66.20 Organización de Computadoras: Trabajo Práctico 0

Eirik Harald Lund, *Padrón Nro. 103081*
eirikharald@hotmail.com

Grupo Nro. ? - 1er. Cuatrimestre de 2018
66.20 Organización de Computadoras
Facultad de Ingeniería, Universidad de Buenos Aires

09/04/2018

Resumen

El objectivo del trabajo es desarrollar un programa parecido a *wc*. Es decir, un programa para contar las palabras, las líneas y los caracteres de un archivo, en el lenguaje C.

El programa recibirá el nombre de un archivo de texto e imprimirá por *stdout* los números y el nombre del archivo. Se puede también usar varias opciones como *-h* para ver el mensaje de ayuda.

El programa debe funcionar en una máquina virtual con MIPS corriendo el sistema operativo NetBSD.

1. Documentación

Para empezar analicé qué necesitaba el programa; abrir y leer un archivo o un texto por *stdin* carácter por carácter, contar y luego imprimir por *stdout*. Todo debía ser en el lenguaje C. Todos los requerimientos eran parte de la biblioteca standard de C.

Entoces, los pasos para desarrollarlo eran:

1. Abrir un archivo o empezar a leer de *stdin*.
2. Leer el texto carácter por carácter.
3. Mientras leyendo, contar cada carácter, palabra y línea

4. Imprimir los datos descubiertos por *stdout*, más el nombre del archivo.

El primer paso se puede realizar usando la función *fopen(const char *filename, const char *mode)*. Si no se usa un archivo, este paso no hace nada.

El segundo paso se puede realizar usando la función *fgetc(FILE *stream)*. Esta funciona con un archivo del último paso y con *stdin* también. La función devuelve el próximo carácter del *stream*.

En el tercer paso, se necesita identificar palabras y líneas. El carácter " n"siempre implica una nueva línea. Para descubrir palabras, se necesita más:

El cuarto paso consiste en imprimir con *printf*. La entrada del usario da cuales datos se van a imprimir; *-l* significa líneas, *–words* significa todos los tres, etcétera.

Para usar el programa en MIPS, se usaba un túnel de SSH entre la máquina *host* y la máquina virtual.

2. Comandos

2.1. Comandos para iniciar NetBSD

Para iniciar MIPS con NetBSD, se usa:

```
hostOS$ ./gxemul -e 3max -d netbsd-pmax.img
```

Luego, dado que existe un usario *gxemul* con el IP 172.20.0.1, se conecta a la OS host:

```
guestOS$ ssh -R 2222:127.0.0.1:22 gxemul@172.20.0.1
```

Para completar el túnel de SSH:

```
hostOS$ ssh -p 2222 root@127.0.0.1
```

Ahora se tiene una conexión entre los dos OS y se puede usar la máquina virtual en el terminal.

2.2. Comandos para usar el programa

Para compilar el programa se usa el comando:

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

Para ejecutar el programa con un archivo e imprimir los resultados:

```
$ ./tp0 --words -i example.txt
```

O usando *stdin*:

```
$ ./tp0 --words < example.txt
```

La opción *–words* da todos los datos. Para obtener caracteres, líneas o palabras, se puede usar cualquier combinación de *-c, -l* y *-w*:

```
$ ./tp0 -w -c -i example.txt
```

Hay dos comandos más; *-h* y *-V*, con los variantes *–help* y *-version*, para ver la información de ayuda y de la versión:

```
$ ./tp0 -h -V
$ ./tp0 --help --version
```

3. Corridas de prueba

Se hicieron unas corridas de prueba para asegurar la exactitud y funcionalidad del programa.

3.1. Prueba 1

Dado un archivo vacío *input.txt*, ejecutamos:

```
$ ./tp0 --words -i input.txt
$  0 0 0 input.txt
```

Como el archivo está vacío, no tiene ningún carácter, palabra o línea. Usando *wc*:

```
$ wc input.txt
$ 0 0 0 input.txt
```

3.2. Prueba 2

Si el archivo tiene la frase "Hello, World!", el programa nos da:

```
$ ./tp0 --words -i input.txt
$  1 2 14 input.txt
```

Una línea, dos palabras y 14 caracteres, incluyendo el "\n"de una nueva línea.

3.3. Prueba 3

Se puede también usar *stdin*:

```
$ ./tp0 --words < input.txt
$ Using stdin
$  1 2 14
```

La salida es igual, salvo el nombre del archivo.

3.4. Prueba 4

Con un archivo que no existe:

```
$ ./tp0 -i fake.txt
$ Invalid file name
$ Aborting
```

Si el archivo no existe, deberíamos parar la ejecución.

3.5. Prueba 5

Hacemos una prueba con uno de los archivos dados:

```
$ ./tp0 --words -i alice.txt
$ 4046 30355 177428 alice.txt
$ wc alice.txt
$  4046  30355 177428 alice.txt
```

Los números son iguales, la prueba es exitosa.

3.6. Prueba 6

Un ejemplo más difícil, ya que hay símbolos no de ASCII:

```
$ ./tp0 --words -i elquijote.txt
$ 37826 384258 2198907 elquijote.txt
$ wc elquijote.txt
$  37826  384258 2198907 elquijote.txt
```

No causan problemas.

4. Mediciones

Se hicieron también corridas para comparar el desempeño del programa desarrollado con *wc*. Todas fueron realizadas bajo el entorno MIPS usando un comando de la forma:

```
$ time ./tp0 --words -i input.txt
$ time wc input.txt
```

Texto	tp0	wc	Tamaño
alice.txt	0.3237s	0.3097s	174KB
beowulf.txt	0.3980s	0.3980s	220KB
cyclopedia.txt	1.1243s	1.0610s	644KB
elquijote.txt	3.6635s	3.5086s	2151KB

Cuadro 1: Mediciones de tiempo insumido corriendo en la máquina virtual, 10 corridas (tiempo real).

Tiempo de corrido en función del tamaño de la entrada.

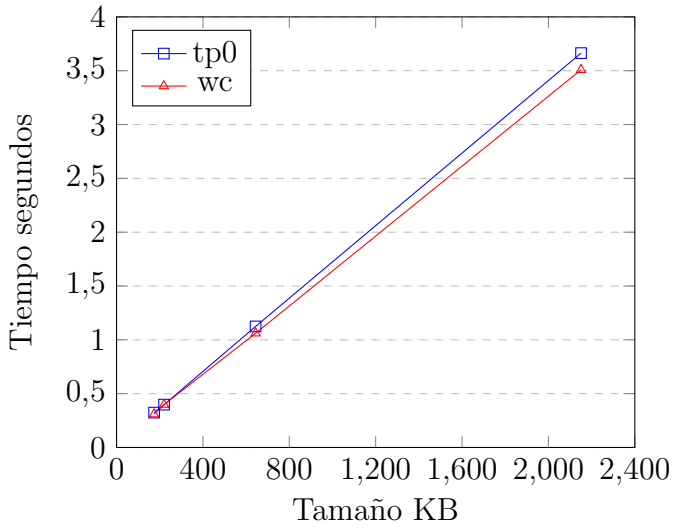
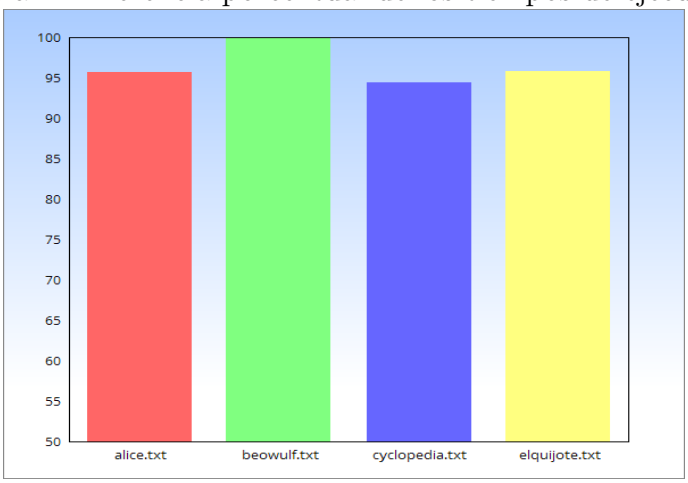


Figura 1: Diferencia porcentual de los tiempos de ejecución



Como se puede ver en el primer gráfico arriba, los dos programas tienen las mismas formas, pero *tp0* tiene una constante más grande. Esto es de esperar en vista de que los programas de GNU y UNIX generalmente son bien optimizados. No obstante, las diferencias son pequeñas. Así que, se puede pensar que la diferencia está en cómo se leen y analizan los caracteres, antes que en algún tipo de *overhead*.

El segundo gráfico muestra la diferencia de tiempo porcentual. La diferencia se queda más o menos constante, que implica que los programas tienen la misma complejidad. Es decir, dado un archivo de tamaño arbitrario, se puede esperar que *wc* lleve aproximadamente 95 % del tiempo que necesite *tp0* para analizarlo.

5. Conclusión

El programa desarrollado es un programa parecido al de GNU/Linux *wc*. *wc* tiene algunas funcionalidades adicionales, pero el uso general es igual. Sin embargo, el desempeño del programa Linux es mejor. Teniendo en cuenta que *wc* está hecho por un grupo de programadores profesionales, durante un tiempo mucho más largo que esto, la diferencia es bastante pequeña. No obstante, la diferencia probablemente viene del método de analizar cada carácter de un archivo. Con más tiempo, se podría realizar un programa con desempeño similar, o mejor, ya que el alcance del *tp0* es más peque no.

6. Código fuente

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <getopt.h>
#include <limits.h>

void
print_help()
{
    printf( "Usage:\n" );
    printf( "\ttp0 -h\n" );
    printf( "\ttp0 -V\n" );
    printf( "\ttp0 [options] file\n" );
    printf( "Options:\n" );
    printf( "\t-V, --version\tPrint version and quit.\n
       " );
    printf( "\t-h, --help\tPrint this information.\n" )
       ;
    printf( "\t-l, --words\tPrint number of lines in
       file.\n" );
    printf( "\t-w, --words\tPrint number of words in
       file.\n" );
    printf( "\t-c, --words\tPrint number of characters
       in file.\n" );
    printf( "\t-i, --input\tPath to input file.\n" );
    printf( "Examples:\n" );
    printf( "\ttp0 -w -i input.txt\n\n" );
}

void print_version()
{
    printf( "tp0 1.0\n" );
    printf( "Copyright Â© 2017 FIUBA.\n" );
    printf( "This is free software: you are free to
       change and redistribute it. There is NO WARRANTY
       , to the extent permitted by law.\n" );
    printf( "Written by Eirik Harald Lund\n\n" );
```

```
}

void
count ( FILE* fp, int line, int word, int character )
{
    int nl; // counter for lines
    nl = 0;
    int nw; // counter for words
    nw = 0;
    int nc; // counter for characters
    nc = 0;

    int c; // Working byte/character
    int co; // Previous character
    co = -1; // To avoid using unassigned variable

    while ( EOF != ( c = fgetc( fp ) ) )
    {
        if ( '\n' == c )
            ++nl;
        if ( !isspace( co ) && isspace( c ) ) // Words
            are determined by a non-space character
            followed by a space of any kind
            ++nw;
        ++nc;

        if ( INT_MAX == nc ) // Avoid infinite files
            crashing the program
        {
            printf( "Maximum amount of allowed
                characters reached, halting execution.\t
                " );
            break;
        }

        co = c;
    }

    if ( line ) printf( " %d", nl );
    if ( word ) printf( " %d", nw );
    if ( character ) printf( " %d", nc );
```

```
}

int
main ( int argc, char *argv[] )
{
    FILE *fp;
    fp = NULL;
    char *file;

    // Values for determining whether to print number
       of lines, words and characters
    int l = 0, w = 0, c = 0;

    static struct option long_options[] = {
        {"version", no_argument, 0, 'V'},
        {"help", no_argument, 0, 'h'},
        {"words", no_argument, 0, 'x'},
        {NULL, 0, NULL, 0}
    };
    int opt;
    int option_index;
    option_index = 0;
    while ((opt = getopt_long (argc, argv, "Vhlwci:",
       long_options, &option_index )) != -1)
    {
        switch (opt)
        {
        case 'V':
            print_version();
            break;
        case 'h':
            print_help();
            break;
        case 'l':
            l = 1;
            break;
        case 'w':
            w = 1;
            break;
        case 'c':
```

```
            c = 1;
            break;
        case 'x':
            l = 1;
            w = 1;
            c = 1;
            break;
        case 'i':
            fp = fopen( optarg, "r" );
            if ( NULL == fp )
            {
                printf( "Invalid file name\nAborting\n"
                    );
                return EXIT_FAILURE;
            }
            file = optarg;
            break;
        }
    }

    if ( NULL == fp )
    {
        printf( "Using stdin\n" );
        fp = stdin;
    }

    count( fp, l, w, c );
    fclose( fp );
    if ( stdin == fp ) printf( "\n" ); // Can't print a
        non-existant file name
    else printf( " %s\n", file );

    return EXIT_SUCCESS;
}
```