Trabajo Práctico 0
Infraestructura Básica

Melina Cheminet, *Padrón 102123*
mcheminet@fi.uba.ar

Fernández Elfi, Facundo, *89843*
ffelfis@gmail.com

David Batallan, *Padrón 97529*
dbatallan@fi.uba.ar

2do. Cuatrimestre de 2022
86.37 / 66.20 Organización de Computadoras
Facultad de Ingeniería, Universidad de Buenos Aires

22/09/2022

Resumen

El objetivo de este trabajo es familiarizarse con la herramienta QEMU que utilizaremos durante el curso. Con este fin se desarrollará un programa en C que resuelva el pedido de conteo de caracteres, palabras y líneas. Dicho programa también tendrá una documentación de ayuda para su uso. Se realizarán pruebas de funcionamiento y mediciones de tiempo de trabajo para la ejecución del programa.

Índice

66:20/86.37 Organización de Computadoras
Primer cuatrimestre de 2018
Trabajo práctico 0: Infraestructura básica

1. Objetivos

Familiarizarse con las herramientas de software que usaremos en los siguientes trabajos, implementando un programa (y su correspondiente documentación) que resuelva el problema piloto que presentaremos 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.

Además, es necesario que el trabajo práctico incluya (entre otras cosas, ver sección 8), la presentación de los resultados obtenidos, explicando, cuando corresponda, con fundamentos reales, las causas o razones de cada resultado obtenido.

El informe deberá respetar el modelo de referencia que se encuentra en el grupo[1], y se valorarán aquellos escritos usando la herramienta TEX / LATEX.

4. Recursos

Usaremos el programa QEMU [3] para simular el entorno de desarrollo que utilizaremos, una máquina MIPS corriendo una versión reciente del sistema operativo Debian [4].

En éste utilizaremos los programas `gcc` [1] y `time` para compilar y para examinar el desempeño de los programas a desarrollar.

[1] `https://github.com/stahlmatias/6620_/blob/master/Modelo_Informe/Modelo_Informe.tex`

5. El comando `wc`

El comando de Unix `wc` [2] toma como entrada un archivo o `stdin`, y cuenta las palabras, las líneas y la cantidad de caracteres que contiene.

6. Programas a desarrollar

El programa a escribir, en lenguaje C, recibirá un nombre de archivo que contiene texto (o el archivo mismo por `stdin`) e imprimirá por `stdout` la cantidad de líneas, palabras y caracteres que contiene, junto con el nombre del archivo.

6.1. Ejemplos

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

```
$ tp0 -h
Usage:
  tp0 -h
  tp0 -V
  tp0 [options] file
Options:
  -V, --version     Print version and quit.
  -h, --help        Print this information.
  -l, --words Print number of lines in file.
  -w, --words Print number of words in file.
  -c, --words Print number of characters in file.
  -i, --input       Path to input file.
Examples:

  tp0 -w -i input.txt
```

Luego, lo usamos con un pequeño fragmento de texto:

```
$echo -n "El tractorcito rojo que silbo y bufo" > entrada.txt
$tp0 -w -i entrada.txt
7 entrada.txt
$
```

7. Mediciones

Se deberá medir el tiempo insumido por el programa para el caso de los archivos `alice.txt`, `beowulf.txt`, `cyclopedia.txt` y `elquijote.txt`. Graficar el tiempo insumido contra el tamaño de muestra. Se deberá comprobar que el programa acepta las opciones dadas, y que reporta un error ante situaciones anómalas (como no encontrar el archivo solicitado). La ejecución del programa debe realizarse bajo el entorno MIPS.

8. Informe

El informe deberá incluir:

- Documentación relevante al diseño e implementación del programa;
- Las corridas de prueba, con los comentarios pertinentes;
- Gráficos de tiempo completo para los diferentes archivos, en función del tamaño de la entrada.
- El código fuente, en lenguaje C;
- Este enunciado.

9. Fecha de entrega

El cierre de este trabajo práctico es el jueves 23 de septiembre de 2022.

Referencias

[1] GCC, `https://gcc.gnu.org/onlinedocs/gcc-3.3.6/gcc/`

[2] Manual de `wc`, `http://man7.org/linux/man-pages/man1/wc.1.html`

[3] QEmu, `https://www.qemu.org/`.

[4] Debian, the Universal Operating System, `https://www.debian.org/`.

1. Introducción

En este trabajo inicial se quiere usar un procesador de arquitectura MIPS32. Para emular este entorno se utiliza la herramienta QEMU.

Para experimentar con este procesador se desarrolla un programa de finalidad similar a la de la función `wc`. El programa informa ayuda y versión. Como funciones se implementan el conteo de caracteres, palabras y líneas de archivo.

2. Desarrollo

Para este proyecto el programa a ejecutar se llama `tp0`. La función toma argumentos de entrada que indican la acción a realizar y los archivos de texto a leer.

La función debe informar ayuda, esto se puede indicar con dos argumentos:

```
-h
--help
```

La función debe informar versión:

```
-V
--version
```

Se le debe poder indicar a la función el tipo de conteo, la entrada y el nombre de la entrada, que sería el archivo de texto:

```
-c -i example.txt
--characes --input example.txt

-w -i example.txt
--words --input example.txt

-l -i example.txt
--lines --input example.txt
```

Por las funciones implementadas, el programa admite comandos con dos y cuatro argumentos. Dos argumentos se usan para pedir ayuda y el número de versión. Se necesitan cuatro argumentos para las funciones de conteo: nombre del programa, tipo de conteo, indicación de entrada y nombre de entrada.

La salida por pantalla indica el número resultado del conteo y el nombre del archivo sobre el cual se operó.

2.1. Desarrollo del Código Fuente

3. Pruebas

3.1. Comandos

Ingreso incorrecto de cantidad de argumentos

El programa necesita un mínimo de dos argumentos y un máximo de cuatro argumentos. El nombre del programa es uno de los argumentos.

```
# ./tp0
Cantidad incorrecta de argumentos

# ./tp0 -d -c -b -a
Cantidad incorrecta de argumentos
```

En el primer caso solo se ingresa un argumento. Luego de `tp0` no importa si se ingresan muchos espacios, la salida de error es la misma. En el segundo caso se ingresan cinco argumentos, lo que excede la cantidad máxima permitida.

Ingreso de argumentos inválidos

Cuando se ingresa la cantidad correcta de argumentos también se verifica que sean los correctos.

```
# ./tp0 -k
Las opciones ingresadas son incorrectas

# ./tp0 -c -i
Cantidad incorrecta de argumentos

# ./tp0 -c -v beowulf.txt
Las opciones ingresadas son incorrectas

# ./tp0 -b -i beowulf.txt
Las opciones ingresadas son incorrectas
```

En el primer comando la cantidad de argumentos es válida, pero el argumento ingresado no lo es. En el segundo comando los argumentos `-c -i` son correctos, pero falta el argumento con el nombre del archivo. El ingreso de tres argumentos cae en un error de cantidad de argumentos. Los últimos dos comandos tienen la cantidad correcta de argumentos para pedir el conteo, pero al menos uno es inválido.

Intento de lectura de archivo inexistente

```
# .\tp0 -w -i allice2.txt
El archivo que intenta abrir no existe
```

Aquí se intenta leer un archivo que no existe.

Comando de ayuda

```
# ./tp0 -h
Usage:
tp0 -h
tp0 -V
tp0 [options] file
Options:
-V, --version       Print version and quit.
-h, --help          Print this information.
-l, --lines         Print number of lines in file.
-w, --words         Print number of words in files.
-c, --characters    Print number of characters in file.
-i, --input         Path to input file.
Examples:

tp0 -w -i input.txt
```

Caso de prueba provisto

```
# echo -n "El tractorcito rojo que silbo y bufo" > entrada.txt
# ./tp0 -w -i entrada.txt
7 entrada.txt
#
```

3.2. Mediciones

Con el programa `tp0.c`, se midieron las cantidades de caracteres, palabras y líneas de los archivos señalados, cuyos resultados se vuelcan en la tabla a continuación:

Archivo	Cantidad de Caracteres	Cantidad de Palabras	Cantidad de Líneas
beowulf.txt	224839	37048	4562
elquijote.txt	2198907	384258	37862
alice.txt	177428	30355	4046
cyclopedia.txt	658543	105582	17926

Cuadro 1: Tabla cantidades para cada uno de los archivos, utilizando `tp0.c`

Se realizaron las mediciones de las funciones: `contar_letras`, `contar_palabras` y `contar_lineas` con el comando `time` para cada uno de los archivos dados, y se presentan los resultados a en las tablas a continuación:

Prueba	Archivo	Tiempo Real	Tiempo Usuario	Tiempo de Sistema
Melina	beowulf.txt	0m0.127s	0m0.100s	0m0.000s
David	beowulf.txt	0m0.326s	0m0.296s	0m0.024s
Facundo	beowulf.txt	0m0.197s	0m0.188s	0m0.000s
Melina	elquijote.txt	0m1.361s	0m1.312s	0m0.036s
David	elquijote.txt	0m1.717s	0m1.672s	0m0.044s
Facundo	elquijote.txt	0m1.333s	0m1.320s	0m0.000s
Melina	alice.txt	0m0.207s	0m0.176s	0m0.020s
David	alice.txt	0m0.303s	0m0.288s	0m0.012s
Facundo	alice.txt	0m0.175s	0m0.156s	0m0.012s
Melina	cyclopedia.txt	0m0.384s	0m0.348s	0m0.028s
David	cyclopedia.txt	0m0.638s	0m0.616s	0m0.020s
Facundo	cyclopedia.txt	0m0.469s	0m0.448s	0m0.016s

Cuadro 2: Tabla de tiempos para la función `contar_letras`

Prueba	Archivo	Tiempo Real	Tiempo Usuario	Tiempo de Sistema
Melina	beowulf.txt	0m0.228s	0m0.200s	0m0.020s
David	beowulf.txt	0m0.377s	0m0.348s	0m0.028s
Facundo	beowulf.txt	0m0.228s	0m0.200s	0m0.020s
Melina	elquijote.txt	0m1.044s	0m1.016s	0m0.020s
David	elquijote.txt	0m2.303s	0m2.260s	0m0.040s
Facundo	elquijote.txt	0m1.044s	0m1.016s	0m0.020s
Melina	alice.txt	0m0.169s	0m0.140s	0m0.020s
David	alice.txt	0m0.327s	0m0.296s	0m0.032s
Facundo	alice.txt	0m0.169s	0m0.140s	0m0.020s
Melina	cyclopedia.txt	0m0.616s	0m0.584s	0m0.020s
David	cyclopedia.txt	0m0.749s	0m0.724s	0m0.020s
Facundo	cyclopedia.txt	0m0.616s	0m0.584s	0m0.020s

Cuadro 3: Tabla de tiempos para la función `contar_palabras`

Prueba	Archivo	Tiempo Real	Tiempo Usuario	Tiempo de Sistema
Melina	beowulf.txt	0m0.192s	0m0.152s	0m0.032s
David	beowulf.txt	0m0.356s	0m0.324s	0m0.028s
Facundo	beowulf.txt	0m0.247s	0m0.232s	0m0.004s
Melina	elquijote.txt	0m1.173s	0m1.120s	0m0.048s
David	elquijote.txt	0m1.445s	0m1.400s	0m0.044s
Facundo	elquijote.txt	0m1.906s	0m1.884s	0m0.012s
Melina	alice.txt	0m0.138s	0m0.096s	0m0.032s
David	alice.txt	0m0.298s	0m0.268s	0m0.028s
Facundo	alice.txt	0m0.170s	0m0.156s	0m0.004s
Melina	cyclopedia.txt	0m0.327s	0m0.316s	0m0.004s
David	cyclopedia.txt	0m0.531s	0m0.480s	0m0.048s
Facundo	cyclopedia.txt	0m0.436s	0m0.416s	0m0.008s

Cuadro 4: Tabla de tiempos para la función `contar_lineas`

Se representó de manera gráfica la comparación de los tiempos encontrados para cada archivo y la cantidad de palabra, líneas y caracteres calculados, medidos por cada integrante del trabajo. Los mismos se presentan a continuación.

4. Bibliografía

5. Código fuente

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

#define error_cant_param "Cantidad incorrecta de argumentos"
#define error_param_incorrectos "Las opciones ingresadas son incorrectas"
#define error_archivo_inexistente "El archivo que intenta abrir no existe"
#define mensaje_de_ayuda "Usage:\n \ttp0 -h\n
\ttp0 -V\n\ttp0 [options] file\nOptions:\n
\t-V, --version\t\tPrint version and quit.\n
\t-h, --help\t\tPrint this information.\n
\t-l, --lines\t\tPrint number of lines in file.\n
\t-w, --words\t\tPrint number of words in files.\n
\t-c, --characters\tPrint number of characters in file.\n
\t-i, --input\t\tPath to input file.\n
Examples:\n\n\ttp0 -w -i input.txt"
#define version_num "tp0 v1.0"

// Muestra ayuda
void ayuda(){
    printf("%s\n",mensaje_de_ayuda);
}

// Muestra la versión del programa
void version(){
    printf("%s\n",version_num);
}

// Muestra error y termina ejecución
void error(char *mensaje){
    fprintf(stderr,"%s\n",mensaje);
    // fprintf se usa para escribir en stderr
    exit(0);
    // exit termina la ejecución del programa
}
```

```
char * string_token(char *str,const char *delim,char **temp)
{
    register char *tok;

    if(str!=NULL)
    {
        for(tok=str;*tok;tok++)
            if(strchr(delim,*tok)!=NULL)
                break;
            if(!(*tok))
                *temp=tok;
            else{
                *tok='\0';
                *temp=tok+1;
            }
    }else{
if(!(**temp))
return NULL;
str=*temp;
for(tok=str;*tok;tok++)
if(strchr(delim,*tok)!=NULL)
break;

if(!(*tok)){
*temp=tok;
return str;
}else{
*tok='\0';
*temp=tok+1;
}
}
return str;
}
```

```
char * clone_string (const char * orig)
{
size_t len=0;
size_t i=0;
    char * clone=NULL;

if(orig==NULL )
NULL;

len=strlen(orig);

if((clone=(char *)malloc(sizeof(char)*(len+1)))==NULL)
NULL;

for(i=0;i<=len;i++)
clone[i]=orig[i];

return clone;
}

void destroy_string (char *str){

if(str==NULL)
return;
free(str);
return;
}

void destroy_string_array(char **str_arr, size_t len)
{
size_t i=0;

if(str_arr==NULL)
return;

for(i = 0; i < len; i++)
{
free(str_arr[i]);
str_arr[i] = NULL;
}
free(str_arr);
str_arr=NULL;
return;
}
```

```
char ** split_string(char * str,char * delim, size_t *len)
{
char *dup, *q, *p,*temp;
size_t n, i;
    char ** str_arr;

if(str==NULL || delim==NULL || len==NULL)
return NULL;

if((dup=clone_string(str))==NULL)
return NULL;

for(n=0,i=0;dup[i];i++)
if(strchr(delim,dup[i])!=NULL)
n++;

if((str_arr=(char **)malloc(sizeof(char*)*(n+1)))==NULL){
destroy_string(dup);
*len=0;
return NULL;
}
for(i=0,q=dup;(p= string_token(q,delim,&temp))!=NULL;q=NULL,i++)
{
if((str_arr[i] =clone_string(p)) ==NULL)
{
destroy_string_array(str_arr, i);
destroy_string(dup);
*len=0;
return NULL;
}
}
destroy_string(dup);
*len=i;
return str_arr;
}
```

```
// Cuenta caracteres
int contar_caracteres(FILE *archivo){

    char ch;
    int contador = 0;

    while ((ch = fgetc(archivo)) != EOF){
        if ((ch != '\n') || (ch != ' ') || (ch != '\t') || (ch != '\0'))
            contador++;
    }

    fclose(archivo);
    return contador;
}

// Cuenta palabras
int contar_palabras(FILE *archivo){
    char linea[100 + 1];
    int i, cantidad_palabras = 0;
    char ** string_array=NULL;
    size_t len=0;
    char *delim= " \t\n\r";

    while (fgets(linea, 300, archivo) != NULL) {
        string_array=split_string(linea,delim, &len);
        for(i=0;i<len;i++){
            if(strlen(string_array[i])>0)
            cantidad_palabras++;
        }
    destroy_string_array(string_array,len);
    len=0;
    }

    return cantidad_palabras;
}
```

```
// Cuenta líneas
int contar_lineas(FILE *archivo){

    char ch;
    int contador = 0;

    while ((ch = fgetc(archivo)) != EOF){
        if (ch == '\n'){
            contador++;
        }

    }

    fclose(archivo);
    return contador;
}
```

```
int main(int argc, char *argv[]){
//argc -> argument count, argv -> argument vector

    // Puntero a archivo
    FILE *file;

    // Verifica la cantidad de argumentos
    if (argc < 2 || argc > 4){
        error(error_cant_param);
    }

    // Verifica por pedido de ayuda en el primer argumento
    else if ((strcmp(argv[1], "-h") == 0) ||
             (strcmp(argv[1], "--help") == 0)){
        // Por si hay algo luego del pedido de ayuda
        if (argc != 2) error(error_cant_param);
        ayuda();
        return 0;
    }

    // Verifica por pedido de versión en el primer argumento
    else if ((strcmp(argv[1], "-V") == 0) ||
             (strcmp(argv[1], "--version") == 0)){
        // Por si hay algo luego del pedido de versión
        if (argc != 2) error(error_cant_param);
        version();
        return 0;
    }

    // Verifica por pedido de caracteres en el primer argumento
    // y entrada en el segundo argumento
    else if (((strcmp(argv[1], "-c") == 0) ||
              (strcmp(argv[1], "--characters") == 0)) &&
             ((strcmp(argv[2], "-i") == 0) ||
              (strcmp(argv[2], "--input") == 0))){
        // Por si hay algo luego del nombre del archivo
        if (argc != 4) error(error_cant_param);

        // Se intenta abrir el archivo en modo lectura
        file = fopen(argv[3], "r");
        if (file == NULL) error(error_archivo_inexistente);

        int num;
        num = contar_caracteres(file);
        printf("%d %s\n", num, argv[3]);
        return 0;
    }

    // Verifica por pedido de palabras en el primer argumento
    // y entrada en el segundo argumento
    else if (((strcmp(argv[1], "-w") == 0) ||
              (strcmp(argv[1], "--words") == 0)) &&
             ((strcmp(argv[2], "-i") == 0) ||
              (strcmp(argv[2], "--input") == 0))){
        // Por si hay algo luego del nombre del archivo
        if (argc != 4) error(error_cant_param);

        // Se intenta abrir el archivo en modo lectura
        file = fopen(argv[3], "r");
        if (file == NULL) error(error_archivo_inexistente);
```

```
        int num;
        num = contar_palabras(file);
        printf("%d %s\n", num, argv[3]);
        return 0;
    }

    // Verifica por pedido de líneas en el primer argumento
    // y entrada en el segundo argumento
    else if (((strcmp(argv[1], "-l") == 0) ||
              (strcmp(argv[1], "--lines") == 0)) &&
             ((strcmp(argv[2], "-i")) ||
              (strcmp(argv[2], "--input")))){
        // Por si hay algo luego del nombre del archivo
        if (argc != 4) error(error_cant_param);

        // Se intenta abrir el archivo en modo lectura
        file = fopen(argv[3], "r");
        if (file == NULL) error(error_archivo_inexistente);

        int num;
        num = contar_lineas(file);
        // Se convierte num a decimal
        printf("%d %s\n", num, argv[3]);
        return 0;
    }

    // Opciones ingresadas incorrectas
    else {
        error(error_param_incorrectos);
    }

    return 0;
}
```