Organización de Computadoras

Trabajo Práctico 0
Infraestructura Básica

Nicolás Ledesma, *Padrón Nro. 93.118*
nicolas.angel.ledesma@gmail.com

Jonathan Moguilevsky, *Padrón Nro. 95.516*
jmoguilevsky@gmail.com

Leonardo Riego, *Padrón Nro. 94.104*
riegoleonardo@hotmail.com

2do. Cuatrimestre de 2019
66.20 Org. de Computadoras − Práctica: Santi, Pérez Masci, Natale
Facultad de Ingeniería, Universidad de Buenos Aires

Resumen

Este trabajo práctico grupal tiene como objetivo principal familiarizarnos con las herramientas de software que usaremos en los siguientes trabajos. Consiste en la resolución de un problema a través de una solución de código en el lenguaje C.

1. Enunciado

Universidad de Buenos Aires - FIUBA
66.20 Organización de Computadoras
Trabajo práctico 0: Infraestructura básica
2^{do} cuatrimestre de 2019

`$Date: 2019/08/27 23:02:40 $`

1. Objetivos

Familiarizarse con las herramientas de software que usaremos en los siguientes trabajos, implementando un programa y su correspondiente documentación que resuelvan el problema descripto más abajo.

2. Alcance

Este trabajo práctico es de elaboración grupal, evaluación individual, y de carácter obligatorio para todos alumnos del curso.

3. Requisitos

El trabajo deberá ser entregado personalmente, en la fecha estipulada, con una carátula que contenga los datos completos de todos los integrantes, un informe impreso de acuerdo con lo que mencionaremos en la sección 6, y con una copia digital de los archivos fuente necesarios para compilar el trabajo.

4. Recursos

Usaremos el programa GXemul [1] para simular el entorno de desarrollo que utilizaremos en este y otros trabajos prácticos, una máquina MIPS corriendo una versión reciente del sistema operativo NetBSD [2].

Durante la primera clase del curso hemos presentado brevemente los pasos necesarios para la instalación y configuración del entorno de desarrollo.

5. Implementación

5.1. Programa

El programa, a escribir en lenguaje C, deberá multiplicar matrices cuadradas de números reales, representados en punto flotante de doble precisión.

Las matrices a multiplicar ingresarán como texto por entrada estándar (`stdin`), donde cáda línea describe completamente cada par de matrices a multiplicar, según el siguiente formato:

$N\ a_{1,1}\ a_{1,2}\ ...\ a_{N,N}\ b_{1,1}\ b_{1,2}\ ...\ b_{N,N}$

La línea anterior representa a las matrices A y B, de NxN. Los elementos de la matriz A son los $a_{x,y}$, siendo x e y los indices de fila y columna respectivamente[1]. Los elementos de la matriz B se representan por los $b_{x,y}$ de la misma forma que los de A.

El fin de línea es el caracter \n (*newline*). Los componentes de la línea están separados entre sí por uno o más espacios. El formato de los números en punto flotante son los que corresponden al especificador de conversión 'g' de `printf`[2].

Por ejemplo, dado el siguiente producto de matrices cuadradas:

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \times \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$$

Su representación sería:

```
2 1 2 3 4 5 6 7 8
```

Por cada par de matrices que se presenten por cada línea de entrada, el programa deberá multiplicarlas y presentar el resultado por su salida estándar (`stdout`) en el siguiente formato, hasta que llegue al final del archivo de entrada (`EOF`):

$N\ c_{1,1}\ c_{1,2}\ ...\ c_{N,N}$

Ante un error, el progama deberá informar la situación inmediatamente (por `stderr`) y detener su ejecución.

[1] Notar que es una representación del tipo *row major order*, siguiendo el orden en que C dispone las matrices en memoria.

[2] Ver man 3 printf, "Conversion specifiers".

5.2. Ejemplos

Primero, usamos la opción `-h` para ver el mensaje de ayuda:

```
$ tp0 -h
Usage:
  tp0 -h
  tp0 -V
  tp0 < in_file > out_file
Options:
  -V, --version     Print version and quit.
  -h, --help        Print this information and quit.
Examples:
   tp0 < in.txt > out.txt
   cat in.txt | tp0 > out.txt
```

A continuación, ejecutamos algunas pruebas:

```
$ cat example.txt
2 1 2 3 4 1 2 3 4
3 1 2 3 4 5 6.1 3 2 1 1 0 0 0 1 0 0 0 1

$ cat example.txt | ./tp0
2 7 10 15 22
3 1 2 3 4 5 6.1 3 2 1
```

En este ejemplo, realizamos las siguientes multiplicaciones, siendo los miembros izquierdos de la ecuación las matrices de entrada (`stdin`), y los miembros derechos las matrices de salida (`stdout`):

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \times \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 7 & 10 \\ 15 & 22 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6.1 \\ 3 & 2 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6.1 \\ 3 & 2 & 1 \end{pmatrix}$$

5.3. Interfaz

Las matrices deberán ser representadas por el tipo de datos `matrix_t`, definido a continuación:

```
typedef struct matrix {
        size_t rows;
        size_t cols;
        double* array;
} matrix_t;
```

Notar que los atributos `rows` y `cols` representan respectivamente la cantidad filas y columnas de la matriz. El atributo `array` contendrá los elementos de la matriz dispuestos en row-major order [3].

Los métodos a implementar, que aplican sobre el tipo de datos `matrix_t` son:

```
// Constructor de matrix_t
matrix_t* create_matrix(size_t rows, size_t cols);

// Destructor de matrix_t
void destroy_matrix(matrix_t* m);

// Imprime matrix_t sobre el file pointer fp en el formato solicitado
// por el enunciado
int print_matrix(FILE* fp, matrix_t* m);

// Multiplica las matrices en m1 y m2
matrix_t* matrix_multiply(matrix_t* m1, matrix_t* m2);
```

5.4. Portabilidad

Como es usual, es necesario que la implementación desarrollada provea un grado mínimo de portabilidad. Para satisfacer esto, el programa deberá funcionar al menos en NetBSD/pmax (usando el simulador GXemul [1]) y la versión de Linux (Knoppix, RedHat, Debian, Ubuntu) usada para correr el simulador, Linux/i386.

6. Informe

El informe deberá incluir:

- Documentación relevante al diseño e implementación del programa;
- Comando(s) para compilar el programa;
- Las corridas de prueba, con los comentarios pertinentes;
- El código fuente, en lenguaje C;

- El código MIPS32 generado por el compilador[3];
- Este enunciado.

7. Fechas

Fecha de vencimiento: martes 24/9.

Referencias

[1] GXemul, `http://gavare.se/gxemul/`.

[2] The NetBSD project, `http://www.netbsd.org/`.

[3] Row-major order (Wikipedia), `https://en.wikipedia.org/wiki/Row-major_order`.

[3] Por motivos prácticos, en la copia impresa sólo es necesario incluir la primera página del código assembly MIPS32 generado por el compilador.

2. Solución

2.1. Diseño

La solución propuesta para este trabajo comprende una serie de funciones que pueden ser clasificadas según una determinada división de responsabilidades:

- **Procesamiento:** Comprende las funciones dedicadas a lectura y procesamiento de datos para obtener las matrices a multiplicar.
- **Buffering:** El procesamiento requiere la utilización de un buffer dinámico que almacene los valores numéricos de punto flotante de doble precisión, que primero son almacenados como un array de `char`, y luego convertidos al tipo `double`.
- **Implementación de matrices:** Esta parte comprende la interfaz definida por el enunciado para la creación, impresión, multiplicación y destrucción de matrices.
- **Presentación:** Refiere al código generado para imprimir menús y mensajes de error.

2.1.1. Buffering

Este fragmento de la solución contiene las funciones `init_buffer` y `push`. La primer función inicializa el buffer. La segunda, agrega caracteres a un buffer ingresado como arugmento, manejando dinámicamente la memoria alocada por el buffer on demand.

2.1.2. Procesamiento

Esta parte cuenta con las siguientes funciones:

- `get_value`: Esta función escribe sobre un puntero pasado como argumento el valor de punto flotante obtenido, correspondiente a una 'palabra' leida por `stdin`. Esta misma función se utiliza para leer el primer valor entero que indica el tamaño de la matriz cuadrada. Para esto luego se castea el valor obtenido al tipo `size_t`.

  Para controlar y validar el input, la función retorna la cantidad de caracteres que leyó para obtener el valor.

- `read_matrix`: Aquí se invoca las veces necesarias a la función `get_value` para obtener los valores de cada matriz. En caso de llegar a whitespaces o símbolos no esperados, imprime un error que advierte que el formato del input es inválido. Las matrices las almacena en un puntero a `matrix_t`.

- `process_line`: Como bien lo indica el nombre, esta función procesa una línea de standard input. Lee el primer valor para obtener la dimensión de las matrices. Por cada matriz, invoca a la inicialización de las variables de tipo `matrix_t`, y luego a la función `read_matrix`.

  Por último llama a `matrix_multiply` para multiplicar las matrices y almacenar el resultado en un nuevo puntero a `matrix_t`, para luego imprimir esta última matriz.

2.1.3. Implementación de matrices

Esta sección escencialmente implementa las interfaces propuestas por el enunciado. En particular se destacan las siguiente implementaciones:

1. `create_matrix`: Esta función requiere alocar memoria para la estructura de `matrix_t`, y luego para el array de valores una vez conocido el tamaño de la matriz cuadrada.

   Por ello esta función realiza 2 memory allocation. Finalmente se inicializan los atributos de la matriz correspondientemente.

2. `destroy_matrix`: Como complemento de `create_matrix`, esta función debe liberar la memoria alocada tanto para el array de valores como para la estructura inicializada.

2.1.4. Presentación

Esta sección resuelve la mayor parte de las interacciones con el usuario del programa:

1. `argsHandler_parse_arguments`: Parseo de argumentos y ejecución de la rutina correspondiente.

2. `print_help, print_usage, print_options`:

   Impresión de menú de ayuda y detalle de las opciones de ejecución del programa.

2.1.5. Main function

La función `main` sencillamente invoca al parseo de argumentos para imprimir menús de ayuda u opciones según corresponda. Sino invoca a la rutina que lee el input por `stdin` y procesa las líneas.

2.2. Código

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

#define VERSION "1.0.0"
#define INIT_SIZE 2

/* ******* Buffer functions ******* */

char *init_buffer(size_t size)
{
    return malloc(size * sizeof(char));
}

void push(char *buffer, char value, size_t *currLength, size_t *
    bufferSize)
```

```
{
    if (*currLength + 1 > *bufferSize)
    {
        *bufferSize = *bufferSize * 2;
        buffer = realloc(buffer, *bufferSize * sizeof(char));
    }

    buffer[*currLength] = value;
    *currLength = *currLength + 1;
}

/* ******* Matrix implementation ******* */

typedef struct matrix
{
    size_t rows;
    size_t cols;
    double *array;
} matrix_t;

matrix_t *create_matrix(size_t rows, size_t cols)
{
    matrix_t *matrix = malloc(sizeof(matrix_t));
    double *array = malloc(sizeof(double) * cols * rows);
    matrix->array = array;
    matrix->cols = cols;
    matrix->rows = rows;
    return matrix;
}

int print_matrix(FILE *fp, matrix_t *m)
{
    int elementCount = m->rows * m->cols;
    for (int i = 0; i < elementCount; i++)
    {
        char *format = (i == elementCount - 1) ? "%.10g\n" : "%.10g␣";
        int res = fprintf(fp, format, m->array[i]);
        if (res < 0)
        {
            fprintf(stderr, "Error␣writing␣to␣file\n");
            return res;
        }
    }
    return elementCount;
}

matrix_t *matrix_multiply(matrix_t *matrix1, matrix_t *matrix2)
{
    matrix_t *result = create_matrix(matrix1->rows, matrix2->cols);

    for (int i = 0; i < matrix1->rows; i++)
    {
        for (int j = 0; j < matrix2->cols; j++)
        {
```

```
            double acum = 0;
            for (int k = 0; k < matrix1->cols; k++)
            {
                acum = acum + matrix1->array[matrix1->cols * i + k] *
                    matrix2->array[matrix2->cols * k + j];
            }
            result->array[result->cols * i + j] = acum;
        }
    }

    // print_matrix(stdout, result);

    return result;
}

void destroy_matrix(matrix_t *m)
{
    free(m->array);
    free(m);
}

/* ******* Value processing ******* */

int get_value(double *value_ptr, bool *eol, bool *eof)
{
    char c;
    bool stop = false;
    size_t length = 0;
    size_t bufferSize = INIT_SIZE;
    char *buffer = init_buffer(bufferSize);

    while (!stop)
    {
        c = (char)getchar();
        if (c == ' ' || c == '\n' || c == EOF)
        {
            *eol = (c == '\n');
            *eof = (c == EOF);
            stop = true;
        }
        else
        {
            push(buffer, c, &length, &bufferSize);
        }
    }
    if (length > 0)
    {
        *value_ptr = atof(buffer);
    }
    free(buffer);
    return length;
}

int read_matrix(matrix_t *matrix, int dim, bool *eol, bool *eof)
```

```
{
    double value;
    int res;

    for (int i = 0; i < dim * dim; i++)
    {
        res = get_value(&value, eol, eof);
        if (res == 0 && (*eol || *eof))
        {
            fprintf(stderr, "Invalid␣format,␣cannot␣read␣matrix\n");
            return -1;
        }
        else
        {
            matrix->array[i] = value;
        }
    }

    return 0;
}

enum LineEnding
{
    EndOfLine,
    EndOfFile,
    Error
};

enum LineEnding process_line()
{
    double value;
    int res;
    size_t dim;

    bool eol = false;
    bool eof = false;

    res = get_value(&value, &eol, &eof);

    if (eof)
    {
        return EndOfFile;
    }

    if (eol)
    {
        return EndOfLine;
    }

    if (res == 0)
    {
        fprintf(stderr, "Invalid␣format,␣cannot␣read␣matrix\n");
        return Error;
    }
```

```
    dim = (size_t)value;
    int response;

    matrix_t *matrix1 = create_matrix(dim, dim);

    response = read_matrix(matrix1, dim, &eol, &eof);

    if(response == -1) {
        destroy_matrix(matrix1);
        return EndOfLine;
    }

    matrix_t *matrix2 = create_matrix(dim, dim);
    response = read_matrix(matrix2, dim, &eol, &eof);

    if(response == -1) {
        destroy_matrix(matrix1);
        destroy_matrix(matrix2);
        return EndOfLine;
    }

    if (!eol && !eof)
    {
        destroy_matrix(matrix1);
        destroy_matrix(matrix2);
        fprintf(stderr, "Invalid␣format,␣cannot␣read␣matrix\n");
        return EndOfLine;
    }

    printf("first␣matrix\n");
    print_matrix(stdout, matrix1);

    printf("second␣matrix\n");
    print_matrix(stdout, matrix2);

    matrix_t *result = matrix_multiply(matrix1, matrix2);

    printf("result␣matrix\n");
    print_matrix(stdout, result);

    destroy_matrix(matrix1);
    destroy_matrix(matrix2);
    destroy_matrix(result);

    return EndOfLine;
}

static void print_usage(const char* src) {
    printf("%s\n\t%s%s\n\t%s%s\n\t%s%s\n", "Usage:",
           src, "␣-h",
           src, "␣-V",
           src, "␣<␣in_file␣>␣out_file");
}
```

```
static void print_options(const char* src) {
    printf("%s\n%s\n%s\n%s\n%s%s\t%s\n%s%s%s\n",
           "Options:",
          "\t-V, --version\tProgram version.",
          "\t-h, --help\tPrint help.",
          "Examples:",
          "\t", src, "tp0 < in.txt > out.txt",
          "\tcat in.txt ", src, " | tp0 > out.txt");
}

static void print_help(const char* src) {
    print_usage(src);
    print_options(src);
}

static void print_version(const char* src) {
    printf("%s: Version %s\n", src, VERSION);
}

int argsHandler_parse_arguments(const int argc, char* const argv[]) {
    int arg;
    int option_index = 0;
    const char* short_opt = "hV";
    static struct option long_options[] = {
           {"help", no_argument, 0, 'h'},
           {"version", no_argument, 0, 'V'},
           {0, 0, 0, 0 }
    };

    while ((arg = getopt_long(argc, argv, short_opt, long_options, &
        option_index)) != -1) {
       switch (arg) {
           case 'h':
               print_help(argv[0]);
               exit(EXIT_SUCCESS);
           case 'V':
               print_version(argv[0]);
               exit(EXIT_SUCCESS);
           case '?':
               exit(EXIT_FAILURE);
               break;
           default:
               break;
       }
    }

    // extern int optind
    if (optind < argc) {
       fprintf(stderr, "Opciones no reconocidas: ");
       while (optind < argc) {
           fprintf(stderr, "%s ", argv[optind++]);
       }
       fprintf(stderr, "\n");
```

```
        exit(EXIT_FAILURE);
    }

    return EXIT_SUCCESS;
}

int main(int argc, char* const argv[])
{
    argsHandler_parse_arguments(argc, argv);

    enum LineEnding result;

    do
    {
        result = process_line();
    } while (result == EndOfLine);

    if (result == Error)
    {
        return -1;
    }

    return 0;
}
```

2.3. Compilación del programa

Para compilar el programa se utiliza el makefile desde el directorio donde se descomprimió el entregable:

```
$ gmake all
cc -I. -c dynamic.c -o obj/dynamic.o -std=c99 -Wall -Werror -pedantic -
    pedantic-errors
cc obj/dynamic.o -o dynamic
```

3. Casos de prueba

3.0.1. Ejecución

Para automatizar la ejecución de los casos de prueba generamos una serie de scripts de bash:

- `generate_output.sh`: Esta función escribe sobre un puntero pasado como argumento el valor de punto flotante obtenido, correspondiente a una 'palabra' leida por `stdin`. Esta misma función se utiliza para leer el primer valor entero que indica el tamaño de la matriz cuadrada. Para esto luego se castea el valor obtenido al tipo `size_t`.

  Para controlar y validar el input, la función retorna la cantidad de caracteres que leyó para obtener el valor.

- `read_matrix`: Aquí se invoca las veces necesarias a la función `get_value` para obtener los valores de cada matriz. En caso de llegar a whitespaces o símbolos no esperados, imprime un error que advierte que el formato del input es inválido. Las matrices las almacena en un puntero a `matrix_t`.

Los casos de prueba elegidos se distribuyen en 3 grupos: DOS, UNIX y mixed. DOS y UNIX contienen los mismos casos, excepto que en DOS los casos tienen *line endings* con `'\r\n'` y en UNIX con `'\n'`.

En mixed se agrupan los casos neutros, que corresponden a pruebas con archivos vacíos o con ocurrencias de ambos tipos de *line endings*.

3.0.2. Simple.txt

Simple.txt es la prueba básica para evaluar el correcto funcionamiento de los programas. Su contenido es simplemente el siguiente.

```
One line
two lines
three lines
```

Por ejemplo, la ejecución del archivo `simple.txt` de la carpeta UNIX con el programa `unix2dos` es la siguiente:

```
$ ./unix2dos -i casos_prueba/UNIX/simple.txt | od -t c
0000000 O n e l i n e \r \n t w o l i
0000020 n e s \r \n t h r e e l i n e s
0000040 \r \n
0000042
```

Ejecución con valgrind:

```
$ valgrind --leak-check=yes ./unix2dos -i casos_prueba/UNIX/simple.txt -o
     nuevoarchivo.txt

==7662== Memcheck, a memory error detector
==7662== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==7662== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright
    info
==7662== Command: ./unix2dos -i casos_prueba/UNIX/simple.txt -o
    nuevoarchivo.txt
==7662==
==7662==
==7662== HEAP SUMMARY:
==7662==     in use at exit: 0 bytes in 0 blocks
==7662==   total heap usage: 4 allocs, 4 frees, 9,296 bytes allocated
==7662==
==7662== All heap blocks were freed -- no leaks are possible
==7662==
==7662== For counts of detected and suppressed errors, rerun with: -v
==7662== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Puede verse así que en la ejecución no se pierde memoria, y se llega al resultado esperado. En este caso los cortes de línea se reemplazan por `'\r\n'`.

3.0.3. Complex.txt y complex_long.txt

Los archivos complex y complex_long.txt prueban nuestros programas ante archivos con caracteres extraños. Complex_long.txt se caracteriza por tener líneas muy largas.

La siguiente es la ejecución de `dos2unix` con el archivo DOS/complex.txt.

```
$ ./dos2unix -i casos_prueba/DOS/complex.txt | od -t c
  0000000 304 252 302 264 304 207 k 302 210 302 222 303 270 K X y
  0000020 304 261 304 231 302 212 U 304 203 302 250 M 302 256 K 303
  0000040 250 303 233 302 254 303 244 302 212 304 237 u 303 250 M
  0000060 303 266 303 220 E b 303 225 302 270 304 206 302 254 302 201
  0000100 303 204 R 303 224 304 260 C b | 302 245 304 241 304 207
  0000120 L 304 257 303 236 303 225 s p 304 245 304 260 A o 303
  0000140 224 W { o 303 202 303 222 304 267 304 241 302 200 G 303
  0000160 235 w 302 256 p k L i c 304 231 302 221 p s 303
  0000200 272 303 232 304 217 304 241 304 202 N 302 252 303 256 302 202
  0000220 O 303 253 J d 302 241 _ R 303 226 303 202 a 303 236
  0000240 304 255 303 257 304 272 M 304 233 304 252 304 264 304 220 \
  0000260 302 242 304 221 303 261 302 267 j 302 222 304 242 302 213 304
  0000300 212 303 263 302 212 O ^ O 303 251 302 213 304 213 h 302
  0000320 270 304 236 Y 302 216 304 267 302 214 304 256 304 273 [ 302
  0000340 246 303 224 302 234 { 303 233 304 255 d i a p 302 272
  0000360 302 252 304 223 l 304 266 Z 304 205 302 225 303 220 302 222
  0000400 303 267 302 211 304 211 B 304 206 302 220 302 231 h G 304
  0000420 272 p D k 303 221 302 245 302 233 T 304 215 303 204 g
  0000440 303 241 302 270 302 247 N 304 273 } 302 232 k 303 201 177
  0000460 Y 304 270 \ 304 256 303 220 304 275 304 257 304 215 304 250
  0000500 304 241 m 304 230 303 243 O 302 227 304 264 302 276 302 221
  0000520 302 254 303 204 Y 302 224 303 212 302 264 q 304 222 304 273
  0000540 E L 302 273 303 231 T G 304 203 304 213 W 304 255 t
  0000560 y 304 214 e 304 255 303 221 303 274 304 220 302 212 303 203
  0000600 304 233 | 302 245 304 200 p 304 231 302 235 302 201 H
  0000620 304 260 303 267 304 200 P 304 257 303 233 304 221 304 200 ]
  0000640 n \n ^ 303 235 303 272 302 206 304 211 302 244 F 304 236
  0000660 303 267 304 200 302 234 304 271 302 233 302 246 } 304 255 303
  0000700 235 Y N 302 226 302 206 304 244 177 302 270 303 250 302 264
  0000720 303 254 302 227 303 230 i 304 246 303 257 303 265 304 207 304
  0000740 264 304 224 304 217 303 212 303 247 c 304 223 h 303 211 303
  0000760 201 302 230 303 272 303 211 303 257 a 302 211 Y 304 220 303
  0001000 241 302 265 B 302 221 303 233 304 203 303 266 302 241 l 304
  0001020 263 302 200 302 271 302 273 303 217 302 225 304 200 j 302 203
  0001040 X Y D 303 270 304 217 302 252 302 230 303 257 303 244 ^
  0001060 302 206 304 265 304 200 F 304 244 303 242 j 304 237 ‘ 304
  0001100 252 P 302 247 302 225 304 214 302 252 303 213 w 304 262
  0001120 h 304 221 303 204 302 253 304 227 i 303 247 304 235 W 304
  0001140 221 302 255 302 250 302 226 304 226 302 271 O 303 246 302 251
  0001160 302 251 302 275 303 232 303 235 302 272 304 234 304 265 302 270
  0001200 302 254 302 212 304 225 304 204 302 210 303 212 303 207 302 251
  0001220 s 302 242 c 302 224 304 265 ‘ 302 242 p 302 277 Z 302
  0001240 273 304 224 304 233 303 266 303 257 304 273 303 261 303 257 303
  0001260 261 303 263 _ 304 244 302 222 302 244 302 254 302 253 303 252
  0001300 E 304 273 d 303 247 302 231 y e 304 261 303 243 302
  0001320 211 X ] n 304 266 \n H 302 261 k 302 265 302 234 a
  0001340 302 261 302 230 c M 302 204 Q h 302 246 303 201 302 256
```

```
0001360 k 304 204 302 227 302 241 303 265 304 272 | 304 252 302 214
0001400 303 227 304 261 303 263 303 274 R n 303 206 302 277 ] 304
0001420 233 303 241 V 304 253 303 207 303 210 B 303 223 303 252 302
0001440 265 303 240 L 304 225 304 274 304 203 303 261 303 244 303 274
0001460 j 302 205 303 262 304 222 304 243 o q 304 201 304 221 304
0001500 223 302 235 302 235 304 202 302 262 304 251 304 202 302 211 303
0001520 274 ‘ 304 236 304 264 303 277 l 303 205 303 232 302 215 L
0001540 m 303 251 D 302 220 303 215 304 244 303 217 303 214 A 303
0001560 276 302 265 303 245 303 207 p 302 256 303 214 302 264 303 223
0001600 302 245 302 225 g 302 272 304 210 k 304 271 303 250 s 304
0001620 230 303 261 302 277 303 251 304 263 303 256 304 207 304 267 D
0001640 304 250 B 303 224 302 243 302 276 304 235 304 200 ^ 302 223
0001660 302 231 304 252 E 302 201 304 223 303 273 304 255 302 244 303
0001700 254 302 230 303 256 304 221 304 204 302 202 304 222 n j 302
0001720 262 C 303 223 302 271 304 213 304 202 304 213 303 201 303 220
0001740 304 243 302 201 304 266 304 250 c 302 243 303 245 303 261 304
0001760 221 303 202 j 302 277 303 277 303 245 302 211 302 276 303 232
0002000 C ] 303 275 304 260 304 253 303 251 304 274 302 200 304 236
0002020 303 231 304 213 302 215 303 212 S 303 266 } A 302 247 303
0002040 237 a 303 217 303 254 303 213 ] 302 274 304 227 D 302 200
0002060 302 230 304 254 304 213 302 246 { 302 212 303 245 302 250 303
0002100 267 303 202 304 216 303 252 o ^ r 302 224 302 275 j 303
0002120 242 304 260 303 220 Q 302 252 J 302 244 302 241 R 303 204
0002140 V 303 263 304 233 303 223 304 244 303 260 304 220 302 255 F
0002160 304 270 302 262 303 226 303 230 304 273 303 215 303 271 304 261
0002200 302 247 303 203 303 222 302 203 303 231 303 265 302 276 303 214
0002220 303 240 J 302 263 302 274 304 266 303 236 304 250 304 260 303
0002240 235 302 261 j 302 207 O w 302 251 302 204 303 231 302 220
0002260 302 235 i 304 234 303 271 | \n
0002271
```

En esta ejecución se muestra la varidedad de caracteres utilizados y el resultado de reemplazar los cortes de línea por caracteres ‘`\n`’. Los caracteres extraños no afectan la normal ejecución del programa.

3.0.4. Big.txt

El archivo Big.txt corresponde a un documento de texto normal, escrito mayormente en inglés, pero sumamente grande (pesa 6.6Mb). La ejecución con valgrind de este archivo muestra que la memoria alocada por el programa es la misma indistintamente del tamaño del archivo, dado que se utiliza sólamente un puntero para recorrerlo, tanto en `unix2dos` como en `dos2unix`. Ejecución con valgrind:

```
$ valgrind --leak-check=yes ./unix2dos -i casos_prueba/UNIX/big.txt
  ==5509== Memcheck, a memory error detector
  ==5509== Copyright (C) 2002-2017, and GNU GPL'd,␣by␣Julian␣Seward␣et␣al
      .
␣␣==5509==␣Using␣Valgrind-3.13.0␣and␣LibVEX;␣rerun␣with␣-h␣for␣copyright␣
    info
␣␣==5509==␣Command:␣./unix2dos␣-i␣casos_prueba/UNIX/big.txt␣-o␣
    nuevoarchivo.txt
␣␣==5509==
␣␣==5509==
```

```
␣␣==5509==␣HEAP␣SUMMARY:
␣␣==5509==␣␣␣␣␣␣in␣use␣at␣exit:␣0␣bytes␣in␣0␣blocks
␣␣==5509==␣␣␣␣total␣heap␣usage:␣4␣allocs,␣4␣frees,␣9,296␣bytes␣allocated
␣␣==5509==
␣␣==5509==␣All␣heap␣blocks␣were␣freed␣--␣no␣leaks␣are␣possible
␣␣==5509==
␣␣==5509==␣For␣counts␣of␣detected␣and␣suppressed␣errors,␣rerun␣with:␣-v
␣␣==5509==␣ERROR␣SUMMARY:␣0␣errors␣from␣0␣contexts␣(suppressed:␣0␣from␣0)
```

3.0.5. Empty.txt y oneliner.txt

Estos archivos constituyen básicamente un archivo vacío y un archivo de una línea respectivamente. Son casos utilizados para verificar la normal ejecución del programa en los casos borde de un archivo vacío o con tan sólo una línea. La siguiente es por ejemplo la ejecución de `dos2unix` con oneliner.txt:

```
$ ./dos2unix -i casos_prueba/mixed/oneliner.txt | od -t c

0000000 h e l l o t h i s i s a
0000020 l i n e
0000024
```

La ejecución resulta normal y naturalmente no debe reemplazarse ningún caracter.

3.0.6. Mixed.txt

Mixed.txt es un archivo que contiene ambos tipos de fin de línea. A continuación se puede ver el contenido del archivo:

```
$ cat casos_prueba/mixed/mixed.txt | od -t c
0000000 \r \n \n \r \n \r \n \n \n \r \r \n \n \n \r \r
0000020 \r \r \n \r \n
0000025
```

La siguiente es la ejecución de `dos2unix` con este archivo:

```
$ ./dos2unix -i casos_prueba/mixed/mixed.txt | od -t c
0000000 \n \n \n \n \n \n \r \n \n \n \r \r \r \n \n
0000017
```

Como puede verse las sucesiones '`\r\n`' son reemplazadas por '`\n`', mientras que algunos caracteres '`\r`' sobrantes son ignorados al no coincidir con la secuencia que debe reemplazarse por fines de línea estilo UNIX.

4. Código MIPS

4.1. Código assembly generado para *common*

4.2. Código assembly generado para *unix2dos*

4.3. Código assembly generado para *dos2unix*

5. Conclusiones

A modo de conclusión podemos decir que en este trabajo práctico pudimos configurar y familiarizarnos con el entorno de trabajo de la materia, que emula una computadora con procesador MIPS. También pudimos observar algunas de las diferencias entre programar para un sistema operativo linux y para el netBSD que corre emulado. Además adquirimos conocimiento respecto a otras herramientas útiles como ssh, scp, diff y od entre otras.