Trabajo Práctico 1
Conjunto de instrucciones MIPS

Jonathan Medina, *Padrón 100052 Slack: U011C1EGWMD*
jcmedina@fi.uba.ar

Pablo Inoriza, *Padrón 94986 Slack: U011Q8GB197*
pinoriza@fi.uba.ar

Christian Flórez Del Carpio, *Padrón 91011 Slack: U011G8YJ18T*
chflorez@fi.uba.ar

1do. Cuatrimestre de 2020
86.37 / 66.20 Organización de Computadoras – Práctica Jueves
Facultad de Ingeniería, Universidad de Buenos Aires

1. Introducción

Se realizó una recreación en C con portabilidad en MIPS del 'Juego de la Vida' de Conway el cuál nos permitió familiarizarnos con el concepto del ABI y la memoria. En el camino nos topamos con problemas de como funcionaba la memoria para la representacion "matricial"pero pudimos encontrar la solución a dicho problema. Siempre se busco realizar un diseño simple en C y sobre todo para la función vecinos ya que la misma despues fue implementada en assembler MIPS32 y de está manera la traducción de C resultó quasi-directa.

2. Diseño e Implementación

Como ya aclaramos en la introduccción, tuvimos un problema de como funcionaba la memoria, en un principio intentamos utilizar un char** contra un char* para representar la matriz fuera de la sección donde luego también tendríamos que traducir a assembler MIPS32. Nos dimos cuenta del error, cuando al debuggear con GDB, en un momento determinado se accedía afuera de la memoria reservada. Esto era causado, ya que reservabamos memoria N veces(una por cada fila que tenia nuestra matriz), está nos daba N direcciones distintas, entonces no se podía recorrer la memoria de forma contigua.

Es decir la diferencia entre utilizar:

```
unsigned char* a = malloc(sizeof(unsigned char)*M*N);
```

y:

```
unsigned char** a = malloc(sizeof(unsigned char*)*M);
for (int i=0; i<M; i++) {
     a[i]=malloc(sizeof(unsigned char)*N)
}
```

Con respecto a las condiciones de contorno, aplicamos la hipótesis del mundo toroidal, el cuál lo solucionamos con el diseño de obtener los vecinos que están a los costados del punto central en cuestion, entonces se recorre en forma vertical obteniendo los costados de la parte central, superior e inferior, teniendo las consideraciones si se encontraba en los limites de la matriz.

Tambien nos encontramos con un problema al momento de leer el archivo y parsearlo, ya que los archivos provistos todos terminaban con un '\n' esto nos dimos cuenta leyendo el binario mediante el comando de bash: xxd ejemplo:

```
$ xxd glider
00000000: 3520 330a 3520 340a 3520 350a 3320 340a  5 3.5 4.5 5.3 4.
00000010: 3420 350a
```

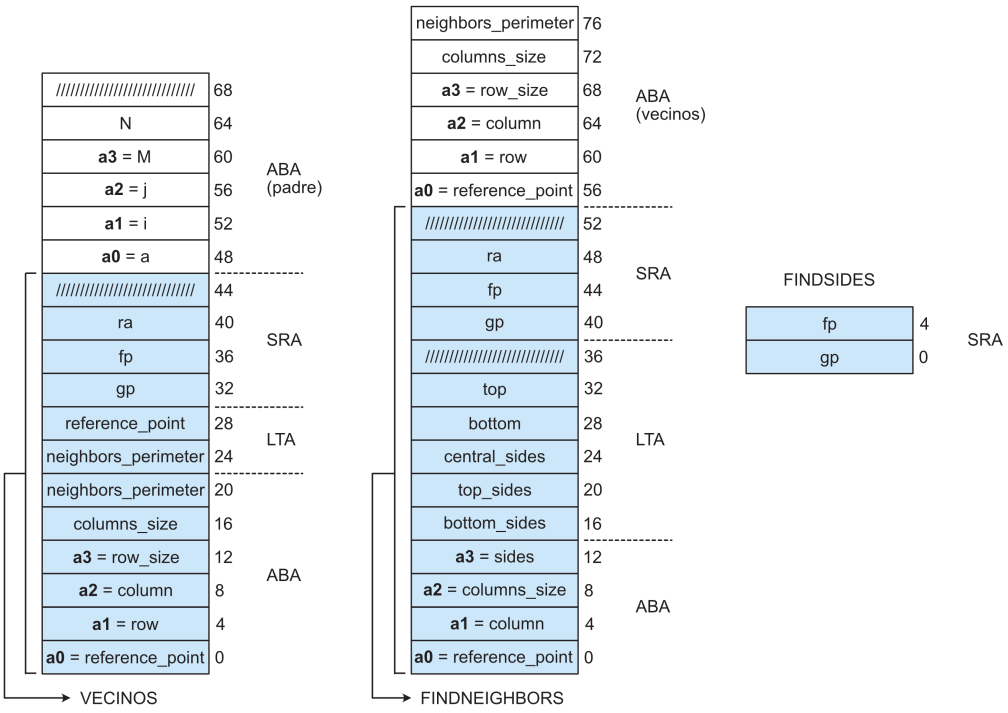
donde 0a representa en ascii el line feed.

2.1. Desarrollo del Código Fuente

1. **archivos.h**: Funciones para el manejo de entrada y salida de los datos mediante archivos.
2. **game.h**: Funciones con toda la lógica del Juego de la Vida.
3. **constants.h**: Constantes definidas para todo el programa.

2.2. Desarrollo en MIPS

Para este desarrolo, a partír de las funciones implementadas en C con *vecinos*, *find_neighbors* y *find_sides*, se procedió a diseñar sus respectivos stacks para inicializar su implementación con MIPS.



Esto nos ayudó no solo a respetar la ABI, sino también a orientarnos a la hora de escribir código con MIPS.

Para diseñar cada uno de los stacks se tuvo en cuenta que la cantidad de palabras de cada región (SRA, LTA y ABA) sea múltiplos de 8 bits. Para aquellos que no cumplían la multiplicidad, se lo completaba con un padding.

Las funciones involucradas en el desarrollo con MIPS son:

1. **vecinos:** Se trata de una función no hoja (por ese motivo se tuvo que guardar el registro ra en el SRA) debido a que invoca a la función *find_neighbors* en la que pide 6 parámetros, los primeros 4 se almacenan en los registros a0..a3 mientras que los 2 restantes se almacenan en el ABA de vecinos (posiciones 16 y 20 del diagrama).
2. **find_neighbors:** Se trata de una función no hoja (se tuvo que guardar el registro ra en el SRA) debido a que invoca a la función *find_sides* en la que pide 4 parámetros, que son almacenados en los registros a0..a3.
3. **find_sides:** Se trata de una función hoja ya que no invoca ninguna otra función, ni tampoco posee variables locales, motivo por el cual cuenta únicamente con SRA.

3. Proceso de Compilación

Se cuenta con un archivo Makefile que indica las reglas de compilación ejecutadas por el comando make.

El código fuente se compila ejecutando el comando make con alguna de las posibles reglas. En todos los casos se compila con los flags de warnings y de debbugging prendidos.

make tp.c: Crea el ejecutable utilizando el archivo escrito en C.

make tp.mips: Crea el ejecutable utilizando el archivo escrito en mips.

Para eliminar todos los archivos generados por el comando make, se puede ejecutar con la regla clean de la forma:

make clean

4. Portabilidad

Dado que se utilizó el lenguaje de programación C, sin hacer uso de funciones específicas de sistemas operativos, o de bibliotecas comerciales, los programas pueden ser compilados en varios de ellos. De todas formas, el Makefile presentado fue hecho particularmente para sistemas de tipo UNIX.

Los casos de prueba presentados fueron llevados a cabo en una arquitectura MIPS emulada. El programa de emulación utilizado fue QEMU y el sistema operativo en la máquina emulada fue Debian.

Los programas también fueron probados en Ubuntu-Linux corriendo en una arquitectura Intel x86 satisfactoriamente.

5. Casos de Prueba

Se corrio la aplicacion tanto en:

```
$ uname -a
Linux PMIT201206 4.15.0-101-generic #102-Ubuntu SMP Mon May 11 10:07:26 UTC 2020 x86_64 x86_
```

```
$ uname -a
Linux debian-stretch-mips 4.9.0-4-5kc-malta #1 Debian 4.9.65-3 (2017-12-03) mips64 GNU/Linux
```

Se probo todos los casos provistos por la catedra de que realmente generaba los archivos correctos, comparando la codificación PBM con lo esperado.

A continuación mostraremos los casos básicos de prueba realizados a la aplicación.

5.1. Archivos de entrada incorrectos

Se probo varios casos diferentes de que el programa resistia todo tipo de entrada. Con archivos inexistentes

```
$ valgrind --leak-check=full --show-leak-kinds=all
      --track-origins=yes ./conway 5 20 20 notexist  -o estado
==13350== Memcheck, a memory error detector
==13350== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==13350== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==13350== Command: ./conway 5 20 20 notexist -o estado
==13350==
Couldn't load the input
==13350==
==13350== HEAP SUMMARY:
==13350==     in use at exit: 0 bytes in 0 blocks
==13350==   total heap usage: 2 allocs, 2 frees, 952 bytes allocated
==13350==
==13350== All heap blocks were freed -- no leaks are possible
==13350==
==13350== For counts of detected and suppressed errors, rerun with: -v
==13350== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Con archivos mal tipificados

```
$ cat glinder_broken
5 3
5 4
```

```
5 5
3 4
4 aaaa
./conway 5 10 10 glider_broken -o estado
Couldn't load the input


$ cat pento_broken
3 5
3
4 4
4 5
5 5
$ ./conway 5 10 10 pento_broken -o estado
Couldn't load the input


$ cat sapo_broken
5 3
5 4
 5
4 4
4 5
4 6
$ ./conway 5 10 10 sapo_broken -o estado
Couldn't load the input
```

Tambien se puede apreciar que en ningun caso corriendo por valgrind hubo perdida de memoria y/o archivos abiertos

```
$ valgrind --leak-check=full --show-leak-kinds=all
    --track-origins=yes ./conway 10 20 20 sapo_broken -o estado
==12768== Memcheck, a memory error detector
==12768== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==12768== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==12768== Command: ./conway 10 20 20 sapo_broken -o estado
==12768==
Couldn't load the input
==12768==
==12768== HEAP SUMMARY:
==12768==     in use at exit: 0 bytes in 0 blocks
==12768==   total heap usage: 4 allocs, 4 frees, 5,072 bytes allocated
==12768==
==12768== All heap blocks were freed -- no leaks are possible
==12768==
==12768== For counts of detected and suppressed errors, rerun with: -v
==12768== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

5.2. Archivos de entrada provistos

En todos los casos se ve que no hay perdida de memoria / archivos abiertos

```
$ valgrind --leak-check=full --show-leak-kinds=all
        --track-origins=yes ./conway 10 20 20 glider -o estado
==13585== Memcheck, a memory error detector
==13585== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==13585== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==13585== Command: ./conway 10 20 20 glider -o estado
==13585==
Grabando estado_000.PBM
```

```
Grabando estado_001.PBM
Grabando estado_002.PBM
Grabando estado_003.PBM
Grabando estado_004.PBM
Grabando estado_005.PBM
Grabando estado_006.PBM
Grabando estado_007.PBM
Grabando estado_008.PBM
Grabando estado_009.PBM
Listo
==13585==
==13585== HEAP SUMMARY:
==13585==     in use at exit: 0 bytes in 0 blocks
==13585==   total heap usage: 45 allocs, 45 frees, 56,723 bytes allocated
==13585==
==13585== All heap blocks were freed -- no leaks are possible
==13585==
==13585== For counts of detected and suppressed errors, rerun with: -v
==13585== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)


$ valgrind --leak-check=full --show-leak-kinds=all
        --track-origins=yes ./conway 10 20 20 pento -o estado
==13661== Memcheck, a memory error detector
==13661== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==13661== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==13661== Command: ./conway 10 20 20 pento -o estado
==13661==
Grabando estado_000.PBM
Grabando estado_001.PBM
Grabando estado_002.PBM
Grabando estado_003.PBM
Grabando estado_004.PBM
Grabando estado_005.PBM
Grabando estado_006.PBM
Grabando estado_007.PBM
Grabando estado_008.PBM
Grabando estado_009.PBM
Listo
==13661==
==13661== HEAP SUMMARY:
==13661==     in use at exit: 0 bytes in 0 blocks
==13661==   total heap usage: 45 allocs, 45 frees, 56,723 bytes allocated
==13661==
==13661== All heap blocks were freed -- no leaks are possible
==13661==
==13661== For counts of detected and suppressed errors, rerun with: -v
==13661== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)


18:02 $ valgrind --leak-check=full --show-leak-kinds=all
        --track-origins=yes ./conway 10 20 20 sapo -o estado
==13747== Memcheck, a memory error detector
==13747== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==13747== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==13747== Command: ./conway 10 20 20 sapo -o estado
==13747==
Grabando estado_000.PBM
Grabando estado_001.PBM
Grabando estado_002.PBM
Grabando estado_003.PBM
Grabando estado_004.PBM
```

```
Grabando estado_005.PBM
Grabando estado_006.PBM
Grabando estado_007.PBM
Grabando estado_008.PBM
Grabando estado_009.PBM
Listo
==13747==
==13747== HEAP SUMMARY:
==13747==     in use at exit: 0 bytes in 0 blocks
==13747==   total heap usage: 45 allocs, 45 frees, 56,727 bytes allocated
==13747==
==13747== All heap blocks were freed -- no leaks are possible
==13747==
==13747== For counts of detected and suppressed errors, rerun with: -v
==13747== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Ademas se muestra una salida del PPM generado por la aplicacion:

```
$ ./conway 10 20 20 sapo -o estado
[...]
$ cat estado_000.PBM
P1
20 20
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

6. Conclusión

Después de ejecutar el juego compilado primero con la función vecinos definida en C y luego con la misma definida en assembler MIPS32 obtuvimos los siguientes tiempos de ejecución:

función vecinos	real	user	sys
C	0m0.058s	0m0.028s	0m0.032s
MIPS32	0m0.065s	0m0.040s	0m0.020s

Se puede ver que en C corrió más rápido, dado que la función vecinos definida en C no hace uso de bibliotecas externas, y por ende gcc al traducir el codigo de C a lenguaje máquina genera un codigo mucho más óptimo que el que nosotros desarrollamos para la función vecino en assembler MIPS32.

Referencias

[1] QEMU. https://www.qemu.org/.

[2] MIPS Instruction Reference. http://www.mrc.uidaho.edu/mrc/people/jff/digital/MIPSir.html.

[3] PPM Format. http://netpbm.sourceforge.net/doc/ppm.html.

[4] MIPS Instruction Set. https://www.dsi.unive.it/~gasparetto/materials/MIPS_Instruction_Set.pdf.

[5] ffmpeg. https://www.ffmpeg.org/.

[6] Stop-motion with ffmpeg. https://lukecyca.com/2013/stop-motion-with-ffmpeg.html

[7] B. Kernighan, D.Ritchie, *The C Programming Languaje*. Person Prentice Hall

[8] B. Kernighan, R. Pike, *The Unix Programming Environment*. Prentice Hall

[9] John Conway's Game of Life. https://bitstorm.org/gameoflife/

[10] Valgrind http://valgrind.org/

[11] gcc optimization options https://gcc.gnu.org/onlinedocs/gcc-4.0.3/gcc/Optimize-Options.html#Optimize-Options

A. Enunciado

B. Código fuente