

TRABAJO PRACTICO N° 0

Infraestructura básica

Ezequiel Paez, *Padrón Nro.*

Juan Manuel Gonzales, *Padrón Nro.*

Osiris Argüello, *Padrón Nro. 83062*
`osirisarguello@yahoo.com.ar`

1er. Cuatrimestre de 2012

66.20 Organización de Computadoras - Curso 02
Facultad de Ingeniería, Universidad de Buenos Aires

Índice

1. Resumen	6
2. Diseño e Implementación	6
2.1. procesar_ordenamiento	6
2.2. mergeSort	6
2.3. selectionSort	7
3. Instrucciones de Compilación	7
4. Corridas de Prueba	8
4.1. Corridas exitosas	8
4.2. Mensajes de Error	10
5. Mediciones y Profiling	12
6. Código fuente	13
6.1. Lenguaje C	13
7. Conclusiones	19

1. Resumen

En el presente trabajo práctico, se deben implementar en un programa en lenguaje C dos algoritmos de ordenamiento: Mergesort y Selection sort. Una vez implementados, procederemos a realizar mediciones para evaluar las posibilidades de mejora y el desempeño relativo entre ambas implementaciones, utilizando los programas `time` y `gprof`.

2. Diseño e Implementación

El diseño de la aplicación se realizó pensando en varias «capas». Una capa es encargada de analizar y validar opciones de ejecución, otra capa, la encargada de realizar las operaciones principales de conversión y otras más pequeñas encargadas de emitir mensajes de error o información de ayuda en pantalla para el usuario.

Dado que la aplicación puede tomar parámetros variables de ejecución o adoptar valores por defecto, se realizó un análisis completo de todas las opciones que un usuario puede ingresar, aceptando aquellas que fueren válidas y rechazando las que no, con el consiguiente mensaje de error que el usuario visualizará en pantalla.

La aplicación fue implementada en lenguaje ANSI C, es decir que está garantizada la compilación exitosa ante la utilización de la opción `-ansi`.

La siguiente capa importante, es la capa de los algoritmos de ordenamiento. Se desarrollaron dos algoritmos de ordenamiento: *mergesort* y *selectionsort*.

La implementación para ambos algoritmos básicamente consta de procesos similares, ligeramente diferenciados en el punto en que se invoca una función ANSI C distinta de acuerdo al tipo de ordenamiento que se esté realizando. Como primer paso del algoritmo, se abre el *stream* de entrada de datos que es especificado por el usuario. De no resultar satisfactoria esta operación, ya sea porque el archivo no existe o se produjere un fallo en el disco en el momento de la apertura, se retorna de inmediato el correspondiente código de error. Si el archivo es abierto satisfactoriamente se procede a leer su contenido, realizando el ordenamiento indicado por el usuario y el resultado de dicha operación es volcado al archivo de salida también especificado en los parámetros de ejecución.

Las principales funciones en nuestra implementación son las siguientes:

2.1. procesar_ordenamiento

Este método se encarga de tomar los archivos de entrada y salida, abrirlos y luego ejecutar el método de ordenamiento correspondiente. Por último devuelve el resultado en el archivo de salida indicado.

2.2. mergeSort

Este método toma el array de caracteres leído del archivo, lo divide en 2 partes e invoca recursivamente la función **mergeSort** sobre cada parte. Como último paso invoca a la función **merge** para fusionar ambas partes, ordenandolas.

2.3. selectionSort

Este método recorre el array de caracteres, buscando el menor caracter en cada iteración y ubicandolo al principio del array.

3. Instrucciones de Compilación

Si el objetivo es exclusivamente obtener el ejecutable y probar la aplicación, las instrucciones básicas para compilación mediante línea de comandos (situado en el directorio que contiene el archivo «main.c») son:

```
$ gcc -o tp0 main.c
```

Donde:

- -o: archivo de salida (en este caso, tp0).

Si en cambio, se desean visualizar todos los mensajes de advertencia, verificar problemas de acceso a memoria, depurar el código, optimizarlo, etc., las opciones de compilación pueden ser:

```
$ gcc -Wall -g -o tp0 main.c
```

Donde:

- -Wall: permite que se visualicen todos los mensajes de advertencia,
- -g: permite realizar un seguimiento generando información de depuración. Por ejemplo, con algún «*profiler*» como ***gdb***,
- -o: archivo de salida (en este caso, tp0)

Para generar el código assembler se utiliza:

```
$ gcc -Wall -O0 -S -mrnames main.c,
```

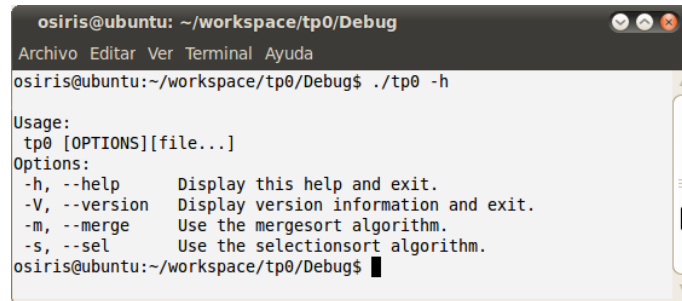
Donde:

- -O0: indica nivel cero de optimización,
- -S: envía la instrucción de no ejecutar el linker luego de generar el código assembler en el archivo *main.s*,
- -mrnames (solo para MIPS): indica al compilador que genere la salida utilizando nombre de registro en lugar de número de registro.

4. Corridas de Prueba

4.1. Corridas exitosas

Mensaje de ayuda. Se obtiene utilizando los comandos `-h` o `-help`.

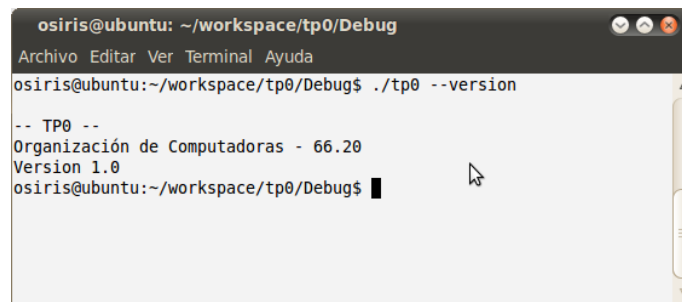


```
osiris@ubuntu: ~/workspace/tp0/Debug
Archivo Editar Ver Terminal Ayuda
osiris@ubuntu:~/workspace/tp0/Debug$ ./tp0 -h

Usage:
tp0 [OPTIONS][file...]
Options:
-h, --help      Display this help and exit.
-V, --version   Display version information and exit.
-m, --merge     Use the mergesort algorithm.
-s, --sel       Use the selectionsort algorithm.
osiris@ubuntu:~/workspace/tp0/Debug$
```

Figura 1: Mensaje de ayuda

Versión del programa. Se obtiene utilizando los comandos `-V` o `-version`.

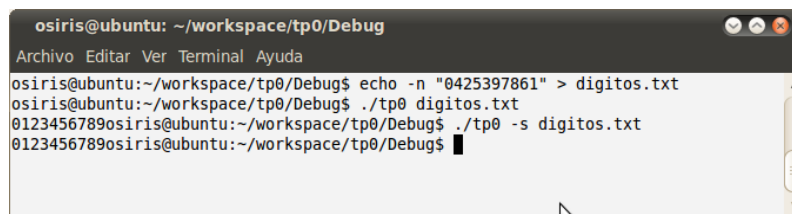


```
osiris@ubuntu: ~/workspace/tp0/Debug
Archivo Editar Ver Terminal Ayuda
osiris@ubuntu:~/workspace/tp0/Debug$ ./tp0 --version

-- TP0 --
Organización de Computadoras - 66.20
Version 1.0
osiris@ubuntu:~/workspace/tp0/Debug$
```

Figura 2: Mensaje de versión del programa

Para esta corrida se eligió ordenar un archivo con dígitos y se prueban ambos métodos de ordenamiento. En el primer caso, al no pasar parámetro se está tomando por defecto el ordenamiento por mergesort. El resultado se muestra por pantalla.



```
osiris@ubuntu: ~/workspace/tp0/Debug
Archivo Editar Ver Terminal Ayuda
osiris@ubuntu:~/workspace/tp0/Debug$ echo -n "0425397861" > digitos.txt
osiris@ubuntu:~/workspace/tp0/Debug$ ./tp0 digitos.txt
0123456789osiris@ubuntu:~/workspace/tp0/Debug$ ./tp0 -s digitos.txt
0123456789osiris@ubuntu:~/workspace/tp0/Debug$
```

Figura 3: Ordenar digitos.txt

Para la siguiente corrida se eligió ordenar un archivo de caracteres con minúsculas y mayúsculas y se corren ambos métodos de ordenamiento. En el primer caso la entrada y la salida son por *stdin* y *stdout*. En el segundo ambos son archivos.

```
osiris@ubuntu: ~/workspace/tp0/Debug
Archivo Editar Ver Terminal Ayuda
osiris@ubuntu:~/workspace/tp0/Debug$ cat letters.txt
aAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ
osiris@ubuntu:~/workspace/tp0/Debug$ ./tp0 --merge <letters.txt

ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyzosiris@ubuntu:~/workspace/tp0/
Debug$
osiris@ubuntu:~/workspace/tp0/Debug$ ./tp0 --sel letters.txt result.txt
osiris@ubuntu:~/workspace/tp0/Debug$ cat result.txt

ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyzosiris@ubuntu:~/workspace/tp0/
Debug$
```

Figura 4: Ordenar letters.txt

En esta corrida se eligió ordenar un archivo con caracteres aleatorios.

```
osiris@ubuntu: ~/workspace/tp0/Debug
Archivo Editar Ver Terminal Ayuda
osiris@ubuntu:~/workspace/tp0/Debug$ head -c 64 /dev/urandom > random.txt
osiris@ubuntu:~/workspace/tp0/Debug$ hexdump -C random.txt
00000000  67 35 60 45 10 4f fd ce 13 2b 8b 15 8d c5 b5 2c |g5`E.0...+....|
00000010  73 8e 8b db c7 57 2c 54 29 60 28 54 e1 41 f7 2a |S...W,T)`(T.A.*|
00000020  11 58 b7 a7 4e bc b5 2b 6f 59 6f 5c a4 8c bd 2d |.X..N..+oYo\...-|
00000030  5c d7 6f ac 60 8e 29 53 2a ac 76 1a 52 bf a4 a0 |\.o.`.)S*.v.R...|
00000040
osiris@ubuntu:~/workspace/tp0/Debug$ ./tp0 -m random.txt result.txt
osiris@ubuntu:~/workspace/tp0/Debug$ hexdump -C result.txt
00000000  8b 8b 8c 8d 8e 8e a0 a4 a4 a7 ac ac b5 b5 b7 bc |.....|
00000010  bd bf c5 c7 ce d7 db e1 f7 fd 10 11 13 15 1a 28 |.....(|
00000020  29 29 2a 2a 2b 2b 2c 2c 2d 35 41 45 4e 4f 52 53 |))**+,,-5AENORS|
00000030  54 54 57 58 59 5c 5c 60 60 60 67 6f 6f 6f 73 76 |TTWXY\\```goosv|
00000040
osiris@ubuntu:~/workspace/tp0/Debug$ ./tp0 -s random.txt result2.txt
osiris@ubuntu:~/workspace/tp0/Debug$ hexdump -C result2.txt
00000000  8b 8b 8c 8d 8e 8e a0 a4 a4 a7 ac ac b5 b5 b7 bc |.....|
00000010  bd bf c5 c7 ce d7 db e1 f7 fd 10 11 13 15 1a 28 |.....(|
00000020  29 29 2a 2a 2b 2b 2c 2c 2d 35 41 45 4e 4f 52 53 |))**+,,-5AENORS|
00000030  54 54 57 58 59 5c 5c 60 60 60 67 6f 6f 6f 73 76 |TTWXY\\```goosv|
00000040
osiris@ubuntu:~/workspace/tp0/Debug$
```

Figura 5: Ordenar random.txt

4.2. Mensajes de Error

Las siguientes situaciones producen mensajes de error:

Si ocurre un fallo al abrir el file, se mostrará el mensaje de la figura junto con el nombre del file.

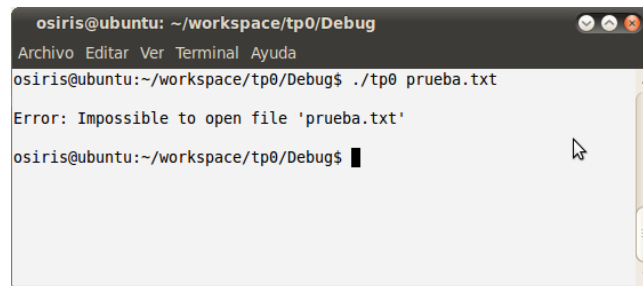
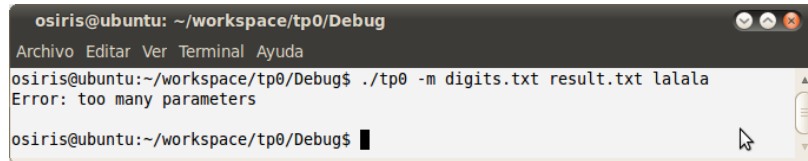


Figura 6: Error al abrir el file

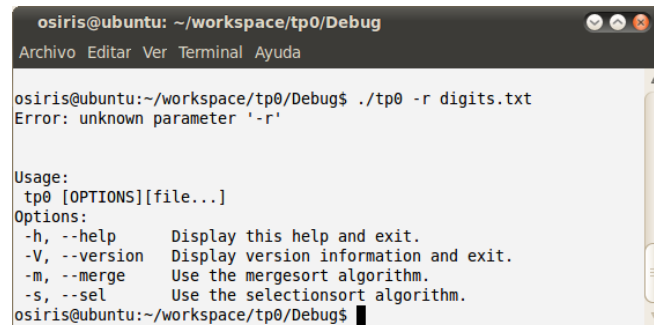
Si al ejecutar se ingresan más parámetros de los que puede interpretar el programa, se mostrará el mensaje de la figura y seguido el menú de ayuda.



```
osiris@ubuntu: ~/workspace/tp0/Debug
Archivo Editar Ver Terminal Ayuda
osiris@ubuntu:~/workspace/tp0/Debug$ ./tp0 -m digits.txt result.txt lalala
Error: too many parameters
osiris@ubuntu:~/workspace/tp0/Debug$
```

Figura 7: Error debido a muchos parámetros

Si al ejecutar se ingresa algún parámetro inesperado por el programa, se mostrará el mensaje de la figura junto con el parámetro desconocido y el menú de ayuda.



```
osiris@ubuntu: ~/workspace/tp0/Debug
Archivo Editar Ver Terminal Ayuda
osiris@ubuntu:~/workspace/tp0/Debug$ ./tp0 -r digits.txt
Error: unknown parameter '-r'

Usage:
tp0 [OPTIONS][file...]
Options:
-h, --help      Display this help and exit.
-V, --version   Display version information and exit.
-m, --merge     Use the mergesort algorithm.
-s, --sel       Use the selectionsort algorithm.
osiris@ubuntu:~/workspace/tp0/Debug$
```

Figura 8: Error por parámetro desconocido

5. Mediciones y Profiling

6. Código fuente

6.1. Lenguaje C

main.c

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

void PrintHelp();
void PrintVersion();
void PrintTooManyParamError();
void PrintParameterError(char* param);
void PrintFileError(char* filename);
int decodeProcess(const char* filein, const char* fileou);
int encodeProcess(const char* input, const char* output, int lineLength);
int procesarOrdenamiento(const char* filein, const char* fileou, char* action);
char* mergeSort(char* list);
char* selectionSort(char* list);
long filesize(FILE** fd);
char* merge(char* left, char* right);
void swap(char* array, int index1, int index2);
char* substring(char* string, int position, int length);

int main(int argc, char* argv[])
{
    char showVersion = 0;
    char showHelp = 0;
    char showUnknownParam=0;
    char foundInput = 0;
    char* action = 0;
    char* input = 0;
    char* output = 0;

    int numArg = 1;

    if (argc>4){
        PrintTooManyParamError();
        return 1;
    }

    while (numArg < argc && !showVersion && !showHelp && !showUnknownParam){
        char* param = argv[numArg];
        int paramLength = strlen(param);

        if (param[0] == '-')
        {
            if (paramLength == 2)
            {
                if (param[1] == 'v' || param[1] == 'V')
                    showVersion = 1;
                else if (param[1] == 'h' || param[1] == 'H')
                    showHelp = 1;
                else if (param[1] == 'm' || param[1] == 'M'
                    || param[1] == 's' || param[1] == 'S'){
                    action=param;
                }
                else{
                    PrintParameterError(param);
                    showUnknownParam = 1;
                }
            }
            else if (strcmp(param, "--version") == 0)
```

```

        showVersion = 1;
    else if (strcmp(param, "--help") == 0)
        showHelp = 1;
    else if (strcmp(param, "--merge") == 0 ||
             strcmp(param, "--sel") == 0){
        action=param;
    }
    else{
        PrintParameterError(param);
        showUnknownParam = 1;
    }

    } else {
        if (!foundInput){
            input=param;
            foundInput=1;
        } else{
            output=param;
        }
    }

    numArg++;
}

if (showVersion){
    if (argc>2){
        PrintTooManyParamError();
        PrintHelp();
    } else {
        PrintVersion();
    }
}
else if (showHelp){
    if (argc>2){
        PrintTooManyParamError();
    }
    PrintHelp();
} else if (showUnknownParam){
    PrintHelp();
}
else
{
    int error = 0;

    if (!input)
        input = "stdin";

    if (!output)
        output = "stdout";

    error=procesarOrdenamiento(input, output, action);

    if (error==-1){
        PrintFileError(input);
    } else {
        if (error==-2){
            PrintFileError(output);
        }
    }

    return error;
}

return 0;
}

```

```

long filesize(FILE **fd){
    fseek(*fd, 0, SEEK_END); /* seek to end of file*/
    long size = ftell(*fd); /* get current file pointer*/
    fseek(*fd, 0, SEEK_SET); /* seek back to beginning of file*/
    /* proceed with allocating memory and reading the file*/

    return size;
}

int procesarOrdenamiento(const char* filein ,const char* fileou ,
                        char* action) {
    FILE *fdi , *fdo;
    char* leido=0;

    fdi=fopen(filein , "rb");
    if (!fdi) return -1; /* Error while opening input file */
    fdo=fopen(fileou , "wt");
    if (!fdo) return -2; /* Error while opening output file */

    long size=filesize(&fdi);

    leido=malloc((size)*sizeof(char));

    fgets(leido , size+1,fdi);

    if (!action || ((strcmp(action , "--merge") == 0) ||
        (strcmp(action , "-m") == 0))){
        /* Invocando el metodo mergeSort */
        leido=mergeSort(leido);
    }
    else if (action && ((strcmp(action , "--sel") == 0) ||
        (strcmp(action , "-s") == 0))){
        /*Invocando el metodo selectionSort */
        leido=selectionSort(leido);
    }
    else
    {
        fprintf(stderr , "Unknown action\n");

        free(leido);
        fclose(fdi);
        fclose(fdo);
        return 1;
    }

    fputs(leido , fdo);

    free(leido);

    fclose(fdi);
    fclose(fdo);
    return 0; /*Successfully finished*/
}

char* mergeSort(char* list){

    char* left=0;
    char* right=0;
    char* result=list;

    long length=strlen(list);

    if (length==1)

```

```

        return result;

    long middle=length/2;
    long lengthRight=length-middle;

    left=substring(list,0,middle);
    right=substring(list,middle+1,lengthRight);

    left=mergeSort(left);
    right=mergeSort(right);

    result=merge(left,right);

    return result;
}

char* selectionSort(char* list) {

    int i, j;
    int minPos;

    int length=strlen(list);

    for (i = 0; i < length; i++) {
        minPos = i;

        for (j = i+1; j < length; j++) {
            if (list[j] < list[minPos]) {
                minPos = j;
            }
        }

        if (minPos != i) {
            swap(list, i, minPos);
        }
    }

    return list;
}

void swap(char* array, int index1, int index2) {
    char aux = array[index1];
    array[index1] = array[index2];
    array[index2] = aux;
}

char* merge(char* left, char* right){

    long length_left=strlen(left);
    long length_right=strlen(right);

    char* result=malloc((length_left+length_right+1)*sizeof(char));

    long i=0;
    long j=0;
    long c=0;

    while(i<length_left || j<length_right){
        if (i<length_left && j<length_right){
            if (left[i]<=right[j]){
                result[c]=left[i];
                i++;
            } else {

```

```

        result[c]=right[j];
        j++;
    }

    } else if(i<length_left){
        result[c]=left[i];
        i++;
    } else if(j<length_right) {
        result[c]=right[j];
        j++;
    }

    c++;
}

result[c]='\0';

return result;
}

char *substring(char *string, int position, int length)
{
    char *pointer;
    int c;

    pointer = malloc(length+1);

    if( pointer == NULL )
    {
        printf("Unable to allocate memory.\n");
        exit(EXIT_FAILURE);
    }

    for( c = 0 ; c < position -1 ; c++ )
        string++;

    for( c = 0 ; c < length ; c++ )
    {
        *(pointer+c) = *string;
        string++;
    }

    *(pointer+c) = '\0';

    return pointer;
}

void PrintTooManyParamError()
{
    fprintf(stderr,"Error: too many parameters\n\n");
}

void PrintParameterError(char* param)
{
    fprintf(stderr,"Error: unknown parameter '%s' \n\n",param);
}

void PrintFileError(char* filename)
{
    fprintf(stderr,"Error: Impossible to open file '%s' \n\n",filename);
}

void PrintVersion(){
    printf("-- TP0 --\n");
    printf("Organizaci n de Computadoras - 66.20\n");
}

```

```
        printf(" Version  1.0\n");
    }

void PrintHelp()
{
    printf(" Usage:\n");
    printf("  tp0  [OPTIONS] [ file ... ]\n");
    printf(" Options:\n");
    printf(" -h, --help \t Display this help and exit.\n");
    printf(" -V, --version \t Display version information and exit.\n");
    printf(" -m, --merge \t Use the mergesort algorithm.\n");
    printf(" -s, --sel \t Use the selectionsort algorithm.\n");
}
```

7. Conclusiones

El Trabajo Práctico nos permitió introducirnos en las herramientas de la materia y familiarizarnos con las mismas: el emulador GXEmule, el Compilador GCC. Además tuvimos un breve repaso de programación en C y un pantallazo de lo próximamente veremos más a fondo que es Assembly MIPS.

Referencias

- [1] “Computer Architecture. A Quantitative Approach”, J. L. Hennessy and D. A. Patterson, 3ra Edición, Morgan Kaufmann Publishers, 2000.
- [2] “Taller de Programacion I”, Gabriel Agustín Praino
- [3] “Usando L^AT_EX 1.97”, Laura M. Castro Souto, Juan José Iglesias González,
<http://latex.gpul.org/html/index.html>