

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 descrito 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 cada línea describe completamente cada par de matrices a multiplicar, según el siguiente formato:

```
N a1,1 a1,2 ... aN,N b1,1 b1,2 ... bN,N
```

La línea anterior representa a las matrices A y B , de $N \times N$. Los elementos de la matriz A son los $a_{x,y}$, siendo x e y los índices de fila y columna respectivamente¹. 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 carácter `\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 c1,1 c1,2 ... cN,N
```

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

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

²Ver 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 compilador³;
- 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.

³Por 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 primera función inicializa el buffer. La segunda, agrega caracteres a un buffer ingresado como argumento, 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' leída 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 esencialmente 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 <string.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 calloc(size, size * sizeof(char));
}

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



```

{
    if (*currLength >= *bufferSize)
    {
        char *res;
        *bufferSize = *bufferSize * 2;
        printf("about to realloc to size %lu\n", *bufferSize * sizeof(
            char));
        res = realloc(buffer, *bufferSize * sizeof(char));
        res[*currLength] = value;
        res[*currLength + 1] = '\0';
        *currLength = *currLength + 1;
        return res;
    }
    buffer[*currLength] = value;
    *currLength = *currLength + 1;
    return buffer;
}

/* ***** 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;
        }
    }

    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 == '\u' || c == '\n' || c == EOF)
        {
            *eol = (c == '\n');
            *eof = (c == EOF);
            stop = true;
        }
        else
        {
            char* new_buffer = push(buffer, c, &length, &bufferSize);
            buffer = new_buffer;
        }
    }
    if (length > 0)
    {
        // printf("buffer value is %s \n", buffer);
    }
}

```

```

        // char string_value[length];
        // strncpy(string_value, buffer, length - 1);
        // string_value[length] = '\0';
        // *value_ptr = atof(buffer);
        sscanf(buffer, "%lg", value_ptr);
    }
    free(buffer);
    return length;
}

int read_matrix(matrix_t *matrix, int dim, bool *eol, bool *eof) {
    int res;
    double value;

    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);
        return EndOfLine;
    }

    matrix_t *result = matrix_multiply(matrix1, matrix2);

    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\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()
{
    printf("Version_ %s\n", 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();
                exit(EXIT_SUCCESS);
            case '?':
                exit(EXIT_FAILURE);
                break;
            default:
                break;
        }
    }
}

```

```

    }
}

// extern int optind
if (optind < argc)
{
    fprintf(stderr, "Unrecognized options:");
    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. Pruebas

3.0.1. Ejecución

Para la ejecución de las pruebas generamos un script llamado `run_tests.sh` que se encarga de, para cada caso generado, ejecutar el programa y escribir el output en un archivo. Finalmente, se comparan los outputs contra archivos que contienen los resultados esperados, en la carpeta `results/`.

3.1. Casos de prueba

Para probar nuestro programa generamos una serie de casos de prueba:

- `big_matrix_ints.txt`: Matriz de 100x100 con valores enteros
- `big_matrix_doubles.txt`: Matriz de 100x100 con valores de punto flotante
- `exponential_doubles_matrix.txt`: Matriz pequeña con valores de punto flotante en notación exponencial.
- `invalid_input.txt`: Prueba con input invalido.
- `simple_multiline.txt`: Prueba de input con varias líneas.
- `single_line.txt`: Prueba simple de una sola línea.
- `small_double_matrix.txt`: Prueba simple de una matriz 3x3 de doubles.
- `small_int_matrix.txt`: Prueba simple de una matriz 3x3 de integers.

La ejecución de los casos se realiza de la siguiente manera:

```
$ ./run_tests.sh
```

Este script de ejecución valida también la impresión del menú y de la versión del programa.

4. Código MIPS

4.1. Código assembly generado para *dynamic.c*

```
.file 1 "dynamic.c"
.section .mdebug.abi32
.previous
.abicalls
.text
.align 2
.globl init_buffer
.ent init_buffer
init_buffer:
.frame $fp,40,$ra # vars= 0, regs= 3/0, args= 16, extra= 8
.mask 0xd0000000,-8
.fmask 0x00000000,0
.set noreorder
```

```

.cpload $t9
.set reorder
subu $sp,$sp,40
.cprestore 16
sw $ra,32($sp)
sw $fp,28($sp)
sw $gp,24($sp)
move $fp,$sp
sw $a0,40($fp)
lw $a0,40($fp)
la $t9,malloc
jal $ra,$t9
move $sp,$fp
lw $ra,32($sp)
lw $fp,28($sp)
addu $sp,$sp,40
j $ra
.end init_buffer
.size init_buffer, .-init_buffer
.align 2
.globl push
.ent push
push:
.frame $fp,48,$ra # vars= 8, regs= 3/0, args= 16, extra= 8
.mask 0xd0000000,-8
.fmask 0x00000000,0
.set noreorder
.cpload $t9
.set reorder
subu $sp,$sp,48
.cprestore 16
sw $ra,40($sp)
sw $fp,36($sp)
sw $gp,32($sp)
move $fp,$sp
sw $a0,48($fp)
move $v0,$a1
sw $a2,56($fp)
sw $a3,60($fp)
sb $v0,24($fp)
lw $v0,56($fp)
lw $v0,0($v0)
addu $v1,$v0,1
lw $v0,60($fp)
lw $v0,0($v0)
sltu $v0,$v0,$v1
beq $v0,$zero,$L19
lw $v1,60($fp)
lw $v0,60($fp)
lw $v0,0($v0)
sll $v0,$v0,1
sw $v0,0($v1)
lw $v0,60($fp)
lw $a0,48($fp)

```



```

    lw $a1,0($v0)
    la $t9,realloc
    jal $ra,$t9
    sw $v0,48($fp)
$L19:
    lw $v0,56($fp)
    lw $v1,48($fp)
    lw $v0,0($v0)
    addu $v1,$v1,$v0
    lbu $v0,24($fp)
    sb $v0,0($v1)
    lw $v1,56($fp)
    lw $v0,56($fp)
    lw $v0,0($v0)
    addu $v0,$v0,1
    sw $v0,0($v1)
    move $sp,$fp
    lw $ra,40($sp)
    lw $fp,36($sp)
    addu $sp,$sp,48
    j $ra
.end push
.size push, .-push
.align 2
.globl create_matrix
.ent create_matrix
create_matrix:
    .frame $fp,48,$ra # vars= 8, regs= 3/0, args= 16, extra= 8
    .mask 0xd0000000,-8
    .fmask 0x00000000,0
    .set noreorder
    .cpld $t9
    .set reorder
    subu $sp,$sp,48
    .cpstore 16
    sw $ra,40($sp)
    sw $fp,36($sp)
    sw $gp,32($sp)
    move $fp,$sp
    sw $a0,48($fp)
    sw $a1,52($fp)
    li $a0,12 # 0xc
    la $t9,malloc
    jal $ra,$t9
    sw $v0,24($fp)
    lw $v1,52($fp)
    lw $v0,48($fp)
    mult $v1,$v0
    mflo $v0
    sll $v0,$v0,3
    move $a0,$v0
    la $t9,malloc
    jal $ra,$t9
    sw $v0,28($fp)

```

```

lw $v1,24($fp)
lw $v0,28($fp)
sw $v0,8($v1)
lw $v1,24($fp)
lw $v0,52($fp)
sw $v0,4($v1)
lw $v1,24($fp)
lw $v0,48($fp)
sw $v0,0($v1)
lw $v0,24($fp)
move $sp,$fp
lw $ra,40($sp)
lw $fp,36($sp)
addu $sp,$sp,48
j $ra
.end create_matrix
.size create_matrix, .-create_matrix
.rdata
.align 2
$LC0:
.ascii "%.10g\n\000"
.align 2
$LC1:
.ascii "%.10g\000"
.align 2
$LC2:
.ascii "Error writing to file\n\000"
.text
.align 2
.globl print_matrix
.ent print_matrix
print_matrix:
.frame $fp,64,$ra # vars= 24, regs= 3/0, args= 16, extra= 8
.mask 0xd0000000,-8
.fmask 0x00000000,0
.set noreorder
.cpload $t9
.set reorder
subu $sp,$sp,64
.cprestore 16
sw $ra,56($sp)
sw $fp,52($sp)
sw $gp,48($sp)
move $fp,$sp
sw $a0,64($fp)
sw $a1,68($fp)
lw $v0,68($fp)
lw $v1,68($fp)
lw $a0,0($v0)
lw $v0,4($v1)
mult $a0,$v0
mflo $v0
sw $v0,24($fp)
sw $zero,28($fp)

```

```

$L22:
    lw $v0,28($fp)
    lw $v1,24($fp)
    slt $v0,$v0,$v1
    bne $v0,$zero,$L25
    b $L23
$L25:
    lw $v0,24($fp)
    addu $v1,$v0,-1
    lw $v0,28($fp)
    bne $v0,$v1,$L26
    la $v0,$LC0
    sw $v0,44($fp)
    b $L27
$L26:
    la $v0,$LC1
    sw $v0,44($fp)
$L27:
    lw $v0,44($fp)
    sw $v0,32($fp)
    lw $a0,68($fp)
    lw $v0,28($fp)
    sll $v1,$v0,3
    lw $v0,8($a0)
    addu $v0,$v1,$v0
    lw $a0,64($fp)
    lw $a1,32($fp)
    lw $a2,0($v0)
    lw $a3,4($v0)
    la $t9,fprintf
    jal $ra,$t9
    sw $v0,36($fp)
    lw $v0,36($fp)
    bgez $v0,$L24
    la $a0,___sF+176
    la $a1,$LC2
    la $t9,fprintf
    jal $ra,$t9
    lw $v0,36($fp)
    sw $v0,40($fp)
    b $L21
$L24:
    lw $v0,28($fp)
    addu $v0,$v0,1
    sw $v0,28($fp)
    b $L22
$L23:
    lw $v0,24($fp)
    sw $v0,40($fp)
$L21:
    lw $v0,40($fp)
    move $sp,$fp
    lw $ra,56($sp)
    lw $fp,52($sp)

```

```

    addu $sp,$sp,64
    j $ra
    .end print_matrix
    .size print_matrix, .-print_matrix
    .align 2
    .globl matrix_multiply
    .ent matrix_multiply
matrix_multiply:
    .frame $fp,72,$ra # vars= 32, regs= 3/0, args= 16, extra= 8
    .mask 0xd0000000,-8
    .fmask 0x00000000,0
    .set noreorder
    .cpload $t9
    .set reorder
    subu $sp,$sp,72
    .cprestore 16
    sw $ra,64($sp)
    sw $fp,60($sp)
    sw $gp,56($sp)
    move $fp,$sp
    sw $a0,72($fp)
    sw $a1,76($fp)
    lw $v0,72($fp)
    lw $v1,76($fp)
    lw $a0,0($v0)
    lw $a1,4($v1)
    la $t9,create_matrix
    jal $ra,$t9
    sw $v0,24($fp)
    sw $zero,28($fp)
$L30:
    lw $v0,72($fp)
    lw $v1,28($fp)
    lw $v0,0($v0)
    sltu $v0,$v1,$v0
    bne $v0,$zero,$L33
    b $L31
$L33:
    sw $zero,32($fp)
$L34:
    lw $v0,76($fp)
    lw $v1,32($fp)
    lw $v0,4($v0)
    sltu $v0,$v1,$v0
    bne $v0,$zero,$L37
    b $L32
$L37:
    sw $zero,40($fp)
    sw $zero,44($fp)
    sw $zero,48($fp)
$L38:
    lw $v0,72($fp)
    lw $v1,48($fp)
    lw $v0,4($v0)

```

```

    sltu $v0,$v1,$v0
    bne $v0,$zero,$L41
    b $L39
$L41:
    lw $a0,72($fp)
    lw $v0,72($fp)
    lw $v1,4($v0)
    lw $v0,28($fp)
    mult $v1,$v0
    mflo $v1
    lw $v0,48($fp)
    addu $v0,$v1,$v0
    sll $v1,$v0,3
    lw $v0,8($a0)
    addu $a1,$v1,$v0
    lw $a0,76($fp)
    lw $v0,76($fp)
    lw $v1,4($v0)
    lw $v0,48($fp)
    mult $v1,$v0
    mflo $v1
    lw $v0,32($fp)
    addu $v0,$v1,$v0
    sll $v1,$v0,3
    lw $v0,8($a0)
    addu $v0,$v1,$v0
    l.d $f2,0($a1)
    l.d $f0,0($v0)
    mul.d $f2,$f2,$f0
    l.d $f0,40($fp)
    add.d $f0,$f0,$f2
    s.d $f0,40($fp)
    lw $v0,48($fp)
    addu $v0,$v0,1
    sw $v0,48($fp)
    b $L38
$L39:
    lw $a0,24($fp)
    lw $v0,24($fp)
    lw $v1,4($v0)
    lw $v0,28($fp)
    mult $v1,$v0
    mflo $v1
    lw $v0,32($fp)
    addu $v0,$v1,$v0
    sll $v1,$v0,3
    lw $v0,8($a0)
    addu $v0,$v1,$v0
    l.d $f0,40($fp)
    s.d $f0,0($v0)
    lw $v0,32($fp)
    addu $v0,$v0,1
    sw $v0,32($fp)
    b $L34

```

```

$L32:
    lw $v0,28($fp)
    addu $v0,$v0,1
    sw $v0,28($fp)
    b $L30
$L31:
    lw $v0,24($fp)
    move $sp,$fp
    lw $ra,64($sp)
    lw $fp,60($sp)
    addu $sp,$sp,72
    j $ra
    .end matrix_multiply
    .size matrix_multiply, .-matrix_multiply
    .align 2
    .globl destroy_matrix
    .ent destroy_matrix
destroy_matrix:
    .frame $fp,40,$ra # vars= 0, regs= 3/0, args= 16, extra= 8
    .mask 0xd0000000,-8
    .fmask 0x00000000,0
    .set noreorder
    .cpld $t9
    .set reorder
    subu $sp,$sp,40
    .cpstore 16
    sw $ra,32($sp)
    sw $fp,28($sp)
    sw $gp,24($sp)
    move $fp,$sp
    sw $a0,40($fp)
    lw $v0,40($fp)
    lw $a0,8($v0)
    la $t9,free
    jal $ra,$t9
    lw $a0,40($fp)
    la $t9,free
    jal $ra,$t9
    move $sp,$fp
    lw $ra,32($sp)
    lw $fp,28($sp)
    addu $sp,$sp,40
    j $ra
    .end destroy_matrix
    .size destroy_matrix, .-destroy_matrix
    .align 2
    .globl get_value
    .ent get_value
get_value:
    .frame $fp,64,$ra # vars= 24, regs= 3/0, args= 16, extra= 8
    .mask 0xd0000000,-8
    .fmask 0x00000000,0
    .set noreorder
    .cpld $t9

```

```

.set reorder
subu $sp,$sp,64
.cprestore 16
sw $ra,56($sp)
sw $fp,52($sp)
sw $gp,48($sp)
move $fp,$sp
sw $a0,64($fp)
sw $a1,68($fp)
sw $a2,72($fp)
sb $zero,25($fp)
sw $zero,28($fp)
li $v0,2    # 0x2
sw $v0,32($fp)
lw $a0,32($fp)
la $t9,init_buffer
jal $ra,$t9
sw $v0,36($fp)
$L44:
lbu $v0,25($fp)
beq $v0,$zero,$L46
b $L45
$L46:
lw $v0,__$sF+4
addu $v0,$v0,-1
sw $v0,__$sF+4
bgez $v0,$L47
la $a0,__$sF
la $t9,__$srget
jal $ra,$t9
sb $v0,40($fp)
b $L48
$L47:
la $v0,__$sF
lw $v1,0($v0)
move $a0,$v1
lbu $a0,0($a0)
sb $a0,40($fp)
addu $v1,$v1,1
sw $v1,0($v0)
$L48:
lbu $v0,40($fp)
sb $v0,24($fp)
lb $v1,24($fp)
li $v0,32    # 0x20
beq $v1,$v0,$L50
lb $v1,24($fp)
li $v0,10    # 0xa
beq $v1,$v0,$L50
lb $v1,24($fp)
li $v0,-1    # 0xffffffffffffffff
beq $v1,$v0,$L50
b $L49
$L50:

```

```

    lw $v1,68($fp)
    lb $v0,24($fp)
    xori $v0,$v0,0xa
    sltu $v0,$v0,1
    sb $v0,0($v1)
    lw $a0,72($fp)
    lb $v1,24($fp)
    li $v0,-1 # 0xffffffffffffffff
    xor $v0,$v1,$v0
    sltu $v0,$v0,1
    sb $v0,0($a0)
    li $v0,1 # 0x1
    sb $v0,25($fp)
    b $L44
$L49:
    lb $v0,24($fp)
    addu $v1,$fp,28
    addu $a3,$fp,32
    lw $a0,36($fp)
    move $a1,$v0
    move $a2,$v1
    la $t9,push
    jal $ra,$t9
    b $L44
$L45:
    lw $v0,28($fp)
    beq $v0,$zero,$L52
    lw $a0,36($fp)
    la $t9,atof
    jal $ra,$t9
    lw $v0,64($fp)
    s.d $f0,0($v0)
$L52:
    lw $a0,36($fp)
    la $t9,free
    jal $ra,$t9
    lw $v0,28($fp)
    move $sp,$fp
    lw $ra,56($sp)
    lw $fp,52($sp)
    addu $sp,$sp,64
    j $ra
    .end get_value
    .size get_value, .-get_value
    .rdata
    .align 2
$LC3:
    .ascii "Invalid format, cannot read matrix\n\000"
    .text
    .align 2
    .globl read_matrix
    .ent read_matrix
read_matrix:
    .frame $fp,64,$ra # vars= 24, regs= 3/0, args= 16, extra= 8

```



```

.mask 0xd0000000,-8
.fmask 0x00000000,0
.set noreorder
.cpload $t9
.set reorder
subu $sp,$sp,64
.cprestore 16
sw $ra,56($sp)
sw $fp,52($sp)
sw $gp,48($sp)
move $fp,$sp
sw $a0,64($fp)
sw $a1,68($fp)
sw $a2,72($fp)
sw $a3,76($fp)
sw $zero,36($fp)
$L54:
lw $v1,68($fp)
lw $v0,68($fp)
mult $v1,$v0
mflo $v1
lw $v0,36($fp)
slt $v0,$v0,$v1
bne $v0,$zero,$L57
b $L55
$L57:
addu $a0,$fp,24
lw $a1,72($fp)
lw $a2,76($fp)
la $t9,get_value
jal $ra,$t9
sw $v0,32($fp)
lw $v0,32($fp)
bne $v0,$zero,$L58
lw $v0,72($fp)
lbu $v0,0($v0)
bne $v0,$zero,$L59
lw $v0,76($fp)
lbu $v0,0($v0)
bne $v0,$zero,$L59
b $L58
$L59:
la $a0, __sF+176
la $a1,$LC3
la $t9,fprintf
jal $ra,$t9
li $v0,-1 # 0xffffffffffffffff
sw $v0,40($fp)
b $L53
$L58:
lw $a0,64($fp)
lw $v0,36($fp)
sll $v1,$v0,3
lw $v0,8($a0)

```

```

    addu $v0,$v1,$v0
    l.d $f0,24($fp)
    s.d $f0,0($v0)
    lw $v0,36($fp)
    addu $v0,$v0,1
    sw $v0,36($fp)
    b $L54
$L55:
    sw $zero,40($fp)
$L53:
    lw $v0,40($fp)
    move $sp,$fp
    lw $ra,56($sp)
    lw $fp,52($sp)
    addu $sp,$sp,64
    j $ra
    .end read_matrix
    .size read_matrix, .-read_matrix
    .rdata
    .align 3
$L4:
    .word 0
    .word 1105199104
    .text
    .align 2
    .globl process_line
    .ent process_line
process_line:
    .frame $fp,104,$ra # vars= 64, regs= 3/0, args= 16, extra= 8
    .mask 0xd0000000,-8
    .fmask 0x00000000,0
    .set noreorder
    .cpld $t9
    .set reorder
    subu $sp,$sp,104
    .cpstore 16
    sw $ra,96($sp)
    sw $fp,92($sp)
    sw $gp,88($sp)
    move $fp,$sp
    sb $zero,40($fp)
    sb $zero,41($fp)
    addu $v0,$fp,40
    addu $v1,$fp,41
    addu $a0,$fp,24
    move $a1,$v0
    move $a2,$v1
    la $t9,get_value
    jal $ra,$t9
    sw $v0,32($fp)
    lbu $v0,41($fp)
    beq $v0,$zero,$L62
    li $v0,1 # 0x1
    sw $v0,60($fp)

```

```

    b $L61
$L62:
    lbu $v0,40($fp)
    beq $v0,$zero,$L63
    sw $zero,60($fp)
    b $L61
$L63:
    lw $v0,32($fp)
    bne $v0,$zero,$L64
    la $a0, __sF+176
    la $a1,$LC3
    la $t9,fprintf
    jal $ra,$t9
    li $v1,2    # 0x2
    sw $v1,60($fp)
    b $L61
$L64:
    l.d $f0,24($fp)
    s.d $f0,64($fp)
    l.d $f2,$LC4
    s.d $f2,80($fp)
    l.d $f4,64($fp)
    l.d $f0,80($fp)
    c.le.d $f0,$f4
    bc1t $L65
    l.d $f0,64($fp)
    trunc.w.d $f0,$f0,$v0
    s.s $f0,72($fp)
    b $L66
$L65:
    l.d $f2,64($fp)
    l.d $f4,80($fp)
    sub.d $f0,$f2,$f4
    li $v0,-2147483648    # 0xffffffff80000000
    trunc.w.d $f2,$f0,$a0
    s.s $f2,72($fp)
    lw $v1,72($fp)
    or $v1,$v1,$v0
    sw $v1,72($fp)
$L66:
    lw $a0,72($fp)
    sw $a0,36($fp)
    lw $a0,36($fp)
    lw $a1,36($fp)
    la $t9,create_matrix
    jal $ra,$t9
    sw $v0,48($fp)
    addu $v0,$fp,40
    addu $v1,$fp,41
    lw $a0,48($fp)
    lw $a1,36($fp)
    move $a2,$v0
    move $a3,$v1
    la $t9,read_matrix

```

```

jal $ra,$t9
sw $v0,44($fp)
lw $v1,44($fp)
li $v0,-1 # 0xffffffffffffffff
bne $v1,$v0,$L67
lw $a0,48($fp)
la $t9,destroy_matrix
jal $ra,$t9
sw $zero,60($fp)
b $L61
$L67:
lw $a0,36($fp)
lw $a1,36($fp)
la $t9,create_matrix
jal $ra,$t9
sw $v0,52($fp)
addu $v0,$fp,40
addu $v1,$fp,41
lw $a0,52($fp)
lw $a1,36($fp)
move $a2,$v0
move $a3,$v1
la $t9,read_matrix
jal $ra,$t9
sw $v0,44($fp)
lw $v1,44($fp)
li $v0,-1 # 0xffffffffffffffff
bne $v1,$v0,$L68
lw $a0,48($fp)
la $t9,destroy_matrix
jal $ra,$t9
lw $a0,52($fp)
la $t9,destroy_matrix
jal $ra,$t9
sw $zero,60($fp)
b $L61
$L68:
lbu $v0,40($fp)
bne $v0,$zero,$L69
lbu $v0,41($fp)
bne $v0,$zero,$L69
lw $a0,48($fp)
la $t9,destroy_matrix
jal $ra,$t9
lw $a0,52($fp)
la $t9,destroy_matrix
jal $ra,$t9
sw $zero,60($fp)
b $L61
$L69:
lw $a0,48($fp)
lw $a1,52($fp)
la $t9,matrix_multiply
jal $ra,$t9

```

```

sw $v0,56($fp)
la $a0, __sF+88
lw $a1,56($fp)
la $t9,print_matrix
jal $ra,$t9
lw $a0,48($fp)
la $t9,destroy_matrix
jal $ra,$t9
lw $a0,52($fp)
la $t9,destroy_matrix
jal $ra,$t9
lw $a0,56($fp)
la $t9,destroy_matrix
jal $ra,$t9
sw $zero,60($fp)
$L61:
lw $v0,60($fp)
move $sp,$fp
lw $ra,96($sp)
lw $fp,92($sp)
addu $sp,$sp,104
j $ra
.end process_line
.size process_line, .-process_line
.rdata
.align 2
$LC8:
.ascii "_-V\000"
.align 2
$LC9:
.ascii "<_in_file_>_out_file\000"
.align 2
$LC5:
.ascii "%s\n"
.ascii "\t%s%s\n"
.ascii "\t%s%s\n"
.ascii "\t%s%s\n\000"
.align 2
$LC6:
.ascii "Usage:\000"
.align 2
$LC7:
.ascii "_-h\000"
.text
.align 2
.ent print_usage
print_usage:
.frame $fp,56,$ra # vars= 0, regs= 3/0, args= 32, extra= 8
.mask 0xd0000000,-8
.fmask 0x00000000,0
.set noreorder
.cpload $t9
.set reorder
subu $sp,$sp,56

```

```

.cprestore 32
sw $ra,48($sp)
sw $fp,44($sp)
sw $gp,40($sp)
move $fp,$sp
sw $a0,56($fp)
lw $v0,56($fp)
sw $v0,16($sp)
la $v0,$LC8
sw $v0,20($sp)
lw $v0,56($fp)
sw $v0,24($sp)
la $v0,$LC9
sw $v0,28($sp)
la $a0,$LC5
la $a1,$LC6
lw $a2,56($fp)
la $a3,$LC7
la $t9,printf
jal $ra,$t9
move $sp,$fp
lw $ra,48($sp)
lw $fp,44($sp)
addu $sp,$sp,56
j $ra
.end print_usage
.size print_usage, .-print_usage
.rdata
.align 2
$LC14:
.ascii "Examples:\000"
.align 2
$LC15:
.ascii "\t\000"
.align 2
$LC16:
.ascii "tp0_\tin.txt_\tout.txt\000"
.align 2
$LC17:
.ascii "\tcat_in.txt_\000"
.align 2
$LC18:
.ascii "_|_tp0_\tout.txt\000"
.align 2
$LC10:
.ascii "%s\n"
.ascii "%s\n"
.ascii "%s\n"
.ascii "%s\n"
.ascii "%s %s\t%s\n"
.ascii "%s %s %s\n\000"
.align 2
$LC11:
.ascii "Options:\000"

```



```

.mask 0xd0000000,-8
.fmask 0x00000000,0
.set noreorder
.cpload $t9
.set reorder
subu $sp,$sp,40
.cprestore 16
sw $ra,32($sp)
sw $fp,28($sp)
sw $gp,24($sp)
move $fp,$sp
sw $a0,40($fp)
lw $a0,40($fp)
la $t9,print_usage
jal $ra,$t9
lw $a0,40($fp)
la $t9,print_options
jal $ra,$t9
move $sp,$fp
lw $ra,32($sp)
lw $fp,28($sp)
addu $sp,$sp,40
j $ra
.end print_help
.size print_help, .-print_help
.rdata
.align 2
$LC19:
.ascii "Version_\%s\n\000"
.align 2
$LC20:
.ascii "1.0.0\000"
.text
.align 2
.ent print_version
print_version:
.frame $fp,40,$ra # vars= 0, regs= 3/0, args= 16, extra= 8
.mask 0xd0000000,-8
.fmask 0x00000000,0
.set noreorder
.cpload $t9
.set reorder
subu $sp,$sp,40
.cprestore 16
sw $ra,32($sp)
sw $fp,28($sp)
sw $gp,24($sp)
move $fp,$sp
la $a0,$LC19
la $a1,$LC20
la $t9,printf
jal $ra,$t9
move $sp,$fp
lw $ra,32($sp)

```



```

    lw $fp,28($sp)
    addu $sp,$sp,40
    j $ra
    .end print_version
    .size print_version, .-print_version
    .rdata
    .align 2
$LC22:
    .ascii "help\000"
    .align 2
$LC23:
    .ascii "version\000"
    .data
    .align 2
    .type long_options.0, @object
    .size long_options.0, 48
long_options.0:
    .word $LC22
    .word 0
    .word 0
    .word 104
    .word $LC23
    .word 0
    .word 0
    .word 86
    .word 0
    .word 0
    .word 0
    .word 0
    .word 0
    .rdata
    .align 2
$LC21:
    .ascii "hV\000"
    .align 2
$LC24:
    .ascii "Unrecognized options:\000"
    .align 2
$LC25:
    .ascii "%s\000"
    .align 2
$LC26:
    .ascii "\n\000"
    .text
    .align 2
    .globl argsHandler_parse_arguments
    .ent argsHandler_parse_arguments
argsHandler_parse_arguments:
    .frame $fp,64,$ra # vars= 16, regs= 3/0, args= 24, extra= 8
    .mask 0xd0000000,-8
    .fmask 0x00000000,0
    .set noreorder
    .cpld $t9
    .set reorder
    subu $sp,$sp,64

```

```

.cprestore 24
sw $ra,56($sp)
sw $fp,52($sp)
sw $gp,48($sp)
move $fp,$sp
sw $a0,64($fp)
sw $a1,68($fp)
sw $zero,36($fp)
la $v0,$LC21
sw $v0,40($fp)
$L75:
addu $v0,$fp,36
sw $v0,16($sp)
lw $a0,64($fp)
lw $a1,68($fp)
lw $a2,40($fp)
la $a3,long_options.0
la $t9,getopt_long
jal $ra,$t9
sw $v0,32($fp)
lw $v1,32($fp)
li $v0,-1 # 0xfffffffffffffffff
bne $v1,$v0,$L77
b $L76
$L77:
lw $v0,32($fp)
sw $v0,44($fp)
li $v0,86 # 0x56
lw $v1,44($fp)
beq $v1,$v0,$L80
lw $v1,44($fp)
slt $v0,$v1,87
beq $v0,$zero,$L84
li $v0,63 # 0x3f
lw $v1,44($fp)
beq $v1,$v0,$L81
b $L75
$L84:
li $v0,104 # 0x68
lw $v1,44($fp)
beq $v1,$v0,$L79
b $L75
$L79:
lw $v0,68($fp)
lw $a0,0($v0)
la $t9,print_help
jal $ra,$t9
move $a0,$zero
la $t9,exit
jal $ra,$t9
$L80:
la $t9,print_version
jal $ra,$t9
move $a0,$zero

```

```

    la $t9,exit
    jal $ra,$t9
$L81:
    li $a0,1    # 0x1
    la $t9,exit
    jal $ra,$t9
$L76:
    lw $v0,optind
    lw $v1,64($fp)
    slt $v0,$v0,$v1
    beq $v0,$zero,$L85
    la $a0,__$sF+176
    la $a1,$LC24
    la $t9,fprintf
    jal $ra,$t9
$L86:
    lw $v0,optind
    lw $v1,64($fp)
    slt $v0,$v0,$v1
    bne $v0,$zero,$L88
    b $L87
$L88:
    la $a1,optind
    lw $v1,0($a1)
    move $v0,$v1
    sll $a0,$v0,2
    lw $v0,68($fp)
    addu $v0,$a0,$v0
    addu $v1,$v1,1
    sw $v1,0($a1)
    la $a0,__$sF+176
    la $a1,$LC25
    lw $a2,0($v0)
    la $t9,fprintf
    jal $ra,$t9
    b $L86
$L87:
    la $a0,__$sF+176
    la $a1,$LC26
    la $t9,fprintf
    jal $ra,$t9
    li $a0,1    # 0x1
    la $t9,exit
    jal $ra,$t9
$L85:
    move $v0,$zero
    move $sp,$fp
    lw $ra,56($sp)
    lw $fp,52($sp)
    addu $sp,$sp,64
    j $ra
.end argsHandler_parse_arguments
.size argsHandler_parse_arguments, .-argsHandler_parse_arguments
.align 2

```

```

.globl main
.ent main
main:
    .frame $fp,48,$ra # vars= 8, regs= 3/0, args= 16, extra= 8
    .mask 0xd0000000,-8
    .fmask 0x00000000,0
    .set noreorder
    .cpload $t9
    .set reorder
    subu $sp,$sp,48
    .cprestore 16
    sw $ra,40($sp)
    sw $fp,36($sp)
    sw $gp,32($sp)
    move $fp,$sp
    sw $a0,48($fp)
    sw $a1,52($fp)
    lw $a0,48($fp)
    lw $a1,52($fp)
    la $t9,argsHandler_parse_arguments
    jal $ra,$t9
$L90:
    la $t9,process_line
    jal $ra,$t9
    sw $v0,24($fp)
    lw $v0,24($fp)
    beq $v0,$zero,$L90
    lw $v1,24($fp)
    li $v0,2 # 0x2
    bne $v1,$v0,$L94
    li $v0,-1 # 0xffffffffffffffff
    sw $v0,28($fp)
    b $L89
$L94:
    sw $zero,28($fp)
$L89:
    lw $v0,28($fp)
    move $sp,$fp
    lw $ra,40($sp)
    lw $fp,36($sp)
    addu $sp,$sp,48
    j $ra
.end main
.size main, .-main
.ident "GCC:(GNU) 3.3.3 (NetBSD nb3 20040520)"

```

5. Conclusiones

Como conclusión podemos enumerar una serie de conocimientos que adquirimos y/o pusimos en práctica durante el desarrollo de este trabajo práctico.

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. . Además adquirimos conocimiento respecto a otras herramientas útiles como ssh, scp, y diff entre otras.

- 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.
- Empleamos comandos y herramientas útiles como ssh, scp, diff, echo, cat, entre otros.
- Implementamos un buffer de manejo dinámico de memoria.
- Utilizamos valgrind para analizar la memoria alocada durante la ejecución, detectar leaks tanto de heap como de stack, y revisar los accesos a memoria con punteros.