{\text{WorldWidth * WorldLength}}$.
  We will use this to "resume" the different co-routines (note that the order is important, since we identify co-routines by their index).

As stated above, we may test your code with different schedulers. Hence, this naming convention is crucial!

Submission Instructions

You are to submit a single zip file containing ass3.c (main), scheduler.s, (**only** scheduler co-routine code), printer.s, coroutines.s. Your executable must be named ass3. People who prefer to do everything in assembly language can submit ass3.s (main) instead of ass3.c and can still use the same C standard library functions, of course.

Make sure you follow the coding and submission instructions correctly.
Submissions which deviate from these instructions will not be graded!

Good Luck!