Facultad de Ingeniería - U.B.A.

66.20 Organización de Computadoras - Práctica Martes

2do. Cuatrimestre de 2017

Trabajo práctico Nº 1

Programación MIPS

Matias Leandro Feld, Padrón: 99170
feldmatias@gmail.com

Federico Funes, Padrón: 98372
fede.funes96@gmail.com

Agustín Zorzano, Padrón: 99224
aguszorza@gmail.com

1. Documentación e implementación

El objetivo del trabajo es realizar un programa en lenguaje MIPS32 que lea palabras de una archivo (o de entrada estándar) y guarde en otro archivo (mostrar por salida estándar) únicamente aquellas palabras que sean palíndromos. Además, para analizar como influyen en el tiempo de ejecución las lecturas y escrituras en archivos, se implementó un sistema de buffer.

Esto significa que al leer de un archivo no se hará de a un caracter por vez, sino que se llenará el buffer de entrada y luego se leerán los caracteres desde éste.

Asimismo, para la escritura de archivos se realizará algo similar. Se guardarán en el buffer los caracteres a escribir, y se escribirán en el archivo una vez que el buffer se llene. De este modo, variando el tamaño del buffer, se podrá analizar como afectan al tiempo de ejecución las operaciones con archivos.

El programa se divide en las siguientes funciones:

1. La función principal, main, que se encargará de la lógica de leer los parámetros de entrada y el manejo de los archivos. Si algun archivo no se puede abrir, no se pasaron correctamente los parámetros, o se produjo un error en la ejecución, el programa mostrará un mensaje de error en el archivo stderr y finalizará con un código de error. Esta funcion será escrita en lenguaje C.

2. La función palindrome, que consiste en leer una palabra del archivo de entrada, comprobar si es palíndromo y escribirla en el archivo de salida si corresponde. Ésta es la función de entrada al programa en MIPS que deberá ser llamada desde el programa en C. Recibe por parámetro el archivo de entrada, el de salida y los tamaños de los buffer. Al ser llamada lo primero que hará es crear los buffer de entrada y salida, utilizando la función crear_buffer(). Luego entrará en el bucle hasta que todos los caracteres del archivo de entrada sean analizados. El bucle termina cuando se lee el EOF, y en este caso se llamará una vez más a la función que escribe en archivos para escribir todo lo que haya quedado en el buffer de salida. El stackframe correspondiente a esta función quedará definido de la siguiente manera:

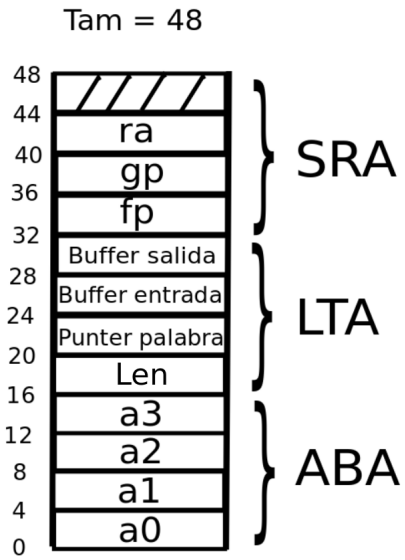

Figura 1: Stackframe de palindrome

3. La función leer_palabra, que se encarga de leer una palabra del archivo. Debido a las limitaciones de lo que se considera palabra, y a que no hay limitación con respecto a cantidad

de letras de una palabra, lo que hacemos es leer carácter por carácter, guardándolos en un vector alojado en memoria dinámica que se irá redimensionando a medida que sea necesario. Para ello, definimos una variable TAM que determinará la cantidad de memoria que se pide al inicio y al redimensionar. En principio esa variable puede contener cualquier número, pero para no estar redimensionando muchas veces y para no pedir mucha memoria innecesaria, definimos ese valor en 30. La función recibe por parámetro un puntero a entero, que sirve para guardar la longitud de la palabra leída, con el objetivo de no tener que calcularla nuevamente en otro momento. Para leer un caracter del archivo llamará a la función getch(). Para facilitar la escritura de la palabra en el archivo de salida, al final de cada palabra se insertará un \n en lugar de un \0, ya que \n no es considerado un caracter, y además necesitamos imprimirlo luego de cada palabra. El stackframe correspondiente a la función quedará definido de la siguiente manera:

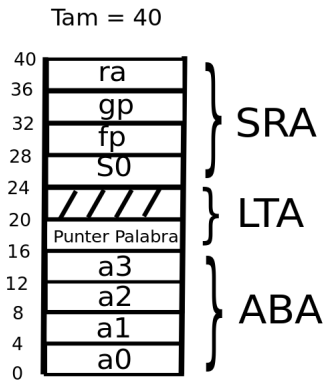

Figura 2: Stackframe de leer_palabra

4. La función es_capicúa, que se encarga de comprobar si la palabra es o no un palíndromo, y devuelve un valor booleano según corresponda. Ésta función recibe por parámetro el puntero a la palabra y la longitud de la misma. El stackframe correspondiente a esta función quedará definido de la siguiente manera:

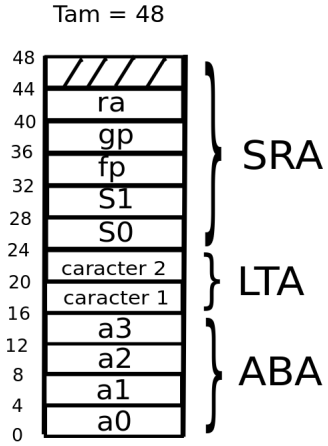

Figura 3: Stackframe de es_capicua

5. La función my_tolower, que fue implementada para reemplazar la del lenguaje C, se encarga de pasar a minúscula un caracter. Para eso, recibe por parámetro el caracter, y lo transforma únicamente si es una letra mayúscula, caso contrario lo devuelve como viene. El stackframe correspondiente a esta función quedará definido de la siguiente manera:

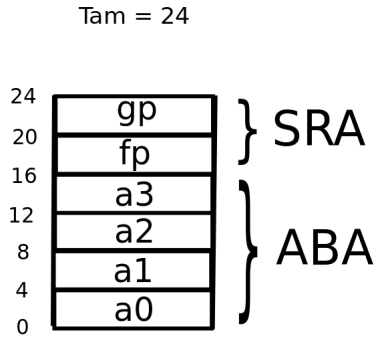

Figura 4: Stackframe de my_tolower

6. La función crear_buffer, es la encargada de crear los buffers. Para ello recibirá por parámetro el tamaño del mismo, y lo creará haciendo uso de la función mymalloc. Como resultado devuelve el puntero al buffer correspondiente. El stackframe correspondiente a esta función quedará definido de la siguiente manera:

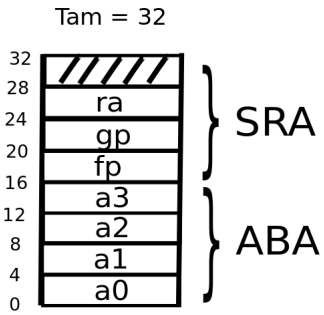

Figura 5: Stackframe de crear_buffer

7. La función getch, que se encarga de leer un carácter del archivo de entrada. Como se explicó anteriormente, ésta hace uso de un buffer. Por lo tanto, conociendo el tamaño del buffer y la última posición leída, devolverá el caracter correspondiente, y cuando la posición sea mayor o igual al tamaño se encargará de llenar el buffer nuevamente con nuevos datos. Esta función tiene una complicación adicional, ya que debe indicar cuando fue leído el final del archivo en el buffer. Para eso, utilizaremos una variable global, que será nula hasta el momento en que se lee el EOF, que cambiará de valor y permitirá avisar a las demás funciones que ya se leyó todo el archivo. Si se produjera algún error en la lectura devolverá un código de error. Para la lectura del archivo hace uso de un syscall, puede ocurrir que se lean menos bytes de los pedidos, en ese caso pueden ser por dos razones, que no hay más por leer o que se leyó menos pero se puede leer más. Esto lo solucionamos haciendo que la lectura se haga en un loop, que termina cuando no hay más para leer o cuando se llenó el buffer. Cuando se lee el EOF, para poder distinguir si quedan o no caracteres en el buffer para analizar, lo que hacemos es actualizar la variable eof_leido, que indica que no hay más para leer pero todavía hay cosas en el buffer. Cuando se intente leer nuevamente del archivo, si ésta variable es distinta de cero, significa que no hay más cosas en el buffer, por lo que se actualiza la variable escribir_eof y se devuelve cero. El stackframe correspondiente a esta función quedará definido de la siguiente manera:

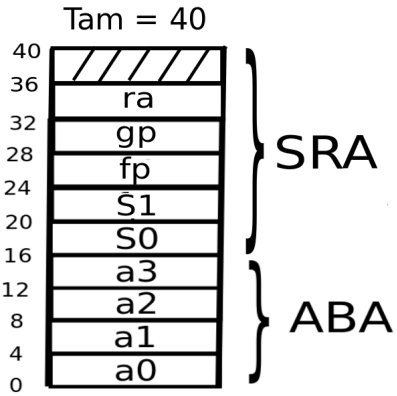

Figura 6: Stackframe de getch

8. La función putch, que se encarga de escribir una palabra en el archivo de salida. Debido a que debe utilizar el buffer, la función recibirá por parámetro la palabra, y guardará de a un caracter por vez en el buffer. Una vez que se llene el buffer, independientemente si se guardó toda la palabra o no, éste se escribirá en el archivo y se vaciará. Al igual que la anterior, también tiene una complicación. Puede ocurrir que el buffer no se llene completamente y se haya terminado el archivo, en cuyo caso, utilizando la variable global que indica si se debe escribir el EOF, escribirá todo lo que se encuentre en el buffer en ese momento. También, puede ocurrir que el syscall no escriba el total de los bytes pedidos, por lo que la escritura se realiza en un loop hasta que escriba todo. El stackframe correspondiente a esta función quedará definido de la siguiente manera:

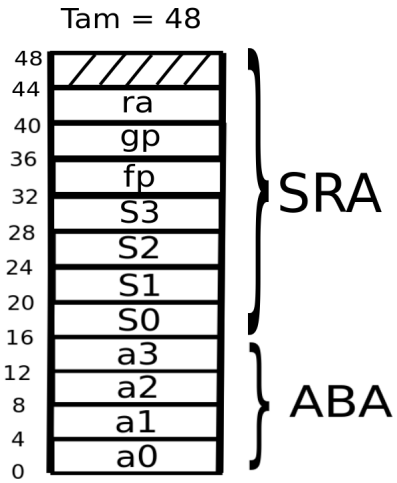

Figura 7: Stackframe de putch

9. Por último, la función myrealloc, que sirve para redimensionar un bloque de memoria dinámica. Para facilitar la programación de la misma, y porque no lo necesitamos, se decidió que solo se podrá redimensionar aumentando el tamaño del bloque y no disminuyéndolo. Por eso, la función recibe por parámetro el puntero al bloque, el tamaño actual, y el tamaño a agregar. Haciendo uso de la función mymalloc crea un nuevo bloque y copia byte por byte los datos del bloque viejo al nuevo. Finalmente libera el bloque viejo y devuelve el nuevo. Si se produjera un error al llamar a la función mymalloc se devolverá un puntero a NULL. El stackframe correspondiente a esta función quedará definido de la siguiente manera:

Figura 8: Stackframe de myrealloc

1.1. Variables globales

A continuación se explican las variables globales utilizadas y cómo se las interpreta.

1. pos_buffer_entrada: Indica la posición del próximo caracter a leer en el buffer de entrada. Se incrementa en uno cada vez que se lee un caracter y se reinicia a cero cuando se vuelve a llenar el buffer. Esta variable comienza seteada en -1, que sin signo es el número más alto, para que se produzca la primer lectura del archivo.

2. pos_buffer_salida: Indica la posición del próximo caracter a escribir en el buffer de salida. Se incrementa en uno cada vez que se escribe un caracter y se reinicia a cero cuando se realiza la escritura del archivo y se vacía el buffer.

3. tam_buffer_entrada: Indica el tamaño del buffer de entrada. Se inicializa en la creación del buffer con su valor correspondiente. Solo se actualiza cuando se realiza la última lectura (que lee menos bytes) para saber cuántos bytes quedan.

4. tam_buffer_salida: Indica el tamaño del buffer de salida. Se inicializa en la creación del buffer con su valor correspondiente. Solo se actualiza cuando se debe realizar la última escritura (que escribe menos bytes) para saber cuántos bytes quedan en el buffer.

5. eof_leido: Permite saber cuando se alcanzó el final del archivo de lectura y que por lo tanto no hay que hacer más lecturas. Su valor es 0 hasta el momento que se hayan leído todos los caracteres del archivo de entrada, hayan sido analizados todos los caracteres o no.

6. escribir_eof: Indica que ya se analizaron todos los caracteres del archivo, y que por lo tanto debe escribirse todo lo que queda en el buffer al archivo y finalizar el programa. Es 0 hasta el momento en que se analizan todos los caracteres.

7. TAM: Es el tamaño del bloque de memoria que se crea para cada palabra, y también el tamaño a agregarle al redimensionar.

2. Comandos para compilacion

Para compilar el programa utilizamos el siguiente comando:

$ gcc -Wall -o tp1 main.c mymalloc.S myrealloc.S palindrome.S getch.S putch.S crear_buffer.S leer_palabra.S es_capicua.S my_tolower.S

o se puede optar por ejecutar el script de bash:

$ bash compilar.sh

3. Pruebas

Para probar el programa utilizamos un script de bash llamado 'pruebas.sh' que contiene un conjunto de pruebas que se realizan automáticamente. Entre ellas,se encuentran pruebas con archivos vacios, archivos con un solo caracter y archivos solo con simbolos. Por otro lado, también se prueba que funcionen correctamente los mensajes de error cuando los parámetros no son usados correctamente. Se realizan pruebas para distintos tamaños de buffer para asegurarnos que funcione correctamente. Todas las pruebas utilizan el siguiente comando:

$ diff salida.txt resultado.txt

Donde si no muestra nada significa que ambos archivos son iguales, y que por lo tanto todas las pruebas del programa funcionan correctamente.

En algunas de las pruebas utilizamos un archivo de texto "entrada.txt" que contiene un conjunto de palabras con combinaciones de letras, numeros y guiones y mezclando mayúsculas y minúsculas. Luego tenemos otro archivo, "resultado.txt" que es lo que se espera que devuelva el programa al ejecutarse con ese archivo de entrada. En la siguiente sección se muestran esos archivos. Por otro lado, también se realizan pruebas con un archivo "archivo_largo.txt", que contiene 30 líneas de 5000 caracteres cada una, y donde además todas son palindromos. En el resto de las pruebas se usan archivos creados dentro del mismo script, que se borran al finalizar.

También realizamos pruebas utilizando salida estándar y entrada estándar, los cuales funcionaron correctamente. Cuando se trabaja con entrada estándar y se desea finalizar se debe ingresar "ctrl D", que inserta un EOF, ya que utilizando "ctrl C" finaliza abruptame nte y no se guarda correctamente el resultado.

El script de pruebas se puede ejecutar con el comando:

$ bash pruebas.sh

3.1. Archivo 'pruebas.sh'

```
1  #/bin/bash
2
3  bash compilar.sh
4
5
6  # Pruebas con archivo de pruebas entrada.txt y resultado.txt
7
8  ./tp1 -i entrada.txt -o salida.txt -I 1 -O 1
9  diff salida.txt resultado.txt
10 ./tp1 -i entrada.txt -o salida.txt -I 20 -O 20
11 diff salida.txt resultado.txt
12 ./tp1 -i entrada.txt -o salida.txt -I 100 -O 100
13 diff salida.txt resultado.txt
```

```
14 ./tp1 -i entrada.txt -o salida.txt -I 1000 -O 1000
15 diff salida.txt resultado.txt
16 ./tp1 -i entrada.txt -o salida.txt -I 20 -O 100
17 diff salida.txt resultado.txt
18 ./tp1 -i entrada.txt -o salida.txt -I 100 -O 20
19 diff salida.txt resultado.txt
20 ./tp1 -i entrada.txt -o salida.txt -I 1 -O 100
21 diff salida.txt resultado.txt
22 ./tp1 -i entrada.txt -o salida.txt -I 100 -O 1
23 diff salida.txt resultado.txt
24 ./tp1 -i entrada.txt -o salida.txt -I 20 -O 1000
25 diff salida.txt resultado.txt
26 ./tp1 -i entrada.txt -o salida.txt -I 1000 -O 20
27 diff salida.txt resultado.txt
28
29 # Prueba con archivo vacio
30 touch vacio.txt
31 touch resultado_vacio.txt
32 ./tp1 -i vacio.txt -o salida.txt -I 1 -O 1
33 diff salida.txt resultado_vacio.txt
34 ./tp1 -i vacio.txt -o salida.txt -I 20 -O 20
35 diff salida.txt resultado_vacio.txt
36 ./tp1 -i vacio.txt -o salida.txt -I 100 -O 100
37 diff salida.txt resultado_vacio.txt
38 ./tp1 -i vacio.txt -o salida.txt -I 1000 -O 1000
39 diff salida.txt resultado_vacio.txt
40 ./tp1 -i vacio.txt -o salida.txt -I 20 -O 100
41 diff salida.txt resultado_vacio.txt
42 ./tp1 -i vacio.txt -o salida.txt -I 100 -O 20
43 diff salida.txt resultado_vacio.txt
44 ./tp1 -i vacio.txt -o salida.txt -I 1 -O 100
45 diff salida.txt resultado_vacio.txt
46 ./tp1 -i vacio.txt -o salida.txt -I 100 -O 1
47 diff salida.txt resultado_vacio.txt
48 ./tp1 -i vacio.txt -o salida.txt -I 20 -O 1000
49 diff salida.txt resultado_vacio.txt
50 ./tp1 -i vacio.txt -o salida.txt -I 1000 -O 20
51 diff salida.txt resultado_vacio.txt
52
53 # Pruebas con una sola letra mayúscula
54 echo M > res.txt
55 echo M | ./tp1 -o salida.txt -I 1 -O 1
56 diff salida.txt res.txt
57 echo M | ./tp1 -o salida.txt -I 20 -O 20
58 diff salida.txt res.txt
59 echo M | ./tp1 -o salida.txt -I 100 -O 100
60 diff salida.txt res.txt
61 echo M | ./tp1 -o salida.txt -I 1000 -O 1000
62 diff salida.txt res.txt
63 echo M | ./tp1 -o salida.txt -I 20 -O 100
64 diff salida.txt res.txt
65 echo M | ./tp1 -o salida.txt -I 100 -O 20
66 diff salida.txt res.txt
```

```
67  echo M | ./tp1 -o salida.txt -I 1 -O 100
68  diff salida.txt res.txt
69  echo M | ./tp1 -o salida.txt -I 100 -O 1
70  diff salida.txt res.txt
71  echo M | ./tp1 -o salida.txt -I 20 -O 1000
72  diff salida.txt res.txt
73  echo M | ./tp1 -o salida.txt -I 1000 -O 20
74  diff salida.txt res.txt
75
76  # Pruebas con una sola letra minúscula
77  echo m > res.txt
78  echo m | ./tp1 -o salida.txt -I 1 -O 1
79  diff salida.txt res.txt
80  echo m | ./tp1 -o salida.txt -I 20 -O 20
81  diff salida.txt res.txt
82  echo m | ./tp1 -o salida.txt -I 100 -O 100
83  diff salida.txt res.txt
84  echo m | ./tp1 -o salida.txt -I 1000 -O 1000
85  diff salida.txt res.txt
86  echo m | ./tp1 -o salida.txt -I 20 -O 100
87  diff salida.txt res.txt
88  echo m | ./tp1 -o salida.txt -I 100 -O 20
89  diff salida.txt res.txt
90  echo m | ./tp1 -o salida.txt -I 1 -O 100
91  diff salida.txt res.txt
92  echo m | ./tp1 -o salida.txt -I 100 -O 1
93  diff salida.txt res.txt
94  echo m | ./tp1 -o salida.txt -I 20 -O 1000
95  diff salida.txt res.txt
96  echo m | ./tp1 -o salida.txt -I 1000 -O 20
97  diff salida.txt res.txt
98
99  # Prueba con un número
100 echo 3 > res.txt
101 echo 3 | ./tp1 -o salida.txt -I 1 -O 1
102 diff salida.txt res.txt
103 echo 3 | ./tp1 -o salida.txt -I 20 -O 20
104 diff salida.txt res.txt
105 echo 3 | ./tp1 -o salida.txt -I 100 -O 100
106 diff salida.txt res.txt
107 echo 3 | ./tp1 -o salida.txt -I 1000 -O 1000
108 diff salida.txt res.txt
109 echo 3 | ./tp1 -o salida.txt -I 20 -O 100
110 diff salida.txt res.txt
111 echo 3 | ./tp1 -o salida.txt -I 100 -O 20
112 diff salida.txt res.txt
113 echo 3 | ./tp1 -o salida.txt -I 1 -O 100
114 diff salida.txt res.txt
115 echo 3 | ./tp1 -o salida.txt -I 100 -O 1
116 diff salida.txt res.txt
117 echo 3 | ./tp1 -o salida.txt -I 20 -O 1000
118 diff salida.txt res.txt
119 echo 3 | ./tp1 -o salida.txt -I 1000 -O 20
```

```
120 diff salida.txt res.txt
121
122 # Pruebas con un guion
123 echo - > res.txt
124 echo - | ./tp1 -o salida.txt -I 1 -O 1
125 diff salida.txt res.txt
126 echo - | ./tp1 -o salida.txt -I 20 -O 20
127 diff salida.txt res.txt
128 echo - | ./tp1 -o salida.txt -I 100 -O 100
129 diff salida.txt res.txt
130 echo - | ./tp1 -o salida.txt -I 1000 -O 1000
131 diff salida.txt res.txt
132 echo - | ./tp1 -o salida.txt -I 20 -O 100
133 diff salida.txt res.txt
134 echo - | ./tp1 -o salida.txt -I 100 -O 20
135 diff salida.txt res.txt
136 echo - | ./tp1 -o salida.txt -I 1 -O 100
137 diff salida.txt res.txt
138 echo - | ./tp1 -o salida.txt -I 100 -O 1
139 diff salida.txt res.txt
140 echo - | ./tp1 -o salida.txt -I 20 -O 1000
141 diff salida.txt res.txt
142 echo - | ./tp1 -o salida.txt -I 1000 -O 20
143 diff salida.txt res.txt
144
145 # Pruebas con un guion bajo
146 echo _ > res.txt
147 echo _ | ./tp1 -o salida.txt -I 1 -O 1
148 diff salida.txt res.txt
149 echo _ | ./tp1 -o salida.txt -I 20 -O 20
150 diff salida.txt res.txt
151 echo _ | ./tp1 -o salida.txt -I 100 -O 100
152 diff salida.txt res.txt
153 echo _ | ./tp1 -o salida.txt -I 1000 -O 1000
154 diff salida.txt res.txt
155 echo _ | ./tp1 -o salida.txt -I 20 -O 100
156 diff salida.txt res.txt
157 echo _ | ./tp1 -o salida.txt -I 100 -O 20
158 diff salida.txt res.txt
159 echo _ | ./tp1 -o salida.txt -I 1 -O 100
160 diff salida.txt res.txt
161 echo _ | ./tp1 -o salida.txt -I 100 -O 1
162 diff salida.txt res.txt
163 echo _ | ./tp1 -o salida.txt -I 20 -O 1000
164 diff salida.txt res.txt
165 echo _ | ./tp1 -o salida.txt -I 1000 -O 20
166 diff salida.txt res.txt
167
168 # Pruebas con un simbolo
169 echo @ | ./tp1 -o salida.txt -I 1 -O 1
170 diff salida.txt vacio.txt
171 echo @ | ./tp1 -o salida.txt -I 20 -O 20
172 diff salida.txt vacio.txt
```

```
173 echo @ | ./tp1 -o salida.txt -I 100 -O 100
174 diff salida.txt vacio.txt
175 echo @ | ./tp1 -o salida.txt -I 1000 -O 1000
176 diff salida.txt vacio.txt
177 echo @ | ./tp1 -o salida.txt -I 20 -O 100
178 diff salida.txt vacio.txt
179 echo @ | ./tp1 -o salida.txt -I 100 -O 20
180 diff salida.txt vacio.txt
181 echo @ | ./tp1 -o salida.txt -I 1 -O 100
182 diff salida.txt vacio.txt
183 echo @ | ./tp1 -o salida.txt -I 100 -O 1
184 diff salida.txt vacio.txt
185 echo @ | ./tp1 -o salida.txt -I 20 -O 1000
186 diff salida.txt vacio.txt
187 echo @ | ./tp1 -o salida.txt -I 1000 -O 20
188 diff salida.txt vacio.txt
189
190 # Prueba con espacios
191 echo "                    " > ent.txt
192 ./tp1 -i ent.txt -o salida.txt -I 1 -O 1
193 diff salida.txt vacio.txt
194 ./tp1 -i ent.txt -o salida.txt -I 20 -O 20
195 diff salida.txt vacio.txt
196 ./tp1 -i ent.txt -o salida.txt -I 100 -O 100
197 diff salida.txt vacio.txt
198 ./tp1 -i ent.txt -o salida.txt -I 1000 -O 1000
199 diff salida.txt vacio.txt
200 ./tp1 -i ent.txt -o salida.txt -I 20 -O 100
201 diff salida.txt vacio.txt
202 ./tp1 -i ent.txt -o salida.txt -I 100 -O 20
203 diff salida.txt vacio.txt
204 ./tp1 -i ent.txt -o salida.txt -I 1 -O 100
205 diff salida.txt vacio.txt
206 ./tp1 -i ent.txt -o salida.txt -I 100 -O 1
207 diff salida.txt vacio.txt
208 ./tp1 -i ent.txt -o salida.txt -I 20 -O 1000
209 diff salida.txt vacio.txt
210 ./tp1 -i ent.txt -o salida.txt -I 1000 -O 20
211 diff salida.txt vacio.txt
212
213 # Pruebas con simbolos
214 echo "@#$%^*()!{}[],./?<>;:*+\|=+" > ent.txt
215 ./tp1 -i ent.txt -o salida.txt -I 1 -O 1
216 diff salida.txt vacio.txt
217 ./tp1 -i ent.txt -o salida.txt -I 20 -O 20
218 diff salida.txt vacio.txt
219 ./tp1 -i ent.txt -o salida.txt -I 100 -O 100
220 diff salida.txt vacio.txt
221 ./tp1 -i ent.txt -o salida.txt -I 1000 -O 1000
222 diff salida.txt vacio.txt
223 ./tp1 -i ent.txt -o salida.txt -I 20 -O 100
224 diff salida.txt vacio.txt
225 ./tp1 -i ent.txt -o salida.txt -I 100 -O 20
```

```
226 diff salida.txt vacio.txt
227 ./tp1 -i ent.txt -o salida.txt -I 1 -O 100
228 diff salida.txt vacio.txt
229 ./tp1 -i ent.txt -o salida.txt -I 100 -O 1
230 diff salida.txt vacio.txt
231 ./tp1 -i ent.txt -o salida.txt -I 20 -O 1000
232 diff salida.txt vacio.txt
233 ./tp1 -i ent.txt -o salida.txt -I 1000 -O 20
234 diff salida.txt vacio.txt
235
236
237 #Prueba con un archivo con 30 lineas de 5000 caracteres cada una,
238 # donde cada una es palindromo en su totalidad
239
240 ./tp1 -i archivo_largo.txt -o salida.txt -I 1 -O 1
241 diff salida.txt archivo_largo.txt
242 ./tp1 -i archivo_largo.txt -o salida.txt -I 20 -O 20
243 diff salida.txt archivo_largo.txt
244 ./tp1 -i archivo_largo.txt -o salida.txt -I 100 -O 100
245 diff salida.txt archivo_largo.txt
246 ./tp1 -i archivo_largo.txt -o salida.txt -I 1000 -O 1000
247 diff salida.txt archivo_largo.txt
248 ./tp1 -i archivo_largo.txt -o salida.txt -I 20 -O 100
249 diff salida.txt archivo_largo.txt
250 ./tp1 -i archivo_largo.txt -o salida.txt -I 100 -O 20
251 diff salida.txt archivo_largo.txt
252 ./tp1 -i archivo_largo.txt -o salida.txt -I 1 -O 100
253 diff salida.txt archivo_largo.txt
254 ./tp1 -i archivo_largo.txt -o salida.txt -I 100 -O 1
255 diff salida.txt archivo_largo.txt
256 ./tp1 -i archivo_largo.txt -o salida.txt -I 20 -O 1000
257 diff salida.txt archivo_largo.txt
258 ./tp1 -i archivo_largo.txt -o salida.txt -I 1000 -O 20
259 diff salida.txt archivo_largo.txt
260
261 # Prueba error: no se ingresa archivo de entrada
262 echo "Debe␣indicar␣un␣archivo␣de␣entrada␣luego␣de␣-i" > res.txt
263 ./tp1 -i 2> error.txt
264 diff error.txt res.txt
265 ./tp1 -I 10 -i 2> error.txt
266 diff error.txt res.txt
267 ./tp1 -I 10 -O 10 -i 2> error.txt
268 diff error.txt res.txt
269 ./tp1 -o salida.txt -i 2> error.txt
270 diff error.txt res.txt
271
272 # Prueba error: no se ingresa archivo de salida
273 echo "Debe␣indicar␣un␣archivo␣de␣salida␣luego␣de␣-o" > res.txt
274 ./tp1 -o 2> error.txt
275 diff error.txt res.txt
276 ./tp1 -i entrada.txt -o 2> error.txt
277 diff error.txt res.txt
278 ./tp1 -I 10 -o 2> error.txt
```

```
279 diff error.txt res.txt
280 ./tp1 -I 10 -O 10 -o 2> error.txt
281 diff error.txt res.txt
282
283 # Prueba error: no se ingresa tamaño del buffer de entrada
284 echo "Debe indicar un numero luego de -I" > res.txt
285 ./tp1 -I 2> error.txt
286 diff error.txt res.txt
287 ./tp1 -i entrada.txt -I 2> error.txt
288 diff error.txt res.txt
289 ./tp1 -O 10 -I 2> error.txt
290 diff error.txt res.txt
291 ./tp1 -i entrada.txt -O 10 -I 2> error.txt
292 diff error.txt res.txt
293
294 # Prueba error: no se ingresa tamaño del buffer de salida
295 echo "Debe indicar un numero luego de -O" > res.txt
296 ./tp1 -O 2> error.txt
297 diff error.txt res.txt
298 ./tp1 -i entrada.txt -O 2> error.txt
299 diff error.txt res.txt
300 ./tp1 -I 10 -O 2> error.txt
301 diff error.txt res.txt
302 ./tp1 -i entrada.txt -I 10 -O 2> error.txt
303 diff error.txt res.txt
304
305
306 # Prueba error: no se puede abrir el archivo de entrada
307 echo "El archivo de entrada no pudo abrirse" > res.txt
308 ./tp1 -i inexistente.txt 2> error.txt
309 diff error.txt res.txt
310 ./tp1 -o salida.txt -i inexistente.txt 2> error.txt
311 diff error.txt res.txt
312 ./tp1 -I 10 -i inexistente.txt 2> error.txt
313 diff error.txt res.txt
314 ./tp1 -i inexistente.txt -I 10 2> error.txt
315 diff error.txt res.txt
316
317 # Prueba error: el tamaño del buffer de entrada no es un numero
318 echo "El parametro de -I debe ser un numero" > res.txt
319 ./tp1 -I abc 2> error.txt
320 diff error.txt res.txt
321 ./tp1 -i entrada.txt -I numero 2> error.txt
322 diff error.txt res.txt
323 ./tp1 -O 10 -I nueve 2> error.txt
324 diff error.txt res.txt
325 ./tp1 -i entrada.txt -O 10 -I abc123 2> error.txt
326 diff error.txt res.txt
327
328 # Prueba error: el tamaño del buffer de salida no es un numero
329 echo "El parametro de -O debe ser un numero" > res.txt
330 ./tp1 -O abc 2> error.txt
331 diff error.txt res.txt
```

```
./tp1 -i entrada.txt -O numero 2> error.txt
diff error.txt res.txt
./tp1 -I 10 -O nueve 2> error.txt
diff error.txt res.txt
./tp1 -i entrada.txt -I 10 -O abc123 2> error.txt
diff error.txt res.txt

#Pruebas con stdin (sin poner '-i' o poniendo '-i -')
./tp1 -o salida.txt -I 1 -O 1  < entrada.txt
diff salida.txt resultado.txt
./tp1 -o salida.txt -I 20 -O 20  < entrada.txt
diff salida.txt resultado.txt
./tp1 -o salida.txt -I 100 -O 100  < entrada.txt
diff salida.txt resultado.txt
./tp1 -o salida.txt -I 1000 -O 1000  < entrada.txt
diff salida.txt resultado.txt
./tp1 -o salida.txt -I 20 -O 100  < entrada.txt
diff salida.txt resultado.txt
./tp1 -o salida.txt -I 100 -O 20  < entrada.txt
diff salida.txt resultado.txt
./tp1 -i - -o salida.txt -I 1 -O 100  < entrada.txt
diff salida.txt resultado.txt
./tp1 -i - -o salida.txt -I 100 -O 1  < entrada.txt
diff salida.txt resultado.txt
./tp1 -i - -o salida.txt -I 20 -O 1000  < entrada.txt
diff salida.txt resultado.txt
./tp1 -i - -o salida.txt -I 1000 -O 20  < entrada.txt
diff salida.txt resultado.txt


#Prueba con stdout (sin poner '-o' o poniendo '-o -')
./tp1 -i entrada.txt -I 1 -O 1 > salida.txt
diff salida.txt resultado.txt
./tp1 -i entrada.txt -I 20 -O 20 > salida.txt
diff salida.txt resultado.txt
./tp1 -i entrada.txt -I 100 -O 100 > salida.txt
diff salida.txt resultado.txt
./tp1 -i entrada.txt -I 1000 -O 1000 > salida.txt
diff salida.txt resultado.txt
./tp1 -i entrada.txt -I 20 -O 100 > salida.txt
diff salida.txt resultado.txt
./tp1 -i entrada.txt -I 100 -O 20 > salida.txt
diff salida.txt resultado.txt
./tp1 -i entrada.txt -I 1 -O 100 > salida.txt
diff salida.txt resultado.txt
./tp1 -i entrada.txt -I 100 -O 1 > salida.txt
diff salida.txt resultado.txt
./tp1 -i entrada.txt -I 20 -O 1000 > salida.txt
diff salida.txt resultado.txt
./tp1 -i entrada.txt -I 1000 -O 20 > salida.txt
diff salida.txt resultado.txt


```

```
385 #Borramos archivos sobrantes
386 rm vacio.txt
387 rm resultado_vacio.txt
388 rm salida.txt
389 rm ent.txt
390 rm error.txt
391 rm res.txt
```

3.2. Archivo 'entrada.txt'

```
 1 Pruebas varias:
 2 aaa      pelota hola como estas
 3
 4 pepep aaaaaaaaaaaaaaa aaaaaaaaaaaaaaaa
 5 _aa_
 6
 7 _aAAa_
 8 -a-a-
 9 -a-a
10 Neuquen
11 -Neuquen-    neu%q%uen
12
13 1234321    ?123?123abc4cba321
14
15 Prueba del enunciado:
16
17 Somos los primeros en completar el TP 0.
18 Ojo que La fecha de entrega del TP0 es el martes 12 de septiembre.
19
20 Palabras largas mezcladas:
21
22 abcdefghijklmnopqrstuvwxyz0123456789_--
      _9876543210zyxwvutsrqponmlkjihgfedcba??==
23 ABCDEFGHIJKLMnopqrstuvwxyz0123456789_--
      _9876543210zyxwvutsrqponmlkjihgfedcba??==
24 EstoesUnPalindromoOMOrdnilapNUSEOTse.........EStono
25
26 Pruebas de guiones  guiones bajos:
27
28 __--__???????######$$$$_-_-_@@@@@-_-__-_-!    ---___
29
30 Pruebas de palabras de una letra:
31
32 a    %%%%1 2 ^4^ - _ C D
33 b  ! @ # $ %^ & * ( ) = + \
34 c
35 d
36
37 Pruebas solo mayusculas:
38
39 AAA ABCDEDCBA    ABC123--321CBA  WXXW
40
41 PALINDROMO  -ABCB-
```

3.3. Archivo 'resultado.txt'

```
1  aaa
2  pepep
3  aaaaaaaaaaaaaaa
4  aaaaaaaaaaaaaaaa
5  _aa_
6  _aAAa_
7  −a−a−
8  Neuquen
9  −Neuquen−
10 q
11 1234321
12 123abc4cba321
13 Somos
14 0
15 Ojo
16 abcdefghijklmnopqrstuvwxyz0123456789_−−−
       _9876543210zyxwvutsrqponmlkjihgfedcba
17 ABCDEFGHIJKLMnopqrstuvwxyz0123456789_−−−
       _9876543210zyxwvutsrqponmlkjihgfedcba
18 EstoesUnPalindromoOMOrdnilapNUSEOTse
19 __−−__
20 _−_−_
21 −_−__−_−
22 a
23 1
24 2
25 4
26 −
27 _
28 C
29 D
30 b
31 c
32 d
33 AAA
34 ABCDEDCBA
35 ABC123−−321CBA
36 WXXW
```

3.4. Archivo archivo_largo.txt

La función de este archivo es trabajar con varios palíndromos largos para así poder probar el programa correctamente. A continuación se muestra la primer línea de este archivo. Las lineas faltantes son una copia de ésta.

```
ElavidolocopaocopleropodiaHablaorecitayarmamasacadadiaSeamorysiesamor
brevesaletodotanensubonitaparteparalamiradadelosotrosyosadulaseramada
solamasleneEvaySaradanelbondadosodonaseresroponesyodiososUlisesilusos
eranagriosAnaSaritaLaraconavidalocasoledadEvaElsalaotraPetrasonamigas
LafacilocubanaSaritasevaapartaaraleaconotroBasilioSeissonamasamareson
inadamamalarotasosamonaTigresyososresidenopacosunosalacazarotadelamad
```

onayotrosbalanalegresLasamadasnacenenlasaridasuralesrocasUnaeslarusaM
irapordepararyanadirapocolaligaoyalacalaDeseonaZolaterajeRosaesolalam
inalamateMoloanaziuranioRehusorimelenodiosoojoLamejorrazaparaamarosal
ejatobasosamasinevasivasAmorbabososiesnoserahoyamorAledenicodasanapar
aplatonicotemaAsurarteplacidonadamodeladoLamoritaLaraseabocadadesimas
eseaveacalopalioylaseparosanomisyatiVeedadIvanasusamaritanaelevalosse
nosAnaropasacaManolocedeLedicededucalesamoresadoropodernenaserimanago
reroTorotasadoyahoracorneadoeselerigidocomoelajadoNoasonodonusoreyEra
honortapaduraosadaOdotedarodaleverYasepisanRutconnatadaredopadaDeloso
loSalnenemetraesonAmaralysolracionadoPetranoconocesolyoirfoliaconIgor
opinanegadoranegadamareartesoloyabusoEsedalocacocainaMortodasoledadEs
everynoseresodioOjocaefataloroModadopaOilimeNococinaTasotiradobusnore
idrusodaraCansenAliVagonorailalsonUnabalacalabasonAgroserometiosolaMo
jonEsamasleoyacosodeimitadasaperturasonomenoseroticasIramalevolaTamar
alamalaalososadosatacoperonoElsamaridososogujocoponysosotitepasonivid
aNosasolalasodasAseramalacosoRasolyzorraosorbasepaganargarboselamoTom
amadreimanagasOiledadalenotraceproyamorolodorarropaNacaradotioconsies
onoesaellasyavalenorocaladasEpopeyarojabarataOirleAnaesoleajedelocaso
catalanasaneherederaHoynomajaMirelavadotadatodavaradalaligaabajoManol
oidemyatalamotiraronelpavoracaloradoRomanavinomilanesaoiResenabemolep
idemiasamosatenaSolacolarapielesorasuraraperrasApaticaseesEsoidoNorep
aresOiDomoconsumonimoderadonisacroLodominaradesderaboacaboSonaramosal
aneveramosaicocolumnaSiUgaldedecorominimosAparemujereslaElsamaslealla
SaramiralamarAdemasellasenelorbelecomeranellevellanodelociosovaronete
reoroedoryamargadopaladinAsorgenOsoirasosovelanaresEressolelademanoso
sivaadelatarosodiraotraharinamoleranimaCaminaraseguródesusabaticonatu
ralNenasosanunisonasEllosleslesadornanunoslacaraperodanotrosalugaresocas
ipululanenasediosinunapicemasamigoCarameloyagrioledieronysupomalAsumi
rehoynosolodiahoratodoesedeterioroMaladiosupersonaRimerotasaparabolas
enanasysinueraesaanulaloriconoconocesecorefranagalerasaremararemaraga
lerasatoroyyermosomreyyorotasarelagarameraramerasarelaganarferoceseco
noconocirolalunaaseareunisysananesalobarapasatoremiRanosrepusoidalaMo
roiretedeseodotarohaidolosonyoherimusAlamopusynoreideloirgayolemaraCo
gimasamecipanunisoidesanenalulupisacoseragulasortonadoreparacalsonuna
nrodaselsollEsanosinunasosaneNlarutanocitabasusedorugesaranimaCaminar
elomanirahartoaridosorataledaavisosonamedalelosserEseranalevososarios
OnegrosAnidalapodagramayrodeoroeretenoravosoicoledonalLevelLenaremoce
lebrolenesallesamedAramalarimaraSallaelsamaslEalserejumerapAsominimor
ocededlagUiSanmulocociasomarevenalasomaranoSobacaobaredsedaranimodoLo
rcasinodaredominomusnocomoDiOseraperoNodiosEseesacitapAsarrepararusar
oseleiparalocaloSanetasomasaimedipelomebaneseRioasenalimonivanamoRoda
rolacarovaplenoraritomalataymediolonaMojabaagilaladaravadotadatodaval
eriMajamonyoHarederehenasanalatacosacoledejaeloseanAelriOatarabajoray
epopEsadalacoronelavaysalleaseonoseisnocoitodaracaNaporrarodoloromayo
rpecartoneladadeliOsaganamierdamamoTomalesobragranagapesabrosoarrozyl
osaRosocalamaresAsadosalalosasoNadivinosapetitososynopocojugososodira
maslEonorepocatasodasosolaalamalaramaTalovelamarIsacitoresonemonosaru
trepasadatimiedosocayoelsamasEnojoMalosoitemoresorgAnosabalacalabanUn
oslaliaronogaVilAnesnaCaradosurdieronsubodaritosaTanicocoNemiliOapoda
doMorolatafeacojOoidoseresonyrevesEdadelosadotroManiacocacoladesEosub

ayolosetraeramadagenarodagenaniporogInocailofrioyloseconoconartePodan
oicarlosylaramAnoseartemenenlaSolosoleDadapoderadatannoctuRnasipesaYr
eveladoradetodOadasoarudapatronoharEyerosunodonosaoNodajaleomocodigir
eleseodaenrocarohayodasatoroToreroganamiresanenredoporodaseromaselacu
dedecideLedecolonaMacasaporanAsonessolaveleanatiramasusanavIdadeeVita
ysimonasorapesalyoilapolacaevaesesamisedadacobaesaraLatiromaLodaledom
adanodicalpetrarusAametocinotalparapanasadocinedelAromayoharesonseiso
sobabromAsavisavenisamasosabotajelasoramaarapazarrojemaLojoosoidonele
mirosuheRoinaruizanaoloMetamalanimalaloseasoRejaretaloZanoeseDalacala
yoagilalocoparidanayrarapedropariMasuralseanUsacorselarusadirasalnene
cansadamasaLsergelanalabsortoyanodamaledatorazacalasonusocaponedisers
osoysergiTanomasosatoralamamadaninoseramasamanossieSoilisaBortonocael
araatrapaavesatiraSanabucolicafaLsagimanosartePartoalaslEavEdadelosac
oladivanocaraLatiraSanAsoirganaresosulisesilUsosoidoysenoporseresanod
osodadnoblenadaraSyavEenelsamalosadamaresaludasoysortosoledadarimalar
apetrapatinobusnenatodotelaseverbromaseisyromaeSaidadacasamamrayatice
roAlbahaidoporelPocoapocolodiVale

4. Mediciones del tiempo de ejecución

Como se mencionó anteriormente, el objetivo del uso de los buffers es medir cómo afectan las operaciones con archivos al tiempo de ejecución. Para eso, creamos un script 'time.sh', que mide el tiempo que tarda en ejecutarse el programa con distintos tamaños de buffer, y guarda el resultado en un archivo 'time.txt'. Para esto, se utilizó el comando time que te indica el tiempo real, el tiempo de usuario y el tiempo de sistema que le llevó a la aplicación ejecutarse. Para la comparación de los tiempos de ejecución, tomamos únicamente los tiempos reales. Se realizaron las mismas mediciones con el archivo 'entrada.txt' (3.2) y con el 'archivo_largo.txt' (3.4), para analizar cómo afecta también el tamaño del archivo.

El script de mediciones se puede ejecutar con el comando:

$ bash time.sh

4.1. Archivo "time.sh"

```
1  #/bin/bash
2
3  bash compilar.sh
4
5  #Crea el archivo vacio
6  cat /dev/null > time.txt
7
8
9  echo -e "MEDICIONES CON ARCHIVO DE PRUEBAS ENTRADA.TXT\n" >> time.txt
10
11 echo -e "Resultado con 1 byte de buffer:" >> time.txt
12 (time ./tp1 -i entrada.txt -o salida.txt -I 1 -O 1)2>>time.txt
13
14 echo -e "\nResultado con 20 bytes de buffer:" >> time.txt
```

```
15 (time ./tp1 -i entrada.txt -o salida.txt -I 20 -O 20)2>>time.txt
16
17 echo -e "\nResultado con 50 bytes de buffer:" >> time.txt
18 (time ./tp1 -i entrada.txt -o salida.txt -I 50 -O 50)2>>time.txt
19
20 echo -e "\nResultado con 100 bytes de buffer:" >> time.txt
21 (time ./tp1 -i entrada.txt -o salida.txt -I 100 -O 100)2>>time.txt
22
23 echo -e "\nResultado con 250 bytes de buffer:" >> time.txt
24 (time ./tp1 -i entrada.txt -o salida.txt -I 250 -O 250)2>>time.txt
25
26 echo -e "\nResultado con 500 bytes de buffer:" >> time.txt
27 (time ./tp1 -i entrada.txt -o salida.txt -I 500 -O 500)2>>time.txt
28
29 echo -e "\nResultado con 1000 bytes de buffer:" >> time.txt
30 (time ./tp1 -i entrada.txt -o salida.txt -I 1000 -O 1000)2>>time.txt
31
32
33 echo -e "\n\nMEDICIONES CON ARCHIVO CON 30 LINEAS DE 5000 CARACTERES CADA
      UNA (TODAS SON PALINDROMO)\n" >> time.txt
34
35 echo -e "\nResultado con 1 byte de buffer:" >> time.txt
36 (time ./tp1 -i archivo_largo.txt -o salida.txt -I 1 -O 1)2>>time.txt
37
38 echo -e "\nResultado con 20 bytes de buffer:" >> time.txt
39 (time ./tp1 -i archivo_largo.txt -o salida.txt -I 20 -O 20)2>>time.txt
40
41 echo -e "\nResultado con 50 bytes de buffer:" >> time.txt
42 (time ./tp1 -i archivo_largo.txt -o salida.txt -I 50 -O 50)2>>time.txt
43
44 echo -e "\nResultado con 100 bytes de buffer:" >> time.txt
45 (time ./tp1 -i archivo_largo.txt -o salida.txt -I 100 -O 100)2>>time.txt
46
47 echo -e "\nResultado con 250 bytes de buffer:" >> time.txt
48 (time ./tp1 -i archivo_largo.txt -o salida.txt -I 250 -O 250)2>>time.txt
49
50 echo -e "\nResultado con 500 bytes de buffer:" >> time.txt
51 (time ./tp1 -i archivo_largo.txt -o salida.txt -I 500 -O 500)2>>time.txt
52
53 echo -e "\nResultado con 1000 bytes de buffer:" >> time.txt
54 (time ./tp1 -i archivo_largo.txt -o salida.txt -I 1000 -O 1000)2>>time.txt
55
56
57 rm salida.txt
```

4.2. Archivo ”time.txt”

Este es el resultado obtenido al ejecutar las mediciones:

```
1 MEDICIONES CON ARCHIVO DE PRUEBAS ENTRADA.TXT
2
3 Resultado con 1 byte de buffer:
4
5 real    0m0.066s
6 user    0m0.000s
```

```
 7 sys 0m0.082s
 8
 9 Resultado con 20 bytes de buffer:
10
11 real    0m0.035s
12 user    0m0.022s
13 sys 0m0.017s
14
15 Resultado con 50 bytes de buffer:
16
17 real    0m0.035s
18 user    0m0.023s
19 sys 0m0.028s
20
21 Resultado con 100 bytes de buffer:
22
23 real    0m0.031s
24 user    0m0.000s
25 sys 0m0.043s
26
27 Resultado con 250 bytes de buffer:
28
29 real    0m0.035s
30 user    0m0.004s
31 sys 0m0.031s
32
33 Resultado con 500 bytes de buffer:
34
35 real    0m0.035s
36 user    0m0.028s
37 sys 0m0.023s
38
39 Resultado con 1000 bytes de buffer:
40
41 real    0m0.031s
42 user    0m0.004s
43 sys 0m0.027s
44
45
46 MEDICIONES CON ARCHIVO CON 30 LINEAS DE 5000 CARACTERES CADA UNA (TODAS SON
       PALINDROMO)
47
48
49 Resultado con 1 byte de buffer:
50
51 real    0m10.133s
52 user    0m1.816s
53 sys 0m8.336s
54
55 Resultado con 20 bytes de buffer:
56
57 real    0m1.750s
58 user    0m1.125s
```

```
59 sys 0m0.625s
60
61 Resultado con 50 bytes de buffer:
62
63 real    0m1.500s
64 user    0m1.160s
65 sys 0m0.355s
66
67 Resultado con 100 bytes de buffer:
68
69 real    0m1.402s
70 user    0m1.094s
71 sys 0m0.320s
72
73 Resultado con 250 bytes de buffer:
74
75 real    0m1.383s
76 user    0m1.082s
77 sys 0m0.316s
78
79 Resultado con 500 bytes de buffer:
80
81 real    0m1.336s
82 user    0m1.035s
83 sys 0m0.316s
84
85 Resultado con 1000 bytes de buffer:
86
87 real    0m1.332s
88 user    0m1.019s
89 sys 0m0.317s
```

4.3. Resultados y comparaciones

Para analizar los resultados, realizamos unos gráficos con los datos obtenidos:

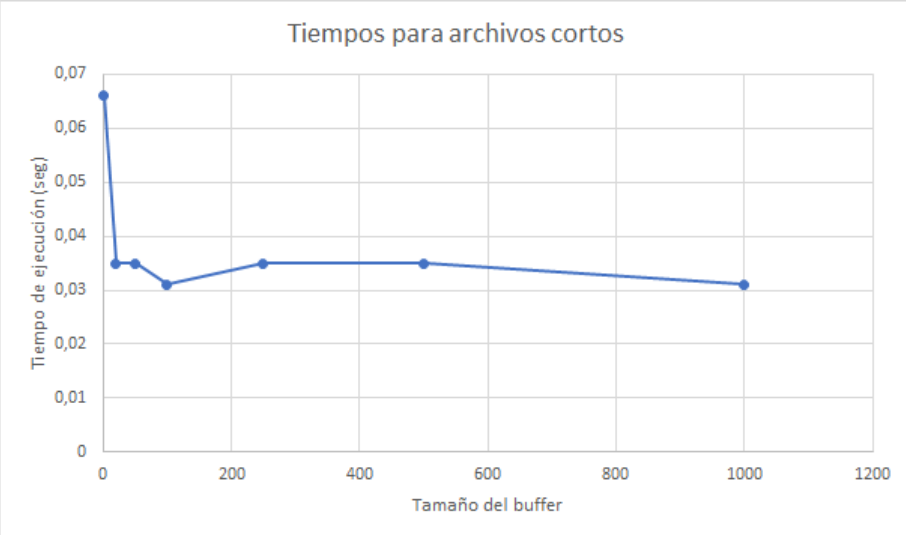

Figura 9: Tiempos de ejecución para archivos cortos

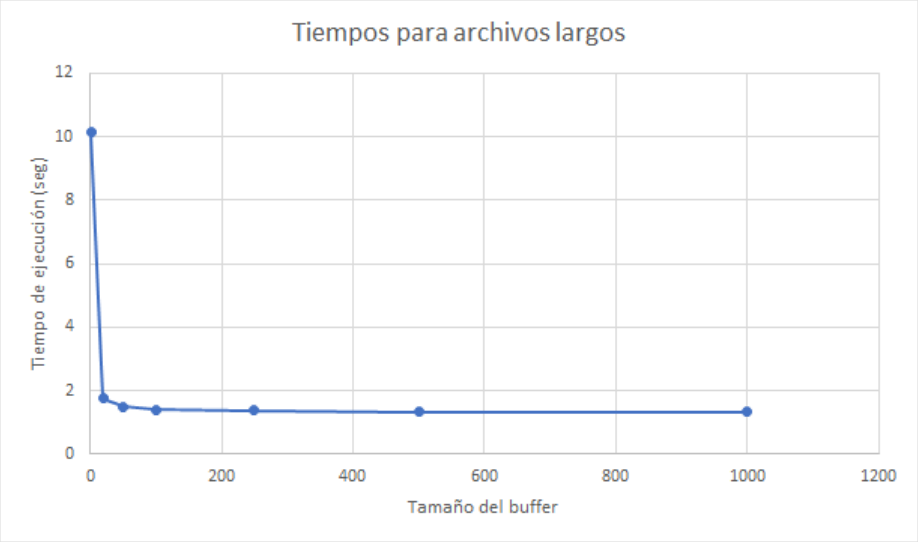

Figura 10: Tiempos de ejecución para archivos largos

Como se puede observar en ambos gráficos, la mayor diferencia se encuentra entre el buffer de tamaño 1 byte y el de 20 bytes, es decir, cuando el buffer es muy chico. Se puede ver que si se sigue aumentando el tamaño el tiempo casi no cambia. Por otro lado, es importante notar que para el archivo largo la diferencia entre el tamaño 1 byte y el de 20 bytes es muy grande, aproximadamente 10 segundos. Esto significa que cuanto más grande es el archivo, mayor es la importancia de que el buffer no sea muy pequeño, aunque tampoco hace falta que sea muy grande. Con los resultados obtenidos, podemos afirmar que el tamaño óptimo del buffer está entre 15 y 100 bytes.

5. Código fuente

5.1. Archivo ”main.c”

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  extern int palindrome (int ifd, size_t ibytes, int ofd, size_t obytes);
6
7  int main(int argc, char* argv[]){
8      FILE* entrada = stdin;
9      FILE* salida = stdout;
10     int tam_buffer_entrada = 1;
11     int tam_buffer_salida = 1;
12     char* parametro;
13
14     int i;
15     for (i = 1; i < argc; i += 2){
16         if ((strcmp(argv[i],"-i") == 0) || (strcmp(argv[i],"--input") == 0)
               ){
17             if (i + 1 >= argc){
18                 fputs("Debe indicar un archivo de entrada luego de -i\n",
                       stderr);
19                 return 3;
```

```
20              }
21              parametro = argv[i + 1];
22              if (strcmp(parametro,"-") != 0){
23                  entrada = fopen(argv[i + 1], "r");
24                  if (!entrada){
25                      fputs("El archivo de entrada no pudo abrirse\n", stderr
                            );
26                      return 4;
27                  }
28              }
29          }
30          else if ((strcmp(argv[i],"-o") == 0) || (strcmp(argv[i],"--output")
                == 0)){
31              if (i + 1 >= argc){
32                  fputs("Debe indicar un archivo de salida luego de -o\n",
                        stderr);
33                  return 3;
34              }
35              parametro = argv[i + 1];
36              if (strcmp(parametro,"-") != 0){
37                  salida = fopen(argv[i + 1], "w");
38                  if (!salida){
39                      fputs("El archivo de salida no pudo abrirse\n", stderr)
                            ;
40                      return 4;
41                  }
42              }
43          }
44          else if ((strcmp(argv[i],"-I") == 0) || (strcmp(argv[i],"--ibuff-
                bytes") == 0)){
45              if (i + 1 >= argc){
46                  fputs("Debe indicar un numero luego de -I\n", stderr);
47                  return 3;
48              }
49              parametro = argv[i + 1];
50              if (strcmp(parametro,"-") != 0){
51                  tam_buffer_entrada = atoi(parametro);
52                  if (tam_buffer_entrada == 0){
53                      fputs("El parametro de -I debe ser un numero\n", stderr
                            );
54                      return 4;
55                  }
56              }
57          }
58          else if ((strcmp(argv[i],"-O") == 0) || (strcmp(argv[i],"--obuff-
                bytes") == 0)){
59              if (i + 1 >= argc){
60                  fputs("Debe indicar un numero luego de -O\n", stderr);
61                  return 3;
62              }
63              parametro = argv[i + 1];
64              if (strcmp(parametro,"-") != 0){
65                  tam_buffer_salida = atoi(parametro);
```

```
66                  if (tam_buffer_salida == 0){
67                      fputs("El␣parametro␣de␣-O␣debe␣ser␣un␣numero\n", stderr
                            );
68                      return 4;
69                  }
70              }
71          }
72          else if ((strcmp(argv[i],"-V") == 0) || (strcmp(argv[i],"--version"
                ) == 0)){
73              fprintf(stdout, "TP1␣version␣1.0001\n");
74              return 0;
75          }
76          else if ((strcmp(argv[i],"-h") == 0) || (strcmp(argv[i],"--help")
                == 0)){
77              fprintf(stdout, "Usage:\n\ntp1␣-h\ntp1␣-V\ntp1␣[options]\n\n");
78              fprintf(stdout, "Options:\n-V,␣--version␣␣Print␣version␣and␣
                    quit.\n");
79              fprintf(stdout, "-h,␣--help␣␣␣Print␣this␣information.\n");
80              fprintf(stdout, "-i,␣--input␣␣␣Location␣of␣the␣input␣file.\n");
81              fprintf(stdout, "-o,␣--output␣␣␣Location␣of␣the␣output␣file.\n"
                    );
82              fprintf(stdout, "-I,␣--ibuf-bytes␣Byte-count␣of␣the␣input␣
                    buffer.\n");
83              fprintf(stdout, "-O,␣--obuf-bytes␣Byte-count␣of␣the␣output␣
                    buffer.\n");
84              fprintf(stdout, "\nExample:\ntp1␣-i␣~/input␣-o␣~/output␣-I␣10␣-
                    O␣10\n\n");
85              return 0;
86          }
87          else {
88              fputs("La␣opcion␣seleccionada␣no␣existe,␣ejecute␣la␣opcion␣-h␣
                    para␣mas␣informacion.\n", stderr);
89              return 3;
90          }
91      }
92
93      int resultado = palindrome(fileno(entrada), tam_buffer_entrada, fileno(
            salida), tam_buffer_salida);
94
95      if (resultado != 0){
96          fputs("Ocurrio␣un␣error\n", stderr);
97      }
98
99      fclose(entrada);
100     fclose(salida);
101
102     return resultado;
103 }
```

5.2. Archivo ”palindrome.S”

```
1 #include <mips/regdef.h>
2 #define TAM_SF 48
3 #define FP_SF 32
```

```
 4 #define GP_SF 36
 5 #define RA_SF 40
 6 #define ARG0 48
 7 #define ARG1 52
 8 #define ARG2 56
 9 #define ARG3 60
10 #define BUFF_IN 24
11 #define BUFF_OUT 28
12 #define LEN 16
13 #define POINTER 20
14
15 .text
16 .abicalls
17 .align 2
18
19 .globl palindrome
20 .ent palindrome
21
22 palindrome:
23     .frame $fp, TAM_SF, ra
24     .set noreorder
25     .cpload t9
26     .set reorder
27
28     subu sp, sp, TAM_SF
29     sw $fp, FP_SF(sp)
30     .cprestore GP_SF
31     sw ra, RA_SF(sp)
32     move $fp, sp
33
34     sw a0, ARG0($fp)                #guardamos el archivo de entrada
35     sw a1, ARG1($fp)                #guardamos el tamanio buffer entrada
36     sw a2, ARG2($fp)                #guardamos el archivo de salida
37     sw a3, ARG3($fp)                #guardamos el tamanio buffer salida
38
39     move a0, a1                     #a0 = tamanio del buffer entrada
40
41     la t9, crear_buffer
42     jalr t9                             #Crea el buffer de entrada
43     beq v0, zero, error_primer_buffer   #Si devuelve 0 ocurrio un error
44     sw v0, BUFF_IN($fp)                 #Guardamos el buffer de entrada
45     la t0, tam_buffer_entrada
46     lw t1, ARG1($fp)
47     sw t1, 0(t0)                    #Actualiza el tam del buffer entrada
48
49     lw a0, ARG3($fp)                    #a0 = tam del buffer de salida
50     la t9, crear_buffer
51     jalr t9                             #Crea el buffer de salida
52     beq v0, zero, error_segundo_buffer  #Si devuelve 0 ocurrio un error
53     sw v0, BUFF_OUT($fp)                #Guardamos el buffer de salida
54     la t0, tam_buffer_salida
55     lw t1, ARG3($fp)
56     sw t1, 0(t0)                     #Actualiza el tam del buffer salida
```

```
57
58 loop:
59     lw a0, ARG0($fp)                    #a0 = archivo de entrada
60     addu a1, $fp, LEN                   #a1 = puntero a len
61     lw a2, BUFF_IN($fp)                 #a2 = buffer de entrada
62
63     la t9, leer_palabra
64     jalr t9                             #Lee la proxima palabra
65
66     sw v0, POINTER($fp)                #Guardamos el puntero a la palabra
67     beq v0, zero, error               #Salta si no se pudo leer la palabra
68
69     move a0, v0                         #a0 = puntero a palabra
70     lw a1, LEN ($fp)                    #a1 = len(palabra)
71     la t9, es_capicua
72     jalr t9                             #Llama a es_capicua
73
74     beq v0, zero, continuar_loop        #Si no es capicua sigue el loop
75
76     lw a0, ARG2($fp)                    #a0 = archivo de salida
77     lw a1, POINTER($fp)                 #a1 = puntero a la palabra
78     lw a2, BUFF_OUT($fp)                #a2 = buffer de salida
79     la t9, putch
80     jalr t9                             #Escribe la palabra en el archivo
81     beq v0, zero, error_escritura       #Si devuelve 0 ocurrio un error
82
83 continuar_loop:
84
85     lw a0, POINTER($fp)                 #a0 = puntero a palabra
86     la t9, myfree
87     jalr t9                             #Llama a free(palabra)
88
89     la t0, escribir_eof
90     lw t1, 0(t0)                        #t1 = hay que escribir eof?
91     beq t1, zero, loop                  #salta si no hay que escribirlo
92
93     lw a0, ARG2($fp)                    #a0 = archivo de salida
94     lw a2, BUFF_OUT($fp)                #a2 = buffer de salida
95     la t9, putch
96     jalr t9                        #Escribe todo lo que queda en el buffer
97     beq v0, zero, error                 #Si devuelve 0 ocurrio un error
98
99 terminar:
100    move v0, zero
101 free_segundo_buffer:
102    lw a0, BUFF_OUT($fp)                 #Cargamos el buffer de salida
103    la t9,myfree
104    jalr t9                              #Liberamos el segundo buffer
105 free_primer_buffer:
106    lw a0, BUFF_IN($fp)                  #Cargamos el buffer de entrada
107    la t9,myfree
108    jalr t9                              #Liberamos el primer buffer
109
```

```
110 return:
111     lw ra, RA_SF($fp)
112     lw gp, GP_SF($fp)
113     lw $fp, FP_SF($fp)
114     addiu sp, sp, TAM_SF
115     jr ra
116
117 error_segundo_buffer:
118     addiu v0, zero, 1                   #Devuelve codigo de error 1
119     b free_primer_buffer                #Libera el primer buffer
120 error_primer_buffer:
121     addiu v0, zero, 1                   #Devuelve codigo de error 1
122     b return                            #No libera nada
123
124 error_escritura:
125     lw a0, POINTER($fp)                 #a0 = puntero a la palabra
126     la t9, myfree
127     jalr t9                             #Libera la palabra
128 error:
129     addiu v0, zero, 2                   #Devuelve codigo de error 2
130     b free_segundo_buffer               #Libera ambos buffer
131
132 .end palindrome
133 .size palindrome,.-palindrome
134
135
136
137 .data
138
139 .globl pos_buffer_entrada
140 .globl tam_buffer_entrada
141 .globl pos_buffer_salida
142 .globl tam_buffer_salida
143 .globl eof_leido
144 .globl escribir_eof
145 .globl TAM
146
147
148 pos_buffer_entrada: .word -1
149 tam_buffer_entrada: .word 0
150 pos_buffer_salida: .word 0
151 tam_buffer_salida: .word 0
152 eof_leido: .word 0
153 escribir_eof: .word 0
154 TAM: .word 30
```

5.3. Archivo ”getch.S”

```
1 #include <mips/regdef.h>
2 #include <sys/syscall.h>
3 #define TAM_SF 40
4 #define S0_SF 16
5 #define S1_SF 20
6 #define FP_SF 24
```

```
 7 #define GP_SF 28
 8 #define RA_SF 32
 9 #define ARG0 40
10 #define ARG1 44
11
12 .text
13 .abicalls
14 .align 2
15
16 .globl getch
17 .ent getch
18
19 getch:
20
21     .frame $fp, TAM_SF, ra
22     .set noreorder
23     .cpload t9
24     .set reorder
25
26     subu sp, sp, TAM_SF
27     sw s0, S0_SF(sp)
28     sw s1, S1_SF(sp)
29     sw $fp, FP_SF(sp)
30     .cprestore GP_SF
31     sw ra, RA_SF(sp)
32     move $fp, sp
33
34     sw a0, ARG0($fp)                  #Guardamos el archivo de entrada
35     sw a1, ARG1($fp)                  #Guardamos el puntero al buffer
36
37     la t0, pos_buffer_entrada
38     lw s0, 0(t0)                      #s0 = pos actual del buffer
39
40     la t0, tam_buffer_entrada
41     lw a2, 0(t0)                      #a2 = tam actual del buffer
42
43     bltu s0, a2, leer_caracter        #Salta si pos actual < tam buffer
44
45 #Si no salta hay que volver a leer del archivo y llenar el buffer
46
47     la t0, eof_leido
48     lw t1, 0(t0)                      #t1 = eof fue leido?
49     bne t1, zero, leyo_eof            #salta si fue leido
50
51     move s1,zero                      #s1 = cantidad bytes leidos = 0
52     move s0, zero                     #Pos actual = 0
53
54 #Lectura del archivo
55
56 leer_archivo:
57     lw a0, ARG0($fp)                  # a0 = archivo de entrada
58     lw a1, ARG1($fp)
59     addu a1, a1, s1              #a1=puntero buffer + cant bytes leidos
```

```
60      la t0, tam_buffer_entrada
61      lw a2, 0(t0)
62      subu a2, a2, s1                 #a2=tam buffer - cant bytes leidos
63      li v0, SYS_read
64      syscall
65      bne a3, zero, error               #Salta si hubo un error
66      beq v0, zero, eof_fue_leido       #Si es cero leyo eof
67
68      addu s1, s1, v0                   #s1 = cant bytes leidos
69      la t0, tam_buffer_entrada
70      lw t1, 0(t0)                      #t1 = tam buffer entrada
71      blt s1, t1, leer_archivo       #Si bytes leidos < tam vuelve a leer
72      b leer_caracter                   #Si ya leyo todo salta
73
74  eof_fue_leido:
75      la t0, tam_buffer_entrada       #Si leyo menos bytes pero todavia
76      sw s1, 0(t0)                    #hay cosas en el buffer
77                                      #Actualizo el tamanio del buffer
78      la t0, eof_leido
79      addiu t1, zero, 1
80      sw t1, 0(t0)                    #Actualiza eof leido
81      beq s1, zero, leyo_eof          #Salta si no leyo nada
82
83
84  leer_caracter:
85
86      lw a1, ARG1($fp)                  #Cargo el puntero al buffer
87      addu t2, s0, a1                   #t2 = buffer + pos
88      lbu v0, 0(t2)                     #v0 = caracter leido
89      addiu s0, s0, 1                   #Incremento la posicion actual
90
91      la t0, pos_buffer_entrada
92      sw s0, 0(t0)                   #Guardamos la pos actual del buffer
93
94  return:
95      lw s0, S0_SF($fp)
96      lw s1, S1_SF($fp)
97      lw ra, RA_SF($fp)
98      lw gp, GP_SF($fp)
99      lw $fp, FP_SF($fp)
100     addiu sp, sp, TAM_SF
101     jr ra
102
103 leyo_eof:
104     la t0, escribir_eof
105     addiu t1,zero,1
106     sw t1, 0(t0)                      #Actualizo escribir_eof
107     addu v0, zero, zero               #Devuelvo 0
108     b return
109
110 error:
111     addiu v0, zero, -1                #Devuelvo -1
112     b return
```

```
113
114 .end getch
115 .size getch,.-getch
```

5.4. Archivo "putch.S"

```
  1 #include <mips/regdef.h>
  2 #include <sys/syscall.h>
  3 #define TAM_SF 48
  4 #define S0_SF 16
  5 #define S1_SF 20
  6 #define S2_SF 24
  7 #define S3_SF 28
  8 #define FP_SF 32
  9 #define GP_SF 36
 10 #define RA_SF 40
 11 #define ARG0 48
 12 #define ARG1 52
 13 #define ARG2 56
 14 #define ASCII_NEWLINE 10
 15
 16 .text
 17 .abicalls
 18 .align 2
 19
 20 .globl putch
 21 .ent putch
 22
 23 putch:
 24
 25     .frame $fp, TAM_SF, ra
 26     .set noreorder
 27     .cpload t9
 28     .set reorder
 29
 30     subu sp, sp, TAM_SF
 31     sw s0, S0_SF(sp)
 32     sw s1, S1_SF(sp)
 33     sw s2, S2_SF(sp)
 34     sw s3, S3_SF(sp)
 35     sw $fp, FP_SF(sp)
 36     .cprestore GP_SF
 37     sw ra, RA_SF(sp)
 38     move $fp, sp
 39
 40     sw a0, ARG0($fp)              #Guardamos el archivo de salida
 41     sw a1, ARG1($fp)              #Guardamos el puntero a la palabra
 42     sw a2, ARG2($fp)              #Guardamos el buffer
 43
 44     la t0, pos_buffer_salida
 45     lw s0, 0(t0)                  #s0 = pos actual del buffer
 46
 47     addu s1, zero, zero           #s1 = indice de la palabra = 0
 48
```

```
49      la t0, escribir_eof
50      lw t1, 0(t0)                    #t1 = eof debe ser escrito?
51      beq t1,zero, loop               #Salta si eof no debe ser escrito
52
53  #Si no salta es la ultima escritura en el archivo
54      move s3, zero                   #s3 = cant bytes escritos = 0
55      la t0, tam_buffer_salida
56      sw s0, 0(t0)                    #Actualiza el tamanio del buffer
57                                      #a la posicion actual
58      b escribir_todo
59
60
61  loop:
62      lw a1, ARG1($fp)                #Cargamos el puntero a la palabra
63      addu t3, a1, s1                 #t3 = palabra + indice
64      lbu s2, 0(t3)                   #s2 = caracter a escribir
65
66      lw a2, ARG2($fp)                #Cargamos el puntero al buffer
67      addu t5, a2, s0                 #t5 = buffer + pos_actual
68      sb s2, 0(t5)                    #Guardamos el caracter
69
70      addiu s0,s0,1                   #Incrementamos pos_actual
71      la t0, tam_buffer_salida
72      lw t1, 0(t0)                    #Cargamos el tamanio del buffer
73      bltu s0, t1, continuar_loop     #Salta si pos_actual < tam_buffer
74      move s3, zero                   #s3 = cant bytes escritos = 0
75
76  #Si no salta hay que volver a escribir el archivo y vaciar el buffer
77
78  escribir_todo:
79      lw a0, ARG0($fp)                #Cargamos el archivo de salida
80      lw a1, ARG2($fp)                #Cargamos el buffer
81      addu a1, a1, s3                 #a1 = buffer + cant bytes escritos
82      la t0, tam_buffer_salida
83      lw a2, 0(t0)                    #Cargamos tam total del buffer
84      subu a2, a2, s3                 #a2 = tam - cant bytes escritos
85      li v0, SYS_write
86      syscall
87
88      bne a3, zero, error             #Salta si ocurrio un error
89      addu s3, s3, v0                 #s3 = cant bytes escritos
90      la t0, tam_buffer_salida
91      lw t1, 0(t0)                    # t1 = tamanio total del buffer
92      blt s3, t1, escribir_todo      #Si escribio menos vuelve a escribir
93
94      addu s0, zero, zero             #Pos_actual = 0
95
96      la t0, escribir_eof
97      lw t1, 0(t0)                    #t1 = eof escrito ?
98      bne t1, zero, terminar          #Termina si EOF fue escrito
99
100 continuar_loop:
101     addu s1, s1, 1                  #Incrementa el indice de la palabra
```

```
102     bne s2, ASCII_NEWLINE, loop      #Vuelve al loop si no es \n
103
104 terminar:
105     addiu v0, zero, 1                 #Devuelve 1 si no ocurrio un error
106
107 return:
108     la t0, pos_buffer_salida
109     sw s0, 0(t0)                      #Actualiza la posicion actual
110     lw s0, S0_SF($fp)
111     lw s1, S1_SF($fp)
112     lw s2, S2_SF($fp)
113     lw s3, S3_SF($fp)
114     lw ra, RA_SF($fp)
115     lw gp, GP_SF($fp)
116     lw $fp, FP_SF($fp)
117     addiu sp, sp, TAM_SF
118     jr ra
119
120 error:
121     move v0, zero                       #Devuelve 0 si ocurrio un error
122     b return
123
124 .end putch
125 .size putch,.-putch
```

5.5. Archivo ”crear_buffer.S”

```
 1 #include <mips/regdef.h>
 2 #define TAM_SF 32
 3 #define FP_SF 16
 4 #define GP_SF 20
 5 #define RA_SF 24
 6 #define ARG0 32
 7
 8 .text
 9 .abicalls
10 .align 2
11
12 .globl crear_buffer
13 .ent crear_buffer
14
15 crear_buffer:
16     .frame $fp, TAM_SF, ra
17     .set noreorder
18     .cpload t9
19     .set reorder
20
21     subu sp, sp, TAM_SF
22     sw $fp, FP_SF(sp)
23     .cprestore GP_SF
24     sw ra, RA_SF(sp)
25     move $fp, sp
26
27     sw a0, ARG0($fp)                 #Guardamos el tamanio del buffer
```

```
28
29      la t9, mymalloc
30      jalr t9                         #Llama a malloc
31
32  return:
33      lw ra, RA_SF($fp)
34      lw gp, GP_SF($fp)
35      lw $fp, FP_SF($fp)
36      addiu sp, sp, TAM_SF
37      jr ra
38
39  .end crear_buffer
40  .size crear_buffer,.-crear_buffer
```

5.6. Archivo "leer_palabra.S"

```
 1  #include <mips/regdef.h>
 2  #define TAM_SF 40
 3  #define S0_SF 24
 4  #define FP_SF 28
 5  #define GP_SF 32
 6  #define RA_SF 36
 7  #define ARG0 40
 8  #define ARG1 44
 9  #define ARG2 48
10  #define POINTER 16
11  #define ASCII_A_MAY 65
12  #define ASCII_Z_MAY 90
13  #define ASCII_A_MIN 97
14  #define ASCII_Z_MIN 122
15  #define ASCII_CERO 48
16  #define ASCII_NUEVE 57
17  #define ASCII_GUION 45
18  #define ASCII_GUIONBAJO 95
19  #define ASCII_NEWLINE 10
20
21  .text
22  .abicalls
23  .align 2
24
25  .globl leer_palabra
26  .ent leer_palabra
27
28  leer_palabra:
29      .frame $fp, TAM_SF, ra
30      .set noreorder
31      .cpload t9
32      .set reorder
33
34      subu sp, sp, TAM_SF
35      sw s0, S0_SF(sp)
36      sw $fp, FP_SF(sp)
37      .cprestore GP_SF
38      sw ra, RA_SF(sp)
```

```
39      move $fp, sp
40
41      sw a0, ARG0($fp)                    #Guardamos el puntero al archivo
42      sw a1, ARG1($fp)                    #Guardamos el puntero a longitud
43      sw a2, ARG2($fp)                    #Guardamos el puntero al buffer
44
45      la t0, TAM
46      lw a0, 0(t0)                        #Carga TAM en a0
47      la t9, mymalloc
48      jalr t9                             #Llama a malloc
49      beq v0, zero, terminar_con_error    #Si devuelve 0 ocurrio un error
50
51      sw v0, POINTER($fp)                 #Guardamos el puntero a la palabra
52      addu s0, zero, zero                 #Inicializamos len(palabra) en 0
53
54
55  loop:
56      lw a0, ARG0($fp)                    #Recuperamos el puntero al archivo
57      lw a1, ARG2($fp)                    #Recuperamos el puntero al buffer
58      la t9, getch
59      jalr t9                             #Leemos un caracter, queda en v0
60      beq v0, -1, error                   #Si devuelve -1 ocurrio un error
61
62
63      beq v0, ASCII_GUION, es_caracter        #Salta si es -
64
65      beq v0, ASCII_GUIONBAJO, es_caracter    #Salta si es _
66
67      sgeu t0, v0, ASCII_A_MAY                #Mayor que "A"
68      sleu t1, v0, ASCII_Z_MAY                #Menor que "Z"
69      beq t0, t1, es_caracter                 #Salta si es letra mayuscula
70
71      sgeu t0, v0, ASCII_A_MIN                #Mayor que "a"
72      sleu t1, v0, ASCII_Z_MIN                #Menor que "z"
73      beq t0, t1, es_caracter                 #Salta si es letra minuscula
74
75      sgeu t0, v0, ASCII_CERO                 #Mayor que "0"
76      sleu t1, v0, ASCII_NUEVE                #Menor que "9"
77      beq t0, t1, es_caracter                 #Salta si es un numero
78
79  no_es_caracter:
80      lw a0, POINTER($fp)                 #Recuperamos puntero a palabra
81      addu t0, s0, a0                     #t0 = palabra + len
82      addiu t1, zero, ASCII_NEWLINE
83      sb t1, 0(t0)                        #Guardamos el "\n"
84
85      lw a1, ARG1($fp)                    #Cargamos puntero a len
86      sw s0, 0(a1)                        #Guardamos el len
87      b terminar                          #Sale del loop
88
89  es_caracter:
90      lw a0, POINTER($fp)                 #Recuperamos puntero a palabra
91      addu t0, s0, a0                     #t0 = palabra + len
```

```
92      sb v0, 0(t0)                    #Guardamos el caracter
93
94      addiu s0, s0, 1                 #Incrementamos len en 1
95
96      la t0, TAM
97      lw t1, 0(t0)                    #Carga TAM en t1
98      remu t2, s0, t1                 #t2 = len %tam
99
100     bne t2, zero, loop              #Salta si el modulo no es 0
101
102     lw a0, POINTER($fp)             #a0 =  puntero a palabra
103     move a1, s0                     #a1 = len
104     la t0, TAM
105     lw a2, 0(t0)                    #a2 = TAM
106
107     la t9, myrealloc
108     jalr t9                         #Llama a realloc
109     beq v0, zero, error             #Si devuelve 0 ocurrio un error
110     sw v0, POINTER($fp)             #Guardamos el nuevo puntero
111
112     b loop                          #Vuelve siempre al loop
113
114 terminar:
115     lw v0, POINTER($fp)             #v0 = puntero a la palabra
116
117 return:
118     lw s0, S0_SF($fp)
119     lw ra, RA_SF($fp)
120     lw gp, GP_SF($fp)
121     lw $fp, FP_SF($fp)
122     addiu sp, sp, TAM_SF
123     jr ra
124
125 error:
126     lw a0, POINTER($fp)             # Recuperamos puntero a palabra
127     la t9, myfree
128     jalr t9                         #Llama a free con la palabra
129 terminar_con_error:
130     move v0, zero                   #Devuelve 0 si ocurrio un error
131     b return
132
133 .end leer_palabra
134 .size leer_palabra,.-leer_palabra
```

5.7. Archivo ”es_capicua.S”

```
1 #include <mips/regdef.h>
2 #define TAM_SF 48
3 #define S0_SF 24
4 #define S1_SF 28
5 #define FP_SF 32
6 #define GP_SF 36
7 #define RA_SF 40
8 #define ARG0 48
```

```
 9 #define ARG1 52
10 #define CHAR1 16
11 #define CHAR2 20
12
13 .text
14 .abicalls
15 .align 2
16
17 .globl es_capicua
18 .ent es_capicua
19
20 es_capicua:
21     .frame $fp, TAM_SF, ra
22     .set noreorder
23     .cpload t9
24     .set reorder
25
26     subu sp, sp, TAM_SF
27     sw s0, S0_SF(sp)
28     sw s1, S1_SF(sp)
29     sw $fp, FP_SF(sp)
30     .cprestore GP_SF
31     sw ra, RA_SF(sp)
32     move $fp, sp
33
34     sw a0, ARG0($fp)            # Guardamos el puntero a la palabra
35     sw a1, ARG1($fp)            # Guardamos la longitud
36
37     beq a1, zero, return_false  #Si len es 0 devuelve false
38
39     addu s0, zero, zero         #Inicializamos la variable inicio
40     subu s1, a1, 1              #Inicializamos la variable final
41
42 loop:
43     lw t0, ARG0($fp)            #Recuperamos el puntero a la palabra
44     addu t0, t0, s0             #t0 = palabra + inicio
45     lbu a0, 0(t0)               # Leemos el  caracter palabra[inicio]
46     la t9, my_tolower
47     jalr t9                     #Llamamos a tolower() con el caracter
48     sw v0, CHAR1($fp)           #Guardamos el caracter en minuscula
49     lw a0, ARG0($fp)            #Recuperamos el puntero a la palabra
50     addu t0, a0, s1             #t0 = palabra + final
51     lbu a0, 0(t0)               #Leemos el  caracter palabra[final]
52     la t9, my_tolower
53     jalr t9                     #Llamamos a tolower() con el caracter
54     sw v0, CHAR2($fp)           #Guardamos el caracter en minuscula
55     move t0, v0                 #t0 = segundo caracter
56     lw t1, CHAR1($fp)           #t1 = primer caracter
57     bne t0, t1, return_false    #Si son distintos devuelve false
58
59     addiu s0, s0, 1             #Suma uno (un byte) a inicio
60     subu s1, s1, 1              #Resta uno (un byte) a final
61     blt s0, s1, loop            #Si inicio < final vuelve al loop
```

```
62
63 return_true:
64     addiu v0, zero, 1
65     b return
66
67 return_false:
68     addu v0, zero, zero
69
70 return:
71     lw ra, RA_SF($fp)
72     lw gp, GP_SF($fp)
73     lw s1, S1_SF($fp)
74     lw s0, S0_SF($fp)
75     lw $fp, FP_SF($fp)
76     addiu sp, sp, TAM_SF
77     jr ra
78
79 .end es_capicua
80 .size es_capicua,.-es_capicua
```

5.8. Archivo "my_tolower.S"

```
 1 #include <mips/regdef.h>
 2 #define TAM_SF 24
 3 #define FP_SF 16
 4 #define GP_SF 20
 5 #define ARG0 24
 6 #define ASCII_A_MAY 65
 7 #define ASCII_Z_MAY 90
 8 #define DIF_MAY_MIN 32
 9
10 .text
11 .abicalls
12 .align 2
13
14 .globl my_tolower
15 .ent my_tolower
16
17 my_tolower:
18     .frame $fp, TAM_SF, ra
19     .set noreorder
20     .cpload t9
21     .set reorder
22
23     subu sp, sp, TAM_SF
24     sw $fp, FP_SF(sp)
25     .cprestore GP_SF
26     move $fp, sp
27
28     sw a0, ARG0($fp)                #Guardamos el caracter
29
30     blt a0, ASCII_A_MAY, return     #Salta si caracter es menor a A
31     bgt a0, ASCII_Z_MAY, return     #Salta si caracter es mayor a Z
32     addiu a0, a0, DIF_MAY_MIN       #Convierte a minuscula
```

```
33
34 return:
35     move v0, a0                          #Pone el resultado en v0
36     lw gp, GP_SF($fp)
37     lw $fp, FP_SF($fp)
38     addiu sp, sp, TAM_SF
39     jr ra
40
41 .end my_tolower
42 .size my_tolower,.-my_tolower
```

5.9. Archivo "myrealloc.S"

```
 1 #include <mips/regdef.h>
 2 #define TAM_SF 40
 3 #define S0_SF 24
 4 #define FP_SF 28
 5 #define GP_SF 32
 6 #define RA_SF 36
 7 #define ARG0 40
 8 #define ARG1 44
 9 #define ARG2 48
10 #define NEW_POINTER 16
11
12 .text
13 .abicalls
14 .align 2
15
16 .globl myrealloc
17 .ent myrealloc
18
19 myrealloc:
20     .frame $fp, TAM_SF, ra
21     .set noreorder
22     .cpload t9
23     .set reorder
24
25     subu sp, sp, TAM_SF
26     sw s0, S0_SF(sp)
27     sw $fp, FP_SF(sp)
28     .cprestore GP_SF
29     sw ra, RA_SF(sp)
30     move $fp, sp
31
32     sw a0, ARG0($fp)             # guardamos el puntero
33     sw a1, ARG1($fp)             # guardamos el tamanio del bloque
34     sw a2, ARG2($fp)             # guardamos el tamanio a agregar
35
36     addu a0, a1, a2              #a0 = nuevo tamanio bloque
37     la t9, mymalloc
38     jalr t9                      #Llama a malloc
39     beq v0, zero, return         #Si devuelve cero ocurrio un error
40
41     sw v0, NEW_POINTER($fp)      #Guardamos el puntero nuevo
```

```
42      addu s0, zero, zero          #s0 = indice actual
43
44  loop:
45      lw t0, ARG1($fp)             #t0 = tamanio original
46      bgeu s0, t0, terminar        #salta si ya copio todo
47
48      lw t0, ARG0 ($fp)            #t0 = puntero viejo
49      addu t0, t0, s0              #t0 = puntero viejo + indice
50
51      lw t1, NEW_POINTER($fp)      #t1 = puntero nuevo
52      addu t1, t1, s0              #t1 = puntero nuevo + indice
53
54      lbu t2, 0(t0)                #Cargo el byte a copiar en t2
55      sb t2, 0(t1)                 #Guardo el byte
56
57      addiu s0, s0, 1              #incremento el indice
58      b loop                       #Vuelve al loop
59
60  terminar:
61      lw a0, ARG0($fp)             #a0 = puntero viejo
62      la t9, myfree
63      jalr t9                      #Libera el puntero viejo
64      lw v0, NEW_POINTER($fp)      #v0 = puntero nuevo
65
66  return:
67      lw s0, S0_SF($fp)
68      lw ra, RA_SF($fp)
69      lw gp, GP_SF($fp)
70      lw $fp, FP_SF($fp)
71      addiu sp, sp, TAM_SF
72      jr ra
73
74  .end myrealloc
75  .size myrealloc,.-myrealloc
```

66:20 Organización de Computadoras
Trabajo práctico #1: programación MIPS
2^o cuatrimestre de 2017

`$Date: 2017/09/19 09:29:47 $`

1. Objetivos

Familiarizarse con el conjunto de instrucciones MIPS y el concepto de ABI, extendiendo un programa que resuelva el problema descripto en la sección 4.

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, por escrito, 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 5), la presentación de los resultados obtenidos explicando, cuando corresponda, con fundamentos reales, las causas o razones de cada resultado obtenido.

4. Descripción

En este trabajo, se reimplementará parcialmente en assembly MIPS el programa desarrollado en el trabajo práctico anterior.

Para esto, se requiere reescribir el programa, de forma tal que quede organizado de la siguiente forma:

- Arranque y configuración del programa: procesamiento de las opciones de línea de comandos, apertura y cierre de archivos (de ser necesario), y reporte de errores. Desde aquí se invocará a las función de procesamiento del *stream* de entrada.

- Procesamiento: contendrá el código MIPS32 assembly con la función `palindrome()`, encargada de identificar, procesar e imprimir los componentes léxicos que resulten ser palíndromos, de forma equivalente a lo realizado en el TP anterior.

La función MIPS32 `palindrome()` antes mencionada se corresponderá con el siguiente prototipo en C:

```
int palindrome(int ifd, size_t ibytes, int ofd, size_t obytes);
```

Esta función es invocada por el módulo de arranque y configuración del programa, y recibe en `ifd` y `ofd` los descriptores abiertos de los archivos de entrada y salida respectivamente.

Los parámetros `ibytes` y `obytes` describen los tamaños en bytes de las unidades de transferencia de datos desde y hacia el kernel de NetBSD, y permiten implementar un esquema de *buffering* de estas operaciones de acuerdo al siguiente diagrama:

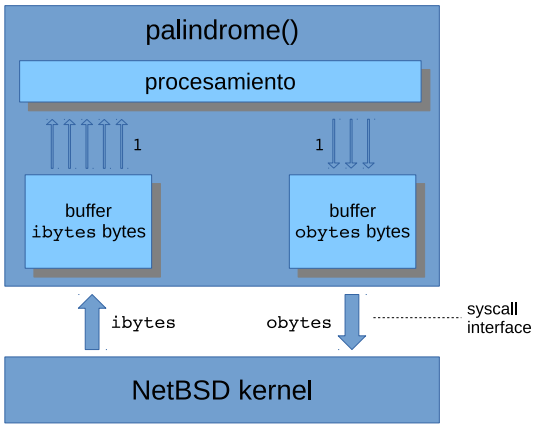

Figura 1: arquitectura de procesamiento.

Como puede verse en la figura 1, la lógica de procesamiento de la función `palindrome()` va leyendo los caracteres del *buffer* de entrada en forma individual.

En el momento en el cual `palindrome()` intente extraer un nuevo caracter, y el *buffer* de entrada se encuentre vacío, deberá ejecutar una llamada al sistema operativo para realizar una lectura en bloque y llenar completamente el buffer, siendo el tamaño de bloque igual a `ibytes` bytes.

De forma análoga, `palindrome()` irá colocando uno a uno los caracteres de las palabras capicúa en el *buffer* de salida. En el momento en el que se agote su capacidad, deberá vaciarlo mediante una operación de escritura hacia el kernel de NetBSD para continuar luego con su procesamiento.

Al finalizar la lectura y procesamiento de los datos de entrada, es probable que exista información esperando a ser enviada al sistema operativo. En ese caso `palindrome()` deberá ejecutar una última llamada al sistema con el fin de vaciar completamente el *buffer* de salida.

Se sugiere encapsular la lógica de *buffering* de entrada/salida con funciones, `getch()` y `putch()`. Asimismo durante la clase del martes 19/9 explicaremos la función `mymalloc()` que deberá ser usada para reservar dinámicamente la memoria de los buffers.

4.1. Ejemplos

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

```
$ tp0 -h
Usage:
  tp0 -h
  tp0 -V
  tp0 [options]
Options:
  -V, --version     Print version and quit.
  -h, --help        Print this information.
  -i, --input       Location of the input file.
  -o, --output      Location of the output file.
  -I, --ibuf-bytes  Byte-count of the input buffer.
  -O, --obuf-bytes  Byte-count of the output buffer.
Examples:
  tp0 -i ~/input -o ~/output
```

Codificamos un archivo vacío (cantidad de bytes nula):

```
$ touch /tmp/zero.txt
$ tp0 -i /tmp/zero.txt -o /tmp/out.txt
$ ls -l /tmp/out.txt
-rw-r--r--  1 user group 0 2017-03-19 15:14 /tmp/out.txt
```

Leemos un *stream* cuyo único contenido es el caracter ASCII `M`,

```
$ echo Hola M | tp0
M
```

Observar que la salida del programa contiene aquellas palabras de la entrada que sean palíndromos (`M` en este caso).

Veamos que sucede al procesar archivo de mayor complejidad:

```
$ cat entrada.txt
Somos los primeros en completar el TP 0.

Ojo que La fecha de entrega del TP0 es el martes 12 de septiembre.
$ tp0 -i entrada.txt -o -
Somos
Ojo
```

4.2. Interfaz

A fin de facilitar la corrección y prueba de los TPs, normalizaremos algunas de las opciones que deberán ser provistas por el programa:

- `-i`, o `--input`, permite especificar la ubicación del archivo de entrada, siendo `stdin` o cuando el argumento es "`-`", o bien cuando no haya sido especificado explícitamente en la línea de comandos (valor por defecto).
- `-o`, o `--output`, para definir la ubicación del archivo de salida en forma análoga al punto anterior. Por defecto, el programa deberá escribir sobre `stdout`. Lo mismo sucederá cuando el argumento pasado es "`-`".
- `-I`, o `--ibuf-bytes` determina el tamaño en bytes del *buffer* de entrada. El valor por defecto a usar es `1`.
- `-O`, o `--obuf-bytes` nos permite dimensionar el *buffer* se salida. El valor por defecto también es `1`.

5. Informe

El informe deberá incluir al menos las siguientes secciones:

- Documentación relevante al diseño e implementación del programa.
- Comandos para compilar el programa.
- Las corridas de prueba, con los comentarios pertinentes.
- El código fuente, el cual también deberá entregarse en formato digital compilable (incluyendo archivos de entrada y salida de pruebas).
- Este enunciado.

El informe deberá entregarse en formato impreso y digital.

6. Fechas

- Entrega: 26/9/2017.
- Vencimiento: 10/10/2017.