FACULTAD DE INGENIERIA
Universidad de Buenos Aires

UBA FACULTAD DE INGENIERÍA

66.20 Organización de Computadoras

Trabajo Practico

2^{do} Cuatrimestre 2016

Grupo 1

Integrantes:

Federico Rodriguez	93336
fede.longhi@hotmail.com	
Ezequiel Dufau	91985
fede.longhi@hotmail.com	
Pablo Ascarza	89711
fede.longhi@hotmail.com	

Índice

1. Enunciado

2. Introducción

El presente trabajo tiene como objetivo familiarizarse con el conjunto de instrucciones *MIPS-32* y el concepto de ABI. Para el cumplimiento de este objetivo se desarrolló un programa que simula el *"Juego de la Vida" de Conway* según lo detallado en el enunciado.

La implementación se realizo en el lenguaje de programacion C. Además se desarrollo una porción en assembler *MIPS-32* que luego será detallada.

El programa fue desarrollado para correr sobre una plataforma *NetBSD / MIPS-32* mediante el emulador *GXEmul*.

3. Utilización

El programa fue implementado para que cumpliera con los requisitos pedidos por el tp. En las siguientes secciones se detallarán los diferentes aspectos para la ejecución del programa.

3.1. Compilación y Ejecución

1. Descargar el archivo fuente "conway.c"
2. Compilar el archivo (por ejemplo con gcc:
   `gcc -Wall -c ‘‘conway.c’’`
   `gcc -Wall -o ‘‘conway’’ ‘‘conway.c’’`)
3. Ejecutar el programa con: `./conway i M N input [-o output]`

3.2. Documentación de Parámetros

- `i` es la cantidad de simulaciones que queremos realizar.
- `M` y `N` especifican las dimensiones de la matriz sobre la cual queremos simular.
- `input` es el nombre del archivo que contiene las coordenadas de las celdas vivas e identifica el estado inicial de la matriz.
- `-o` es un parámetro opcional que especifica que se utilizará el nombre `output` como prefijo de los nombres de los archivos pbm generados. En caso de no existir este parámetro tomara como prefijo `input`.
- `-V` o `--version` muestra la versión del programa.
- `-h` o `--help` muestra la ayuda.

3.3. Documentación de Errores

A continuación se detallan los errores y su significado:

- **Actions Count must be a positive integer:** El primer parámetro tiene que ser un entero positivo.
- **Number of rows must be a positive integer:** El número de filas tiene que ser un entero positivo.
- **Number of columns must be a positive integer:**El número de columnas tiene que ser un entero positivo.
- **Invalid parameter:** Para el caso de `-V`, `-h` y `-o`. El parámetro no coincide con estos valores (o sus equivalentes).
- **Wrong number of parameters:** Hay parámetros de mas o de menos (se pasó un número de parámetros distinto a 1 o 6).
- **Error while opening input file:** No se pudo abrir el archivo de entrada.
- **Error while opening output file: [nombre_de_archivo]:** No se pudo abrir el archivo de salida con el nombre *nombre_de_archivo*

3.4. Algunas Aclaraciones

- Las imágenes pbm generadas se guardan en la carpeta imágenes.
- Todos los errores se imprimen directamente a `stderr`.

3.5. Ejemplos de Uso

Para ver la documentación:

```
./conway -h
```

Para ver la informacion sobre la version:

```
./conway -V
```

Para generar un tablero de 100×50 a partir del archivo `glider` y realizar 20 iteraciones:

```
./conway 20 100 50 glider
```

Los archivos pbm generados por el comando anterior seran nombrados de la forma: “glider_N.pbm”.

Para generar un tablero de 20×30 a partir del archivo `pento`, realizar 10 iteraciones y que los archivos pbm generados tengan como prefijo el nombre `jorge`:

```
./conway 10 20 30 pento -o jorge
```

Los archivos pbm generados por el comando anterior seran nombrados de la forma: “jorge_N.pbm”.

4. Implementación

En esta sección se presentar porciones del programa. Para ver el código completo dirigirse al apéndice A.

Para la implementación se diseño una función en C según la documentación del enunciado que corresponde a:

```
unsigned int vecinos(unsigned char *a,
unsigned int i, unsigned int j,
unsigned int M, unsigned int N);
```

Tenemos que considerar un detalle sobre el tratamiento de la matriz para entender el algoritmo. En la figura siguiente se muestra la conversión de la matriz a array que utilizamos.

$$\begin{bmatrix} 0,0 & 0,1 & 0,2 \\ 1,0 & 1,1 & 1,2 \\ 2,0 & 2,1 & 2,2 \end{bmatrix} \rightarrow \begin{bmatrix} \underset{0}{0,0} & \underset{1}{0,1} & \underset{2}{0,2} & \underset{3}{1,0} & \underset{4}{1,1} & \underset{5}{1,2} & \underset{6}{2,0} & \underset{7}{2,1} & \underset{8}{2,2} \end{bmatrix}$$

A continuación se muestra el algoritmo implementado en C:

Código 1: Código de la funcion vecinos

```
1  unsigned int vecinos(unsigned char *a, unsigned int i, unsigned
       int j, unsigned int M, unsigned int N){
2    unsigned int vecinos = 0;
3    int x,y;
4    for (x = -1; x<2; x++){
5      for (y = -1; y<2; y++){
6        if (x+i>=0 && x+i<N && y+j>=0 && y+j<M){
7          if ((!(x==0&&y==0)) && a[N*(x+i) + (y+j)] == '1'){
8            vecinos++;
9          }
10       }
11     }
12   }
```

```
13    return vecinos;
14  }
```

5. Documentación Diseño e Implementación

5.1. Diagrama del Stack de Vecinos

6. Corridas de Prueba

7. Conclusiones

Apéndice

A. Código Fuente