Trabajo Práctico Nº 0: Infraestructura Básica

Martinez Ariel, *Padrón Nro. 88573*
arielcorreofiuba@gmail.com.ar

Nestor Huallpa, *Padrón Nro. 88614*
huallpa.nestor@gmail.com

Facundo Caldora, *Padrón Nro. 93194]*
facundo.caldora@gmail.com

1º Entrega: 22/09/2015

2do. Cuatrimestre de 2015
66.20 Organización de Computadoras
Facultad de Ingeniería, Universidad de Buenos Aires

Resumen

En el presente trabajo práctico se describirán todos los pasos y conclusiones relacionadas al desarrollo e implementación de multiplicacion de matrices de numero reales, representados en punto flotante de doble precisiòn.

Índice

1. Introducción

El objetivo del presentraba trabajo practico es familiarizarse con el emulador gxemul mediante la implementacion de un programa simple de multiplicación de matrices.

2. Implementación

2.1. Lenguaje

Como lenguaje de implementación se eligió ANSI C [**?**] ya que el mismo permite una alta portabilidad entre diferentes plataformas. El desarrollo del programa se realizó usando un editor de texto (gedit,vim, kwrite) y compilando los archivos fuente con GCC que viene en linux. Para compilar, ejecutar el siguiente comando:

$ make

2.2. Descripción del programa

Cuando se pasa un nombre como argumento, se verifica que dicho nombre que está haciendo referencia a un archivo (.txt) y no a un archivo de directorio. Una vez hecha la verificación el programa se dispone a leer cada una de las líneas desde el principio. Cada par de lineas leidas como validas se las cargan en memoria dinamica para su posterior multiplicacion, luego el resultado de la multiplicacion se lo imprime por salida estándar (stdout). La función `main` se encuentra en `tp0.c` y se encarga de interpretar las opciones y argumentos. En caso de ser una opción, como ayuda o versión, se imprime el mensaje correspondiente y finaliza la ejecución. Cuando no es una opción de ayuda o versión, se procede a procesar los datos de entrada. La salida de estas funciones proveen un codigo de error que sirve como salida del programa. Los mensajes de versión y ayuda se imprimen por stdout y el programa finaliza devolviendo 0 (cero) al sistema. Los mensajes de error se imprimen por la salida de errores (stderr) y el programa finaliza devolviendo 1 (uno) al sistema.

2.2.1. Errores posibles

1. El procesamiento de la entrada estándar causó el agotamiento del heap.
2. La invocación del programa es incorrecta.
3. Alguno de los archivos es inexistente.

Se contemplan otros errores gracias al uso de la variable externa errno. Cuando ocurre un error inesperado, el mismo es informado por stderr y finaliza el programa liberando la memoria que se habia solicitado hasta el momento. (con la funión perror()).

2.3. Desarrollo de actividades

1. Se instaló en un linux un repositorio de fuentes (GIT) para que al dividir las tareas del TP se pudiese hacer una unión de los cambios ingresados por cada uno de los integrantes más fácilmente.

2. Cada persona del grupo se comprometió a que sus cambios en el el código fuente y los cambios obtenidos del repositorio que pudiesen haber subido los otros integrantes del grupo, sean compilados los sistemas operativos Linux y el GXEmul, asegurando así portabilidad entre plataformas planteada en el enunciado.

3. Se estableció que todos los integrantes en mayor o menor medida, contribuyan en el desarrollo de todas las partes del código para que nadie quede en desconocimiento de lo que se hizo en cada sección. Si bien cada parte del código está comenazada por diferentes integrantes (parseo de los argumentos, lectura de los ficheros, etc), todos nos familiarizamos con cada una de estas partes y cumplimos la función de testers de lo hecho por otros integrantes.

4. Se propuso como meta paralela, hacer el programa lo mas reutilizable posible tratando de que los métodos desarrollados, sean los suficientemente modulares como para su posible reutilización en los TPs venideros.

5. Debido desconocimiento de LaTeX [**?**] para algunos integrantes del grupo, uno de los integrantes dió una breve introducción de como desarrollar este informe para que todos pudiésen modificarlo y agregar lo que considerase necesario. Para compilar el archivo del informe se usaron el compilador de LaTeX [**?**] propio que viene en el Linux.

6. Para poder generar el código assembler a partir del código fuente, dentro de NETBSD se utilizó gcc con la siguiente opción:

   gcc -S -Wall -O0 tp0.c -mrnames

7. Para crear el presente informe en formato PDF usando LaTeX [**?**] en Linux, ingresar los siguientes comandos:

   $ pdflatex tp0.tex tp0.pdf

2.4. Corridas de pruebas

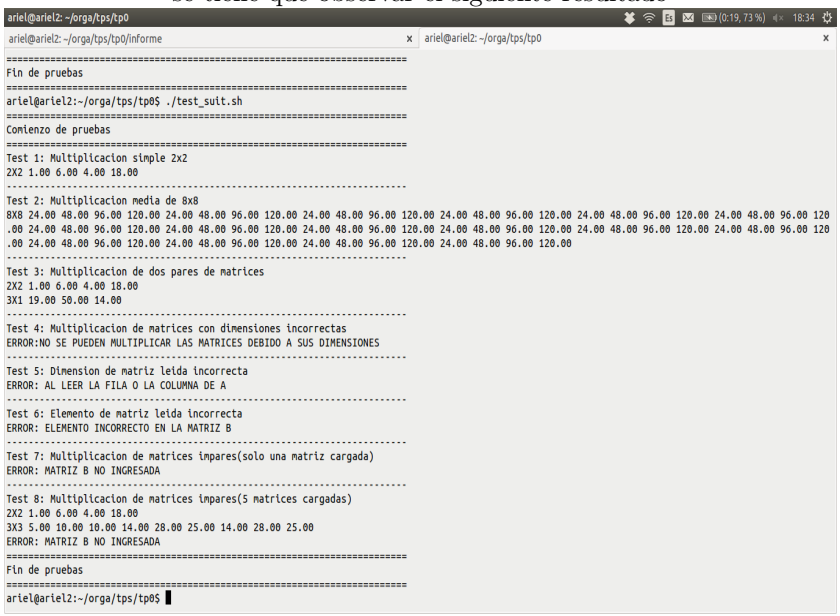
Para automatizar las pruebas se crearon los siguientes 3 script: `test_suit.sh`, `gen_matriz.sh` y `gen_test.sh`

1. Prueba 1

   Al ejecutar el siguiente comando

$ ./test_suit.sh

se tiene que observar el siguiente resultado

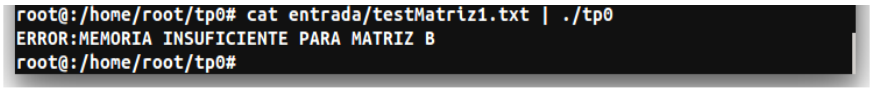


2. Prueba 2

Para poder ejecutar la prueba de memoria insuficiente se utiliza el script `gen_test.sh` que genera un archivo de entrada con dos matrices muy grandes. Adicionalmente este script corre el programa tp0 con valgrind.

$ ./gen_test.sh

Pero la prueba definitiva se realizo en NetBSD y puede observarse que el programa termina correctamente al quedarse sin memoria disponible, cuando lo corremos sobre nuestro NetBSD con 60 MB de memoria disponible.



3. El código fuente, en lenguaje C

```
/**
*    6620 - Organizacion del computador
*    Trabajo Practico 0
*    Alumnos:
*             88614 - Nestor Huallpa
*             88573 - Ariel Martinez
*             93194 - Facundo Caldora
*  - Validaciones de I/O y formato.
*  - Probar en el emulador y generar dump de assemble.
*  - Informe.
*/

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

typedef struct {
        int cantFil;
        int cantCol;
        double** datos;
} matriz;

double** mallocMatrizDouble(int cantFila, int cantCol)
{
    int i;
    int k;
    double** datos = ((double**)malloc(cantFila*sizeof(double*)));
    if (datos == NULL)
    {
        return NULL;
    }
    else
    {
        for (i=0; i<cantFila; i++)
        {
            datos[i] = NULL;
        }

        for (i=0;i<cantFila;i++)
        {
            datos[i] = ((double*)malloc(cantCol*sizeof(double)));
            if (datos[i]==NULL)
            {
                for (k=0;k<cantFila;k++)
                {
                    if (datos[k] != NULL) {
                        free(datos[i]);
                        datos[k] = NULL;
                    }
                }
                free(datos);
                datos = NULL;
                return NULL;
            }
        }
    }
    return datos;
}

void liberarMemoria(matriz* p_m)
{
    if (p_m != NULL)
    {
        int i;
        for(i=0;i<(*p_m).cantFil;i++)
        {
            if ((*p_m).datos[i] != NULL) {
                free((*p_m).datos[i]);
                (*p_m).datos[i] = NULL;
            }
        }
        free((*p_m).datos);
        (*p_m).datos=NULL;
    }
}

int main(int argc, char** argv) {

    int siguiente_opcion;

    /* Una cadena que lista las opciones cortas validas */
    const char* const op_cortas = "hV";

    /* Una estructura de varios arrays describiendo los valores largos */
    const struct option op_largas[] =
    {
      { "help",             0,   NULL,   'h'},
      { "version",          0,   NULL,   'V'},
      { NULL,               0,   NULL,    0 }
    };


    while (1) {
        siguiente_opcion = getopt_long (argc, argv, op_cortas, op_largas, NULL);
        if (siguiente_opcion == -1) break;
```

```
    switch (siguiente_opcion) {
        case 'h' :
            printf("Usage:\n");
            printf("\ttp0 -h\n");
            printf("\ttp0 -V\n");
            printf("\ttp0 <in_file >out_file\n");
            printf("Options:\n");
            printf("\t-V, --version       Print version and quit.\n");
            printf("\t-h, --help          Print this information and quit.\n");
            printf("Examples:\n");
            printf("\ttp0 <in.txt >out.txt\n");
            printf("\tcat in.txt | tp0 >out.txt\n");
            exit(0);
        break;

        case 'V' :
            printf("Tp0: Version 0.1: Grupo:\n");
            exit(0);
        break;
    }
}

    matriz m_a;
    matriz m_b;
    int i,j;
char dato;
FILE * fp = stdin;

    /* se cargan los datos para las 2 matrices desde el archivo*/
while(!feof(fp))
{

    m_a.cantFil = 0;
    m_a.cantCol = 0;
    m_b.cantFil = 0;
    m_b.cantCol = 0;

    /*Se leen los valores desde el archivo de fila columna y separador de ambos*/
    if (fscanf(fp, "%dx%d" , &m_a.cantFil, &m_a.cantCol) == 0)
    {
        fprintf(stderr, "ERROR: AL LEER LA FILA O LA COLUMNA DE A\n");
        exit(1);
    }
    /* Handlers de archivos mal ingresados */
    if (m_a.cantFil<0){
        fprintf(stderr, "ERROR: FILA INGRESADA INVALIDA PARA MATRIZ A\n");
        exit(1);
    }
    if (m_a.cantCol<0){
        fprintf(stderr, "ERROR: COLUMNA INGRESADA INVALIDA PARA MATRIZ A\n");
        exit(1);
    }
    if (m_a.cantFil==0 || m_a.cantCol == 0){
        fprintf(stderr, "ERROR: MATRIZ A NO INGRESADA O ESPACIO DE MAS EN EL ARCHIVO\n" );
        exit(1);
    }
    /* se aloja memoria para la matriz a */
    m_a.datos = mallocMatrizDouble(m_a.cantFil, m_a.cantCol);
    if (m_a.datos == NULL) {
        fprintf(stderr, "ERROR:MEMORIA INSUFICIENTE PARA MATRIZ A\n");
        exit(1);
    }

    for(i=0;i<m_a.cantFil;i++)
    {
        for(j=0;j<m_a.cantCol;j++)
        {
            if ( (dato = fgetc(fp)) == '\n'){
                fprintf(stderr, "ERROR: FALTAN ELEMENTOS EN MATRIZ A\n" );
                liberarMemoria(&m_a);
                exit(1);
            }
            if (fscanf(fp, "%lf", &m_a.datos[i][j]) == 0)
            {
                fprintf(stderr, "ERROR: ELEMENTO INCORRECTO EN LA MATRIZ A\n");
                liberarMemoria(&m_a);
                exit(1);
            }
        }
    }

    if (fscanf(fp, "%dx%d" , &m_b.cantFil, &m_b.cantCol) == 0)
    {
        fprintf(stderr, "ERROR: AL LEER LA FILA O LA COLUMNA DE B\n");
        liberarMemoria(&m_a);
        exit(1);
    }

    if (m_b.cantFil<0)
    {
        fprintf(stderr, "ERROR: FILA INGRESADA INVALIDA PARA MATRIZ B\n");
        liberarMemoria(&m_a);
        exit(1);
    }
    if (m_b.cantCol<0)
    {
        fprintf(stderr, "ERROR: COLUMNA INGRESADA INVALIDA PARA MATRIZ B\n");
        liberarMemoria(&m_a);
        exit(1);
```

```
        }

        if (m_b.cantFil==0 || m_b.cantCol == 0){
            fprintf(stderr, "ERROR: MATRIZ B NO INGRESADA \n" );
            liberarMemoria(&m_a);
            exit(1);
        }

        /* se aloja memoria para la matriz b */
        m_b.datos = mallocMatrizDouble(m_b.cantFil, m_b.cantCol);
        if (m_b.datos==NULL)
        {
            fprintf(stderr, "ERROR:MEMORIA INSUFICIENTE PARA MATRIZ B\n");
            liberarMemoria(&m_a);
            exit(1);
        }


        for(i=0;i<m_b.cantFil;i++)
        {
            for(j=0;j<m_b.cantCol;j++)
            {
                if( (dato = fgetc(fp)) == '\n'){
                    fprintf(stderr, "ERROR: FALTAN ELEMENTOS EN MATRIZ B\n" );
                    liberarMemoria(&m_a);
                    liberarMemoria(&m_b);
                    exit(1);
                }
                if (fscanf(fp, "%lf", &m_b.datos[i][j]) == 0)
                {
                    fprintf(stderr, "ERROR: ELEMENTO INCORRECTO EN LA MATRIZ B\n");
                    liberarMemoria(&m_a);
                    liberarMemoria(&m_b);
                    exit(1);
                }
            }
        }

        if (m_a.cantCol != m_b.cantFil)
        {
                fprintf(stderr, "ERROR:NO SE PUEDEN MULTIPLICAR LAS MATRICES DEBIDO A SUS DIMENSIONES\n");
            liberarMemoria(&m_a);
            liberarMemoria(&m_b);
            exit(1);
        }
        else
        {
            printf("%dX%d", m_a.cantFil, m_b.cantCol);
            int k = 0;
            double suma = 0.0;
            for (i=0; i<m_a.cantFil; i++)
            {
                for (j=0; j<m_b.cantCol; j++)
                {
                        suma = 0.0;
                    for (k=0; k<m_a.cantCol;k++)
                    {
                        suma = suma + (m_a.datos[i][k] * m_b.datos[k][j]);
                    }
                    printf(" %4.2lf", suma);
                }
            }
            printf("\n");

        }

        liberarMemoria(&m_a);
        liberarMemoria(&m_b);
    }

    return 0;
}
```

4. El código MIPS32 generado por el compilado

```
        .file   1 "main.c"
        .section .mdebug.abi32
        .previous
        .abicalls
        .rdata
        .align  2
$LC0:
        .ascii  "%dX%d\000"
        .align  2
$LC1:
        .ascii  "␣%4.2lf\000"
        .align  2
$LC2:
        .ascii  "\n\000"
        .text
        .align  2
        .globl  imprimirMatriz
        .ent    imprimirMatriz
imprimirMatriz:
        .frame  $fp,48,$31              # vars= 8, regs= 3/0, args= 16, extra= 8
        .mask   0xd0000000,-8
        .fmask  0x00000000,0
        .set    noreorder
        .cpload $25
        .set    reorder
        subu    $sp,$sp,48
        .cprestore 16
        sw      $31,40($sp)
        sw      $fp,36($sp)
        sw      $28,32($sp)
        move    $fp,$sp
        sw      $4,48($fp)
        lw      $2,48($fp)
        lw      $3,48($fp)
        la      $4,$LC0
        lw      $5,0($2)
        lw      $6,4($3)
        la      $25,printf
        jal     $31,$25
        sw      $0,24($fp)
$L18:
        lw      $2,48($fp)
        lw      $3,24($fp)
        lw      $2,0($2)
        slt     $2,$3,$2
```

5. Conclusiones

1. Si bien lo solicitado por el programa no era excesivamente difícil, la realización completa del TP llevó cierta dificultad al tener que realizarlo en el contexto solicitado: alta portabilidad, desarrollo en C, e informe hecho en LaTeX [**?**].

2. En el primer caso la dificultad radicaba en tener configurado y funcionando el GXEmul dentro de un Linux, y lograr que en ambos casos el programa compile y corra sin problemas.

3. Debido a nuestro desconocimiento con LaTeX [**?**], tuvimos que invertir tiempo en encontrar forma de realizar el presente documento de la manera más correcta posible

4. En cuanto al trabajo grupal en si mismo, no hubo inconvenientes de ningún tipo ya que al ser el grupo relativamente chico y tener conocimiento del manejo del versionado de un proyecto ante cambios ingresado por los integrantes (por medio del GIT), la introducción de modificaciones y correcciones fué fluida.

6. Enunciado del trabajo practico

Universidad de Buenos Aires - FIUBA
66.20 Organización de Computadoras
Trabajo práctico 0: Infraestructura básica
2^{do} cuatrimestre de 2015

`$Date: 2015/09/01 00:28:25 $`

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 de números reales, representados en punto flotante de doble precisión.

Las matrices a multiplicar ingresarán por entrada estándar (`stdin`), donde cáda línea describe una matriz completa en formato de texto, según el siguiente formato:

$NxM\ a_{1,1}\ a_{1,2}\ ...\ a_{1,M}\ \ a_{2,1}\ a_{2,2}\ ...\ a_{2,M}\ \ ...\ \ a_{N,1}\ a_{N,2}\ ...\ a_{N,M}$

La línea anterior representa a la matriz A de N filas y M columnas. Los elementos de la matriz A son los $a_{x,y}$, siendo x e y los indices de fila y columna respectivamente[1]. 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, dada la siguiente matriz:

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

Su representación sería:

```
2x3 1 2 3 4 5 6
```

Por cada par de matrices que se presenten en su entrada, el programa deberá multiplicarlas y presentar el resultado por su salida estándar (`stdout`) en el mismo formato presentado anteriormente, hasta que llegue al final del archivo de entrada (`EOF`). Ante un error, el progama deberá informar la situación inmediatamente (por `stderr`) y detener su ejecución. Tener en cuenta que también se condidera un error que a la entrada se presenten matrices de dimensiones incompatibles entre sí para su multiplicación.

[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 in.txt
2x3 1 2 3 4 5 6.1
3x2 1 0 0 0 0 1
3x3 1 2 3 4 5 6.1 3 2 1
3x1 1 1 0

$ cat in.txt | ./tp0
2x2 1 3 4 6.1
3x1 3 9 5
```

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`):

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

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6.1 \\ 3 & 2 & 1 \end{pmatrix} \times \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 3 \\ 9 \\ 5 \end{pmatrix}$$

5.3. 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 22/9/2015.

Referencias

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

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

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