

Trabajo Práctico 0: Infraestructura básica

Jimenez, Ruben, *Padrón Nro. xx.xxx*
rbnm.jimenez@gmail.com

Reyero, Felix, *Padrón Nro. xx.xxx*
felixcarp@gmail.com

Suárez, Emiliano, *Padrón Nro. xx.xxx*
emilianosuarez@gmail.com

Primera Entrega: *26/03/2015*

1er. Cuatrimestre de 2015

66.20 Organización de Computadoras – Práctica Jueves
Facultad de Ingeniería, Universidad de Buenos Aires

Resumen

Se implementó una versión minimalista del comando **tail** de UNIX.
Para nuestra implementación.

1. Introducción

En este Trabajo Práctico se pretende familiarizarse con las herramientas de software que usaremos en los siguientes trabajos.

Para ello, implementaremos la función **tail** de UNIX para luego hacer un profiling y compararla con la versión nativa. Para esto, utilizaremos la herramienta *gprof* que nos permitirá identificar que parte del código podría ser mejorada.

Además, se utilizará *GXemul* para simular una máquina *MIPS* corriendo una versión reciente del sistema operativo *NetBSD*.

El programa implementado, escrito en lenguaje *C*, muestra por *stdout* las últimas líneas del contenido de uno o mas archivos. En caso de no especificarse algún archivo, se mostrarán las últimas las líneas que se reciban por *stdin*.

2. Diseño e Implementación

Se implementó un programa que realiza la lectura de líneas a través del *stdin* y la lectura desde archivos.

El comando acepta 2 parámetros para mostrar la Ayuda y la Versión del programa:

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

Para desplegar la ayuda del comando. Y los siguientes comandos para mostrar la versión:

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

Inicialmente el programa revisa la cadena de parametros ingresada y determina que tipo operación debe realizar.

En caso de no recibir los parámetros antes mencionados, se lee desde *stdin* si no se reciben parámetros, o desde los archivos que se le pasen al programa.

Para esta última opción, se abren los archivos de uno por vez, y para cada uno de ellos se procede a leer cada línea y mostrar las líneas correspondientes.

2.1. Versión 1

Dado que el programa se basa en mostrar las últimas *n* líneas de un archivo, primero se comenzó implementando un buffer para almacenar las líneas pedidas por el usuario pero, debido a que el buffer tiene un tamaño fijo resultó ser poco viable dado que las líneas no tienen un tamaño definido, y pude llegar a perderse datos.

2.2. Versión 2

Como segunda opción se decidió desarrollar el programa de forma que, dado un archivo, el programa se posiciona al final del mismo y comienza a recorrer desde el final al inicio contando la cantidad de líneas o caracteres, e imprimir cuando se encuentre la cantidad de líneas o caracteres deseados.

Esto solucionaba el problema con los buffers dado que había forma de perder algún dato del archivo, al utilizar un tamaño fijo.

2.3. Versión 3

A pesar de la mejora de la segunda versión, un nuevo problema surgió cuando se usa la entrada estándar como archivo.

Dado que *UNIX* almacena en un buffer los datos ingresados por *stdin* y cuando se escribe el fin de línea los datos del buffer pasa a ser procesado por el programa como si fuese con un archivo, Para lo cual se decidió almacenar los datos de entrada en un archivo auxiliar que será eliminado al final del programa, para procesarlo como cualquier archivo de entrada.

3. Comando para compilar el programa

Para compilar el programa se debe abrir una terminal en la carpeta donde están alojados los archivos fuentes (*src/*) y se ejecuta el siguiente comando:

```
../src$ make
../src$ make % borrar esta línea
```

Para generar el ejecutable *tp0*.

make: se encargara de compilar los archivos generando el ejecutable.

El *Makefile* puede observarse a continuación:

```
1 CC=gcc -g -O0
2 CFLAGS=-c -Wall
3 LDFLAGS=
4 OBJ_DIR=/
5 SOURCES=main.c
6
7 OBJECTS=$(SOURCES:.c=.o)
8 EXECUTABLE=tp0
9
10 all: $(SOURCES) $(EXECUTABLE)
11
12 $(EXECUTABLE): $(OBJECTS)
13     $(CC) $(LDFLAGS) $(OBJECTS) -o $@
14
15 .c.o:
16     $(CC) $(CFLAGS) $< -o $@
17
18 clean:
19     rm -rf $(EXECUTABLE) $(OBJECTS)
```

../src/Makefile

La compilación del programa en NetBSD (asegurando la portabilidad), puede observarse en la figura ?? del Apéndice.

4. Casos de Prueba

Se realizaron distintas pruebas:

```
root@ruben-Lenovo-G460: /home/ruben
root@~/tail# ./tail -n 10
1
2
3
4
5
6
7
8
9
10^D
****las ultimas lineas escritas****
1
2
3
4
5
6
7
8
9
10
¡Fin del programa!
root@~/tail#
```

Figura 1: Archivo con líneas iguales a las pedidas

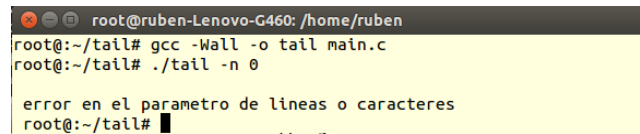
```
root@~/tail# ./tail -n 10
prueba cuando
el archivo
tiene menos lineas
de las que
pide el usuario.^D
error la cantidad de lineas/caracteres pedidas es mayor a la que contiene el archivo
root@~/tail#
```

Figura 2: Archivo con menos líneas que las pedidas

```
root@~/tail# ./tail -c 100
test archivo con caracteres menor a las pedidas
por el usuario.^D
error la cantidad de lineas/caracteres pedidas es mayor a la que contiene el archivo
root@~/tail#
```

Figura 3: Cantidad de caracteres pedidos, mayores a los disponibles en el archivo

Figura 4: Comando equivocado

A terminal window with a dark title bar showing 'root@ruben-Lenovo-G460: /home/ruben'. The terminal has a yellow background. It shows the following commands and output:

```
root@:~/tail# gcc -Wall -o tail main.c
root@:~/tail# ./tail -n 0

error en el parametro de líneas o caracteres
root@:~/tail#
```

Figura 5: Parámetro incorrecto

5. Mediciones

6. Profiling

El tamaño de muestra para hacer un profiling para optimizar un programa debería ser grande o muy grande, dado que una computadora ejecuta procesos en tiempos demasiados cortos, lo cual no nos daría una medición exacta del tiempo insumido por cada función del programa, y el porcentaje sería muy parecido para todos las funciones. Por lo tanto para la prueba se trabaja con un archivo de 90MB que cuenta con aproximadamente 1.600.000 líneas y se le pedirá el último millón de líneas (-n 1000000).

Se obtuvieron los resultados que se muestran a continuación:

```
ruben@ruben-Lenovo-G460:~/Escritorio/tail$ gprof profiler
Flat profile:
Each sample counts as 0.01 seconds.
```

	%	cumulative	self	calls	self	total	
time	seconds	seconds	seconds	ms/call	ms/call	name	
52.94	0.19	0.19	1	190.59	361.11	tail	
47.37	0.36	0.17	1	170.53	170.53	imprimir	
0.00	0.36	0.00	1	0.00	361.11	evaluarArgumentos	
0.00	0.36	0.00	1	0.00	0.00	posicionaAlInicioLinea	
0.00	0.36	0.00	1	0.00	0.00	validacionEntero	

Figura 6: Descripción de Imagen 1

%	the percentage of the total running time of the
time	program used by this function.
cumulative	a running sum of the number of seconds accounted
seconds	for by this function and those listed above it.
self	the number of seconds accounted for by this
seconds	function alone. This is the major sort for this
	listing.
calls	the number of times this function was invoked, if
	this function is profiled, else blank.
self	the average number of milliseconds spent in this
ms/call	function per call, if this function is profiled,
	else blank.
total	the average number of milliseconds spent in this
ms/call	function and its descendents per call, if this
	function is profiled, else blank.
name	the name of the function. This is the minor sort
	for this listing. The index shows the location of
	the function in the gprof listing. If the index is
	in parenthesis it shows where it would appear in
	the gprof listing if it were to be printed.

Figura 7: Descripción de Imagen 2

Según los resultados, la función que mayor tiempo consume es tail con un 52,94% del tiempo total.

Dado que:

$$Su = \frac{T_{old}}{T_{new}}$$

$$Su = \frac{1}{(1 - fm + \frac{fm}{sl})}$$

Call graph (explanation follows)					
granularity: each sample hit covers 2 byte(s) for 2.77% of 0.36 seconds					
index	% time	self	children	called	name
		0.00	0.36	1/1	main [3]
[1]	100.0	0.00	0.36	1	evaluarArgumentos [1]
		0.19	0.17	1/1	tail [2]
		0.00	0.00	1/1	validacionEntero [6]

		0.19	0.17	1/1	evaluarArgumentos [1]
[2]	100.0	0.19	0.17	1	tail [2]
		0.17	0.00	1/1	imprimir [4]
		0.00	0.00	1/1	posicionaAlInicioLinea [5]

					<spontaneous>
[3]	100.0	0.00	0.36		main [3]
		0.00	0.36	1/1	evaluarArgumentos [1]

		0.17	0.00	1/1	tail [2]
[4]	47.2	0.17	0.00	1	imprimir [4]

		0.00	0.00	1/1	tail [2]
[5]	0.0	0.00	0.00	1	posicionaAlInicioLinea [5]

		0.00	0.00	1/1	evaluarArgumentos [1]
[6]	0.0	0.00	0.00	1	validacionEntero [6]

Figura 8: Descripción de Imagen 3

This table describes the call tree of the program, and was sorted by the total amount of time spent in each function and its children.	
Each entry in this table consists of several lines. The line with the index number at the left hand margin lists the current function. The lines above it list the functions that called this function, and the lines below it list the functions this one called.	
This line lists:	
index	A unique number given to each element of the table. Index numbers are sorted numerically. The index number is printed next to every function name so it is easier to look up where the function is in the table.
% time	This is the percentage of the 'total' time that was spent in this function and its children. Note that due to different viewpoints, functions excluded by options, etc, these numbers will NOT add up to 100%.
self	This is the total amount of time spent in this function.
children	This is the total amount of time propagated into this function by its children.
called	This is the number of times the function was called. If the function called itself recursively, the number only includes non-recursive calls, and is followed by a '+' and the number of recursive calls.
name	The name of the current function. The index number is printed after it. If the function is a member of a cycle, the cycle number is printed between the function's name and the index number.

Figura 9: Descripción de Imagen 4

For the function's parents, the fields have the following meanings:

self	This is the amount of time that was propagated directly from the function into this parent.
children	This is the amount of time that was propagated from the function's children into this parent.
called	This is the number of times this parent called the function '/' the total number of times the function was called. Recursive calls to the function are not included in the number after the '/'.
name	This is the name of the parent. The parent's index number is printed after it. If the parent is a member of a cycle, the cycle number is printed between the name and the index number.

If the parents of the function cannot be determined, the word '<spontaneous>' is printed in the 'name' field, and all the other fields are blank.

Figura 10: Descripción de Imagen 5

For the function's children, the fields have the following meanings:

self	This is the amount of time that was propagated directly from the child into the function.
children	This is the amount of time that was propagated from the child's children to the function.
called	This is the number of times the function called this child '/' the total number of times the child was called. Recursive calls by the child are not listed in the number after the '/'.
name	This is the name of the child. The child's index number is printed after it. If the child is a member of a cycle, the cycle number is printed between the name and the index number.

If there are any cycles (circles) in the call graph, there is an entry for the cycle-as-a-whole. This entry shows who called the cycle (as parents) and the members of the cycle (as children.) The '+' recursive calls entry shows the number of function calls that were internal to the cycle, and the calls entry for each member shows, for that member, how many times it was called from other members of the cycle.

Figura 11: Descripción de Imagen 6

$$Su_{max} = \lim_{sl \rightarrow \infty} SU = \frac{1}{(1 - fm)}$$

Como:

$$fm = 0,5294$$

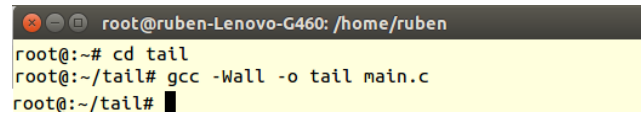
Entonces:

$$Su_{max} = \frac{1}{0,4706} = 2,12$$

7. Conclusiones

8. Apéndice

8.1. Compilación en NetBSD

A terminal window with a dark title bar containing window control icons and the text 'root@ruben-Lenovo-G460: /home/ruben'. The terminal has a yellow background and displays three lines of text: 'root@:~# cd tail', 'root@:~/tail# gcc -Wall -o tail main.c', and 'root@:~/tail#' followed by a black cursor block.

```
root@ruben-Lenovo-G460: /home/ruben
root@:~# cd tail
root@:~/tail# gcc -Wall -o tail main.c
root@:~/tail# █
```

Figura 12: Compilación NetBSD

8.2. Código Fuente: tp0.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #define ayuda 'h'
4 #define version 'V'
5 #define caracteres 'c'
6 #define lineas 'n'
7 #define cantLineasPorDefecto 10
8 #define maxcantArchivos 50
9 typedef int bool;
10 #define true 1
11 #define false 0
12
13
14 /***** $tp0 -c 10 -n 2 <prueba.txt *** /
15
16
17 void evaluarArgumentos();
18 void validacionEntero(int numero);
19 void imprimirAyuda();
20 void imprimirVersion();
21 void tail(int i, char* archivos[], int cantArchivos, bool caso);
22 void imprimir(FILE* fp, unsigned pos);
23 long int posicionaAlInicioLinea(FILE *fp, long int pos);
24
25 //char* getLinea(FILE *fp);
26
27 int main (int argc, char *argv[])
28 {
29     evaluarArgumentos(argc, argv);
30     printf("¡Fin del programa! \n");
31     return 0;
32 }
33
34
35 void evaluarArgumentos(int argc, char *argv[])
36 {
37     int i = 0;
38     char *nombresArchivos[maxcantArchivos];
39     int cantArchivos = 0;
40     bool LineasPorDefecto = true;
41     int cantOpcionesTail = 0;
42
43
44     for(i=1; i<argc; i++) /*buscamos los
45                            archivos*/
46     {
47         if (*argv[i] == '-') //descartamos las
48             opciones
49         {
50             switch ( *(argv[i] +1) )
51             {
52                 case caracteres:
53                     i++; //si es caracter
54                     descartamos su parametro k
55                     cantOpcionesTail++;
56                     break;
57
58                 case lineas: //si es linea
59                     descartamos su parametro k
60                     cantOpcionesTail++;
61             }
62         }
63     }
64 }
```

```

57         i++;
58         break;
59     }
60 }
61 else
62 {
63     nombresArchivos[cantArchivos++] = argv[i];
64 }
65 }
66
67 if(cantOpcionesTail > 1) // solo puede haber una opcion de
    Tail caracter o linea
68 {
69     char* error_doble_funcionalidad = "\n error en los
    parametros, usar -n o -c pero no ambos \n";
70     fputs(error_doble_funcionalidad, stderr);
71     exit(1);
72 }
73
74
75 for (i = 1; i < argc; i++) //no contamos el argumento cero
    dado que es el programa, ni el ultimo porque es el
    archivo
76 {
77     if (*argv[i] == '-')
78         switch ( *(argv[i] + 1) )
79         {
80             case ayuda:
81                 imprimirAyuda();
82                 exit(1);
83                 break;
84
85             case version:
86                 imprimirVersion();
87                 exit(1);
88                 break;
89
90             case caracteres:
91                 validacionEntero(atoi(argv[i+1]));
92                 tail(atoi(argv[i+1]), nombresArchivos,
                    cantArchivos, false);
93                 LineasPorDefecto = false;
94                 break;
95
96             case lineas:
97                 validacionEntero(atoi(argv[i+1]));
98                 tail(atoi(argv[i+1]), nombresArchivos,
                    cantArchivos, true);
99                 LineasPorDefecto = false;
100                 break;
101
102             default:
103                 fputs("\n error la opcion del menu no
                    es correcta: ayuda\n", stderr);
104                 imprimirAyuda(); //imprimir ayuda en
                    caso de parametros incorrectos
105                 exit(1);
106                 break;
107         }
108     }
109
110     if(LineasPorDefecto == true)

```

```

111         tail(cantLineasPorDefecto, nombresArchivos, cantArchivos,
112              true);
113     }
114
115
116 void tail(int cantLineaschar, char* archivos[], int cantArchivos, bool
117          caso)
118 {
119     FILE *fp;
120     int i = 0;
121     char* error_parametro = "\n error la cantidad de lineas/
122                             caracteres pedidas es mayor a la que contiene el archivo \n
123                             ";
124     do
125     {
126         if (cantArchivos < 1)
127         {
128             char c;
129             fp = fopen("archivoAuxiliar.txt", "w+");
130             while( (c = getc(stdin)) != EOF)
131             {
132                 fputc(c, fp);
133             }
134             fputc('\n', fp);
135         }
136         else
137         {
138             if ((fp = fopen(archivos[i], "r")) == NULL)
139             {
140                 fputs("\n error el archivo no existe\n",
141                      stderr);
142                 exit(1);
143             }
144             fputs("\n Archivo ", stdout);
145             fputs(archivos[i], stdout);
146         }
147
148         if (caso == true) //lineas
149         {
150             fseek(fp, 0, SEEK_END);
151             long int posFinal = ftell(fp);
152             char finDeLinea = '\n';
153             int cantLineasLeidas = 0;
154
155             while(cantLineasLeidas != cantLineaschar &&
156                  posFinal != 0)
157             {
158                 if (fgetc(fp) == finDeLinea)
159                 {
160                     cantLineasLeidas++;
161                 }
162
163                 if (fseek(fp, --posFinal, 0) != 0)
164                 {
165                     fputs("\n error al acceder al
166                             archivo \n", stderr);
167                     exit(1);

```

```

166         }
167     }
168
169     if(cantLineasLeidas != cantLineaschar)
170     {
171         fputs(error_parametro, stderr);
172         exit(1);
173     }
174     else
175     {
176         posFinal = posicionaAlInicioLinea(fp,
177             posFinal);
178         imprimir(fp, posFinal);
179     }
180
181     else
182     {
183         fseek(fp, 0, SEEK_END);
184         long int posFinal = ftell(fp);
185         int cantCaracteresLeidos = -1; /**ver esto**/
186
187         while(cantCaracteresLeidos != cantLineaschar &&
188             posFinal != 0)
189         {
190             cantCaracteresLeidos++;
191             --posFinal;
192         }
193
194         if(cantCaracteresLeidos != cantLineaschar)
195         {
196             fputs(error_parametro, stderr);
197             exit(1);
198         }
199         else
200         {
201             if(posFinal >= 0)
202             {
203                 imprimir(fp, posFinal);
204             }
205             else
206             {
207                 fputs("error al acceder al
208                     archivo \n", stderr);
209                 exit(1);
210             }
211         }
212     }
213
214     i++;
215     fclose(fp);
216     remove("archivoAuxiliar.txt");
217 }
218
219 while(i < cantArchivos);
220 }
221
222 void imprimirAyuda()
223 {
224     printf("\n
225     *****\n "

```



```

224     );
225     printf(" Opciones: \n");
226     printf(" Para caracteres usar —> ./programa -c numero
    archivos \n");
227     printf(" Para lineas usar —> ./programa -n numero archivos \n"
    );
228     printf(" Para la version usar —> ./programa -V archivos \n");
229     printf(" Los numeros ingresados tienen que ser mayores que cero
    \n");
230     printf("
    ***** \n"
    );
231 }
232 void imprimirVersion()
233 {
234     printf("\n Version 3.0 \n ");
235 }
236
237 void validacionEntero(int numero)
238 {
239     if(numero <= 0)
240     {
241         char* salida = "\n error en el parametro de lineas o
    caracteres \n ";
242         fputs(salida, stderr);
243         exit(1);
244     }
245 }
246
247
248 void imprimir(FILE* fp, unsigned pos)
249 {
250     printf("\n ****las ultimas lineas escritas**** \n");
251     fseek(fp, pos, 0);
252     char character;
253     while( (character = fgetc(fp)) != EOF)
254     {
255         fputc(character, stdout);
256     }
257 }
258
259
260 long int posicionaAlInicioLinea(FILE *fp, long int pos)
261 {
262     fseek(fp, pos, 0);
263     char character;
264     char finLinea = '\n';
265     while( (character = fgetc(fp)) != finLinea && pos !=0 )
266     {
267         fseek(fp, --pos, 0);
268     }
269
270     if (pos ==0)
271     {
272         return pos;
273     }
274     else
275     {
276         return pos+1;
277     }
278 }

```

```
279 | }  
280 |
```

../src/tp0.c

8.3. tp0.S