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.

La implementación se hizo completamente en C. Luego se programo en MIPS la función vecinos. A continuación se detallan las dos implementaciones.

4.1. Implementación en C

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++){
```

```
      if (x+i>=0 && x+i<N && y+j>=0 && y+j<M){
        if ((!(x==0&&y==0)) && a[N*(x+i) + (y+j)] == '1'){
          vecinos++;
        }
      }
    }
  }
  return vecinos;
}
```

4.2. Implementación en MIPS

4.2.1. Diagrama del Stack de Vecinos

5. Corridas de Prueba

5.1. Funcionamiento

Realizamos varias corridas de prueba con los archivos proporcionados (pento, glider y sapo) para verificar el correcto funcionamiento del programa. Mostramos a continuación algunas salidas de las corridas:

`./conway 10 10 10 glider -s`

```
fedelonghi@fedelonghi-dell $ ./conway 5 10 10 glider -s
starting action
Simulation number: 0

..........
..........
..........
....O.....
.....O....
...OOO....
..........
..........
..........
..........
Simulation number: 1

..........
..........
..........
..........
...O.O....
....OO....
....O.....
..........
..........
..........
Simulation number: 2

..........
```

```
..........
..........
..........
.....O....
...O.O....
....OO....
..........
..........
..........
Simulation number: 3

..........
..........
..........
..........
....O.....
.....OO...
....OO....
..........
..........
..........
Simulation number: 4

..........
..........
..........
..........
.....O....
......O...
....OOO...
..........
..........
..........
```

./test 10 10 10 glider -o archivo_salida

```
fedelonghi@fedelonghi-dell $ ./test 10 10 10 glider -o
    archivo_salida
starting action
```

Los archivos .pbm fueron generados en el directorio imagenes.

./test 3 10 10 sapo -s

```
fedelonghi@fedelonghi-dell $ ./test 3 10 10 sapo -s
starting action
Simulation number: 0

..........
..........
..........
..........
....OOO...
...OOO....
..........
```

```
..........
..........
..........
Simulation number: 1

..........
..........
..........
.....O....
...O..O...
...O..O...
....O.....
..........
..........
..........
Simulation number: 2

..........
..........
..........
..........
....OOO...
...OOO....
..........
..........
..........
..........
```

5.2. Resultados

Además medimos los tiempos de ejecucion para el programa compilado enteramente en c y el compilado con la función vecinos en mips. A continuación se muestran los resultados[1]:

input	it	M	N	time 1	time 2	time 3
pento	10	10	10	0,098	0,117	0,102
	100	10	10	0,848	0,836	0,816
	10	20	20	0,832	0,781	0,750
glider	10	10	10	0,102	0,102	0,102
	100	10	10	0,836	0,836	0,785
	10	20	20	0,766	0,77	0,746
sapo	10	10	10	0,102	0,117	0,102
	100	10	10	0,848	0,836	0,816
	10	20	20	0,73	0,766	0,750

Tabla 1: Tiempo de ejecución del programa compilado enteramente C

[1]Todos los tiempos están medidos en segundos.

input	it	M	N	time 1	time 2	time 3
pento	10	10	10	0,098	0,098	0,102
	100	10	10	0,797	0,801	0,77
	10	20	20	0,746	0,77	0,801
glider	10	10	10	0,102	0,098	0,082
	100	10	10	0,816	0,801	0,781
	10	20	20	0,77	0,73	0,75
sapo	10	10	10	0,098	0,102	0,098
	100	10	10	0,801	0,77	0,77
	10	20	20	0,785	0,766	0,77

Tabla 2: Tiempo de ejecución del programa compilado con la función vecinos en mips

Para tener una mejor noción sobre los tiempos a continuación se muestra el promedio. El promedio se calculo sin considerar el tipo de archivo de input, es decir considerando los todos los tiempos pertenecientes a las ejecuciones con los mismos parámetros i M y N.

Parámetros	Tiempo (C)	Tiempo (Mips)
10 10x10	0,1048	0,0975
100 10x10	0,8285	0,7896
10 20x20	0,7656	0,7653

Tabla 3: Promedio de los tiempos de ejecución

5.3. Análisis de Resultados

Observando las tablas podemos ver claramente que hay una tendencia a disminuir el tiempo de ejecucion utilizando el programa compilado en mips. Esto se ve especialmente analizando la tabla 3. Donde claramente el promedio en todos los casos es menor en mips que en C.

6. Conclusiones

Además de familiarizarnos con las herramientas y el set de instrucciones de Mips pudimos observar como programando una función en mips podemos reducir el costo de tiempo en la ejecucion de un programa.

A la vista de los resultados, la ganancia es muy pequeña. Podemos deducir que se trata debido a que el programa en sí consta de otras partes que tienen mucha injerencia en el costo de tiempo del programa (como puede ser la creación de archivos pbm).

Apéndice

A. Código Fuente

Código 2: cornway.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

void writePBM(unsigned char** board, unsigned int dimx,
    unsigned int dimy, const char* fileName, unsigned int
    actionNumber)
{
int i, j;
int actionNumberStringLenght = 1;
if (actionNumber >= 10){
actionNumberStringLenght = 2;
}
char str[actionNumberStringLenght];
sprintf(str, "%d", actionNumber);
char newFileName[50];
newFileName[0]='\0';
strcat(newFileName,"imagenes/");
strcat(newFileName,fileName);
strcat(newFileName,"_");
strcat(newFileName,str);
strcat(newFileName,".pbm");

FILE *fp = fopen(newFileName, "wb");
if (fp==NULL){
fprintf(stderr, "Error while opening output file: %s\n",
    newFileName);
exit(1);
}

unsigned int amp = 1;
(void) fprintf(fp, "P4\n%d %d\n", dimy*amp, dimx*amp);
for (j = 0; j < dimy; ++j){
for (i = 0; i < dimx; ++i)
{
unsigned int x,y;
unsigned char writeValue = 0;
if ((board[i][j]) == '1'){
writeValue = 1;
}
for (x = i*amp; x < amp*(i+1); x++){
for (y = j*amp; y < amp*(j+1); y++){
fprintf(fp, "%c", writeValue);
}
}
```

```
}
}
(void) fclose(fp);
return;
}


unsigned char ** init_board(unsigned int rows, unsigned int
    cols){
unsigned char** board = (unsigned char**)malloc(cols*sizeof(
    char*));
unsigned int i,j;
for (i = 0; i<cols; i++){
board[i] = (unsigned char *) malloc(rows*sizeof(char*));
for (j = 0; j<rows; j++){
board[i][j]='0';
}
}
return board;
}


void load_board(unsigned char **board, unsigned int rows,
    unsigned int cols, FILE *file){
int x,y;
while (fscanf (file, "%i %i", &x, &y)!=EOF){
if (x<cols||y<rows){
board[x][y]='1';
}
}
}


int free_board (unsigned char **board, unsigned int rows,
    unsigned int cols){
unsigned int i;
for (i = 0; i<cols; i++){
free(board[i]);
}
free(board);
return 0;
}


void print_board(unsigned char **board, unsigned int rows,
    unsigned int cols){
unsigned int i,j;
for (i = 0; i<cols; i++){
for (j = 0; j<rows; j++){
if (board[i][j] == '0'){
printf(".");
}else{
printf("0");
}
```

```
93  }
94  printf("\n");
95  }
96  }
97
98  void print_board_file(unsigned char **board, unsigned int rows,
        unsigned int cols,int n,const char *filename){
99  char newFilename [50];
100 int numLength = 1;
101 if (n >= 10 && n <= 99){
102 numLength = 2;
103 }else if (n >= 100 && n <= 999){
104 numLength = 3;
105 }
106 char str[numLength];
107 sprintf(str, "%d", n);
108 newFilename[0] = '\0';
109 strcat(newFilename, "output/");
110 strcat(newFilename, filename);
111 strcat(newFilename, "_");
112 strcat(newFilename, str);
113 FILE *file = fopen(newFilename,"w+");
114 fprintf(file, "Simulacion N: %i\n",n);
115 unsigned int i,j;
116 for (i = 0; i<cols; i++){
117 for (j = 0; j<rows; j++){
118 if (board[i][j] == '0'){
119 fprintf(file,".");
120 }else{
121 fprintf(file,"0");
122 }
123 }
124 fprintf(file,"\n");
125 }
126 fclose(file);
127 }
128
129
130 unsigned char* board_to_array(unsigned char ** board, unsigned
        int rows, unsigned int cols){
131 unsigned char* array = (unsigned char*)malloc(sizeof(char*)*
        rows*cols);
132 int x,y;
133 for (x=0; x<cols; x++){
134 for (y=0; y<rows; y++){
135 unsigned int pos = cols*x+y;
136 array[pos]=board[x][y];
137 }
138 }
139 return array;
140 }
141
142
```

```
143 unsigned int vecinos(unsigned char *a, unsigned int i, unsigned
        int j, unsigned int M, unsigned int N){
144 unsigned int vecinos = 0;
145 int x,y;
146 for (x = -1; x<2; x++){
147 for (y = -1; y<2; y++){
148 if (x+i>=0 && x+i<N && y+j>=0 && y+j<M){
149 if ((!(x==0&&y==0)) && a[N*(x+i) + (y+j)] == '1'){
150 vecinos++;
151 }
152 }
153 }
154 }
155 return vecinos;
156 }
157
158
159 unsigned int vecinos_m(unsigned char **board, unsigned int x,
        unsigned int y, unsigned int rows, unsigned int cols){
160 unsigned int vecinos = 0;
161 int i,j;
162 for (i = -1; i<2; i++){
163 for (j = -1; j<2; j++){
164 if (x+i>=0 && x+i<cols && y+j>=0 && y+j<rows){
165 if ((!(i==0&&j==0)) && board[x+i][y+j] == '1'){
166 vecinos++;
167 }
168 }
169 }
170 }
171 return vecinos;
172 }
173
174
175 void copy_board(unsigned char **from, unsigned char **to,
        unsigned int rows, unsigned int cols){
176 unsigned int i,j;
177 for (i = 0; i<cols; i++){
178 for (j = 0; j<rows; j++){
179 to[i][j] = from[i][j];
180 }
181 }
182 }
183
184
185 void process_board(unsigned char **board, unsigned int rows,
        unsigned int cols, unsigned int actionCount,const char*
        fileName,bool screen){
186 printf ("starting action\n");
187 unsigned char **thisBoard = init_board(rows, cols);
188 copy_board(board,thisBoard,rows, cols);
189 unsigned int i,x,y;
190 for (i = 0; i < actionCount; i++){
191 unsigned char **nextBoard = init_board(rows, cols);
```

```
192 if (screen){
193 printf("Simulation number: %i\n\n", i);
194 print_board(thisBoard,rows,cols);
195 print_board_file(thisBoard,rows,cols,i,fileName);
196 }else{
197 writePBM(thisBoard, rows, cols, fileName, i);
198 }
199 for (x = 0; x < cols; x++){
200 for (y = 0; y < rows; y++){
201 unsigned char * array = board_to_array(thisBoard, rows, cols);
202 unsigned int neighbours = vecinos(array, x, y, rows, cols);
203 free(array);
204 //unsigned int neighbours = vecinos_m(thisBoard,x,y,rows,cols);
205 if (thisBoard[x][y]=='1'){
206 if (neighbours == 3 || neighbours == 2){nextBoard[x][y] = '1';}
207 }else{
208 if (neighbours == 3){nextBoard[x][y] = '1';}
209 }
210 }
211 }
212 copy_board(nextBoard,thisBoard,rows, cols);
213 free_board(nextBoard, rows, cols);
214 }
215 free_board(thisBoard, rows, cols);
216 }
217
218
219 void print_help(){
220 printf("Uso:\n");
221 printf("  conway -h\n  conway -V\n  conway i M N inputfile [-o
    outputprefix]\n");
222 printf("Opciones:\n");
223 printf("  -h, --help    Imprime este mensaje\n");
224 printf("  -V, --version Da la version del programa\n");
225 printf("  -o            Prefijo de los archivos de salida\n");
226 printf("Ejemplos: \n");
227 printf("  conway 10 20 20 glider -o estado\n");
228 printf("  Representa 10 iteraciones del Juego de la Vida en una
     Matriz\n");
229 printf("  de 20x20, con un estado inicial tomado del archivo ''
    glider''.\n");
230 printf("  Los archivos de salida se llamarán estado_n.pbm,\n");
231 printf("  si no se da un prefijo para el archivo de salida, \n"
    );
232 printf("  el prefijo será el nombre del archivo de entrada.\n")
    ;
233 }
234
235
236 void print_version (){
237 printf("Conway -Game of Life- version: 1.0\n");
238 }
239
240
```

```
241 void validate_actionsCount(int actionsCount){
242 if (actionsCount <= 0){
243 fprintf(stderr, "Actions Count must be a positive integer!\n");
244 exit(1);
245 }
246 }
247
248
249 void validate_rows(int rows){
250 if (rows <= 0){
251 fprintf(stderr, "Number of rows must be a positive integer!\n")
        ;
252 exit(1);
253 }
254 }
255
256
257 void validate_cols(int cols){
258 if (cols <= 0){
259 fprintf(stderr, "Number of columns must be a positive integer!\
        n");
260 exit(1);
261 }
262 }
263
264
265 int main(int argc, char* argv[])
266 {
267 char const *fileName;
268 char const *outputFileName;
269 unsigned int actionsCount;
270 unsigned int rows;
271 unsigned int cols;
272 unsigned char **board;
273 bool screen = 0;
274 //asigno valores de parametros
275 if (argc == 2){
276 char* arg = argv[1];
277 if (strcmp(arg,"-h") == 0 || strcmp(arg,"--help") == 0){
278 print_help();
279 return 0;
280 }
281 else if (strcmp(arg,"-v") == 0 || strcmp(arg,"--version") == 0)
        {
282 print_version();
283 return 0;
284 }
285 else{
286 fprintf(stderr,"Invalid parameter!\n");
287 exit(1);
288 }
289 }else if (argc != 5 && argc != 7 && argc != 6){ //ver
290 fprintf(stderr,"Wrong number of parameters!\n");
291 exit(1);
```

```
}else{
fileName = argv[4];
actionsCount = (int) atoi(argv[1]);
validate_actionsCount(actionsCount);
rows = (int) atoi(argv[2]);
validate_rows(rows);
cols = (int) atoi(argv[3]);
validate_cols(cols);
outputFileName = fileName;
if (argc == 7){
if (strcmp(argv[5],"-o") == 0){
outputFileName = argv[6];
}else{
fprintf(stderr,"Invalid parameter!\n");
exit(1);
}
}
if (argc == 6){
if (strcmp(argv[5],"-s") == 0){
screen = 1;
}else{
fprintf(stderr,"Invalid parameter!\n");
exit(1);
}
}

}

FILE* file = fopen(fileName, "r");
if (file==NULL){
fprintf(stderr, "Error while opening input file\n");
exit(1);
}

board = init_board(rows, cols);
load_board(board, rows, cols, file);
fclose(file);
process_board(board, rows, cols, actionsCount, outputFileName,
    screen);
free_board(board, rows, cols);

return 0;
}
```