66.20 - Organización de Computadoras
Trabajo Práctico 0 - Infraestructura Básica

Nicolás Alvarez, Padrón 93503
Nicolás Fernandez, Padrón 88599
Andrew Parlane

6 de abril de 2018

Índice

1. Resumen

El objetivo de este trabajo se basa en familiarizarse con las herramientas de software que se utilizara en los trabajos futuros. Para simular el entorno de desarrollo que se utilizara en los trabajos, se usó el programa GXemul, el cual nos permite trabajar sobre una máquina MIPS que corre el sistema operativo NetBSD.

2. Introducción

Para este trabajo realizamos un programa en C que consiste en tomar un archivo de texto o bien stdin y generar una salida por stdout con cantidad de líneas, palabras y caracteres que contiene, junto con el nombre del archivo. Realizaremos varias ejecuciones con distintos archivos de prueba para evaluar los distintos resultados en las dos máquinas y compararlos.

3. Configuración entorno MIPS

Para configurar el emulador de MIPS GXEMUL (con la imagen descargada), los pasos que realizamos fueron los siguientes:

- En el directorio de la imagen ejecutamos el comando para extraerla

  ```
  host# bzip2 -dc gxemul-6620.tar.bz2 | cpio --sparse -i -v
  ```

- Luego verificamos que tengamos ssh instalado, para poder realizar el tunel de SSH.
- Iniciamos GXEMUL con el siguiente comando (parados en el directorio donde se extrajo):

  ```
  host# ./gxemul -e 3max -d netbsd-pmax.img
  ```

  Luego nos logueamos con usuario root y el password proporcionado por la cátedra.
- Configramos el loopback con el siguiente comando:

  ```
  host# sudo ifconfig lo:0 172.20.0.1
  ```

- Para conectarnos desde NetBSD con Linux utilizamos el siguiente comando:

  ```
  guest# ssh -R 2222:127.0.0.1:22 USER@172.20.0.1
  ```

  Donde USER es el usuario de Linux que utilizamos cada uno.
- Para conectarnos desde Linux con NetBSD utilizamos el siguiente comando:

  ```
  host# ssh -p 2222 root@127.0.0.1
  ```

- Para pasar archivos de Linux a NetBSD utilizamos el siguiente comando:

  ```
  host# scp -P 2222 <ARCHIVO> root@127.0.0.1:/root/<CARPETA>
  ```

- Para pasar archivos de NetBSD a Linux utilizamos el siguiente comando:

  ```
  guest# scp -p 2222 ARCHIVO USER@172.20.0.1:/DIR/DONDE/QUIERO/COPIAR
  ```

4. Compilacion y ejecución del programa

Esta sección describe los mecanismos para la compilación y ejecución del programa, tanto en entorno MIPS como en Linux. En una primera etapa compilamos y trabajamos en Linux para poder realizar las pruebas. Adicionalmente a esto utilizamos Valgrind para chequear que no tengamos problemas de leaks o free en nuestro código.

Para compilar manualmente:

- Estando parados en la carpeta donde se encuentran los archivos fuente, ejecutamos el siguiente comando:

  ```
  gcc -Wall -o tp0 *.c
  ```

Para compilar usando el Makefile:

- Estando parados en la carpeta donde se encuentran los archivos fuente, ejecutamos el siguiente comando:

  ```
  host# make
  ```

  Y para la machina de MIPS:

  ```
  host# gmake
  ```

Para ejecutar:

- Estando parados en la carpeta donde se encuentran el archivo ejecutable, corremos para cada archivo o fragmento de prueba lo siguiente:

  ```
  ./tp0 [options] file
  ```

- Si se quiere ver el help, que especifica las opciones disponibles al momento de invocar el programa:

  ```
  ./tp0 -h
  ```

- Si se quiere ver la versión del programa:

  ```
  ./tp0 -V
  ```

5. Implementación

5.1. Implementación en C

Para la implementación del programa, se estructuró en un solo archivo:

- **main.c** : Define el proceso principal de ejecución , la validación de los parámetros pasados al programa y además los métodos para parseo y salida del programa.

A continuación enumeramos las funciones definidas en el programa que se usarán luego de la verificacion y validación de los parámetros de entrada:

- **usage**
  parámetros: const char *nombreDeLaAplicacion

  descripción: función que muestra el help de la aplicación.

- **output**
  parámetros: bool cFlag
  bool wFlag
  bool lFlag
  uint32_t caracteres
  uint32_t palabras
  uint32_t lineas
  const char *nombreDelOutput

  descripción: función que muestra una linea de salida.

- **obtenerCharacter**
  parámetros: FILE *stream
  bool *whitespace
  bool *newline

  descripción: función que lee un carácter de ASCII o UTF8 desde un archivo.

- **parseStream**
  parámetros: FILE *stream
  uint32_t *chars
  uint32_t *palabras
  uint32_t *lineas

  descripción: función que cuenta las lineas, palabras y caracteres de un archivo.

6. Pruebas

Realizamos las pruebas en GXEmul para cada uno de los archivos pedidos tomando los tiempos de ejecución para cada archivo.

- **alice.txt**
- **beowulf.txt**
- **cyclopedia.txt**
- **elquijote.txt**

```
guest# time ./tp0 alice.txt
4046    30355   177412  alice.txt

real    0m0.324s
user    0m0.324s
sys     0m0.000s

guest# time ./tp0 beowulf.txt
4562    37048   224806  beowulf.txt

real    0m0.430s
user    0m0.379s
sys     0m0.051s

guest# time ./tp0 cyclopedia.txt
17926   105582  658543  cyclopedia.txt

real    0m1.246s
user    0m1.211s
sys     0m0.023s

guest# Tp0# time ./tp0 elquijote.txt
37862   384258  2155340 elquijote.txt

real    0m3.875s
user    0m3.844s
sys     0m0.031s
```

También se realizó una prueba para medir el tiempo que tarda el procesamiento de los 4 archivos juntos.

```
guest# time ./tp0 alice.txt beowulf.txt cyclopedia.txt elquijote.txt
4046    30355   177412  alice.txt
4562    37048   224806  beowulf.txt
17926   105582  658543  cyclopedia.txt
37862   384258  2155340 elquijote.txt
64396   557243  3216101 total

real    0m6.090s
user    0m6.043s
sys     0m0.031s
```

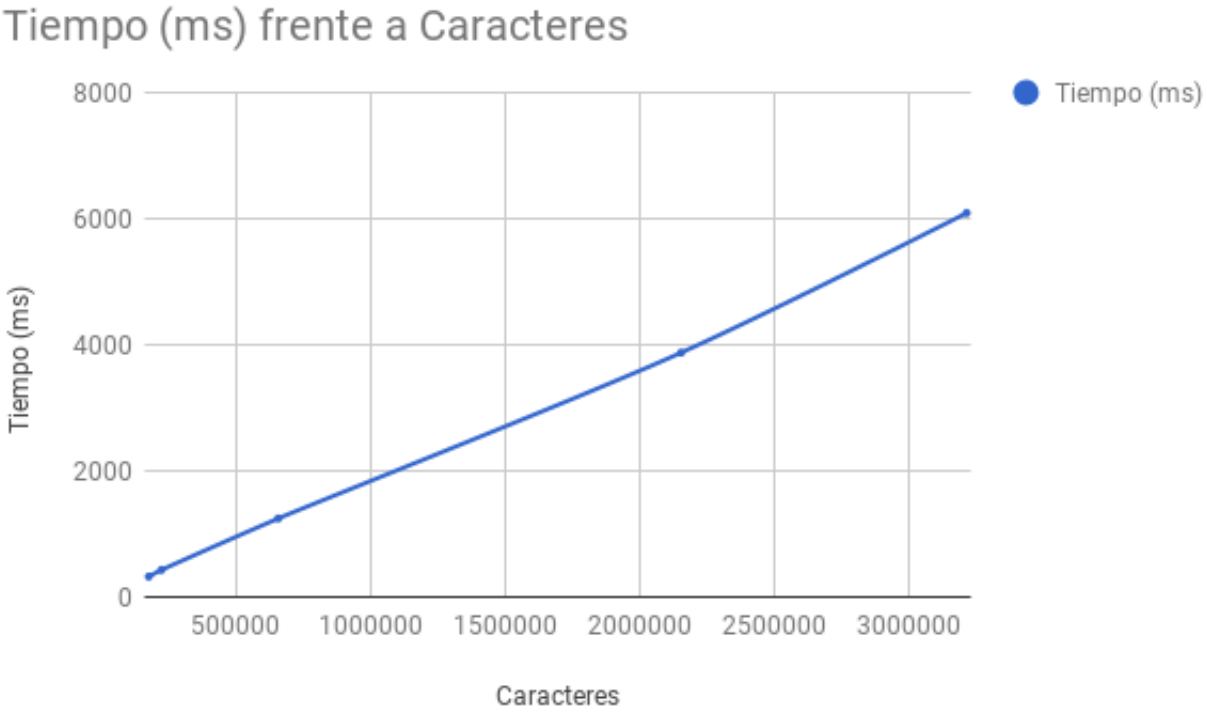
También se realizaron otras pruebas para aseguranos que todo funciona como se pide en el enunciado. Esas pruebas incluyen:

- Solo contar una de las lineas, palabras o los caracteres.
- Uso del flag -i para el input de la ruta de un archivo.
- Archivos en otros carpetas.

- Uso de wildcards en la ruta.
- Archivos que no existen.
- Archivos con nombres especiales: álíçé.txt, \"\".
- Archivos binarios.
- STDIN, usando cat y poniendo texto manualmente.
- Archivos infinitos como /dev/zero.

7. Gráficos en función del tamaño de entrada

Teniendo en cuenta los archivos y los datos del apartado anterior generamos los siguientes gráficos:



Calculamos la regresión lineal a partir de los datos y obtuvimos esta función: Y=-12.7545636573+0.00187008629677x

8. Comparación con WC

Realizamos las mismas pruebas en GXEmul usando WC.

```
guest# time wc -lwm alice.txt
        4046   30355  177428 alice.txt

real    0m0.281s
user    0m0.273s
sys     0m0.008s
guest# time wc -lwm beowulf.txt
    4562   37048  224839 beowulf.txt

real    0m0.352s
user    0m0.344s
sys     0m0.008s
guest# time wc -lwm cyclopedia.txt
        17926  105582  658543 cyclopedia.txt

real    0m1.090s
user    0m1.047s
sys     0m0.035s
guest# time wc -lwm elquijote.txt
   37862  384258 2198907 elquijote.txt

real    0m3.402s
user    0m3.375s
sys     0m0.016s
guest# time wc -lwm alice.txt beowulf.txt cyclopedia.txt elquijote.txt
    4046   30355  177428 alice.txt
    4562   37048  224839 beowulf.txt
   17926  105582  658543 cyclopedia.txt
   37862  384258 2198907 elquijote.txt
   64396  557243 3259717 total

real    0m5.293s
user    0m5.250s
sys     0m0.027s
```

	tp0			wc -lwm		
archivo	lineas	palabras	tiempo real	lineas	palabras	tiempo real
alice.txt	4046	30355	0.324s	4046	30355	0.281s
beowulf.txt	4562	37048	0.430s	4562	37048	0.352s
cyclopedia.txt	17926	105582	1.246s	17926	105582	1.090s
elquijote.txt	37862	384258	3.875s	37862	384258	3.402s
todos	64396	557243	6.090s	64396	557243	5.293s

Notar que no mostramos los resultados por caracteres porque el wc nativo en la maquina del guest no soporta caracteres, solo bytes.

Nuestra aplicación tp0 corre en promedio 15 % más lento que wc.

9. Conclusiones

El desarrollo de este trabajo nos permitió familiarizarnos con el uso del GXemul para lograr levantar una máquina virtual de MIPS y poder trabajar en ella. Aprendimos cómo crear un túnel entre la máquina virtual del GXemul y una pc mediante el loopback, con el cual pudimos realizar traspasos de archivos de uno a otro. Con esto logramos modificar el programa en la computadora, pasarlo a la máquina virtual y compilarlo en ella. En cuanto a los resultados, se puede observar que el tiempo que toma nuestro programa en procesar los archivos es mayor comparado con el tiempo de la funcion wc, ésto nos lleva a pensar que nuestro programa puede ser mejorable para poder alcanzar unos valores mas cercanos a wc.
Con el gráfico obtenido de los tiempos de corrida (en milisegundos) versus el tamaño de entrada (cantidad de caracteres) obtenemos que se parece a una recta. Observando la recta de regresión se puede ver que posee una pendiente pequeña (menor a 1), advirtiendonos de que cada 100 caracteres aproximadamente el tiempo de ejecución aumentara una milésima de segundo, es decir por cada 100.000 de caracteres aproximadamente tardará 1 segundo en procesarlos. Si bien la función wc posee mejores tiempos, nuestro programa posee un buen tiempo de ejecución.

10. Código C

10.1. main.c

```c
#include <stdio.h>
#include <unistd.h>
#include <stdbool.h>
#include <libgen.h>
#include <getopt.h>
#include <ctype.h>
#include <stdint.h>

#define MAJOR_VERSION    1
#define MINOR_VERSION    0

static const struct option long_options[] =
{
    {"help",         no_argument, 0, 'h' },
    {"version",      no_argument, 0, 'V' },
    {"lines",        no_argument, 0, 'l' },
    {"words",        no_argument, 0, 'w' },
    {"characters",   no_argument, 0, 'c' },
    {0,           0,                 0,  0  }
};

static void usage(const char *nombreDeLaAplicacion)
{
    printf("Usage:\n"
           "  %s -h\n"
           "  %s -V\n"
           "  %s [options]... [file]...\n"
           "Options:\n"
           "  -V, --version      Print version and quit.\n"
           "  -h, --help         Print this information.\n"
           "  -l, --lines        Print number of lines in file.\n"
           "  -w, --words        Print number of words in file.\n"
           "  -c, --characters   Print number of characters in file.\n"
           "  -i, --input        Path to input file.\n\n",
           nombreDeLaAplicacion, nombreDeLaAplicacion, nombreDeLaAplicacion);
    // necesitamos usar nombreDeLaAplicacion, nombreDeLaAplicacion, nombreDeLaAplicacion
    // porque no soportamos %1$s
}

static bool obtenerCharacter(FILE *stream, bool *whitespace, bool *newline)
{
    *whitespace = false;
    *newline = false;
    while (1)
    {
        int c = fgetc(stream);
        if (c == EOF)
        {
            return false;
        }

        uint8_t bytesMas;
        // 0xxx_xxxx => 1 byte
        // 110x_xxxx => 2 bytes
        // 1110_xxxx => 3 bytes
        // 1111_0xxx => 4 bytes
        // 1111_1xxx => invalido
        if (c < 128) // ascii
        {
            *whitespace = isspace(c);
            *newline = (c == '\n');
            return true;
        }
        else if (c < 0xE0)
        {
            bytesMas = 1;
        }
        else if (c < 0xF0)
        {
```

```
70              bytesMas = 2;
71          }
72          else
73          {
74              bytesMas = 3;
75          }
76
77          int i;
78          for (i = 0; i < bytesMas; i++)
79          {
80              c = fgetc(stream);
81              if (c == EOF)
82              {
83                  // invalido, no contamos
84                  return false;
85              }
86              // TODO whitespace en UTF8?
87          }
88
89          return true;
90      }
91  }
92
93  static void parseStream(FILE *stream, uint32_t *chars, uint32_t *palabras, uint32_t *lineas)
94  {
95      *chars = 0;
96      *palabras = 0;
97      *lineas = 0;
98      bool ultimoEstuvoWhitespace = true;
99
100     while (1)
101     {
102         bool whitespace;
103         bool newline;
104         if (!obtenerCharacter(stream, &whitespace, &newline))
105         {
106             // termina la palabra corriente
107             if (!ultimoEstuvoWhitespace )
108             {
109                 (*palabras)++;
110             }
111             break;
112         }
113
114         (*chars)++;
115
116         if (whitespace && !ultimoEstuvoWhitespace)
117         {
118             (*palabras)++;
119             ultimoEstuvoWhitespace = true;
120         }
121         if (!whitespace)
122         {
123             ultimoEstuvoWhitespace = false;
124         }
125
126         if (newline)
127         {
128             (*lineas)++;
129         }
130     }
131 }
132
133 void output(bool cFlag, bool wFlag, bool lFlag, uint32_t caracteres, uint32_t palabras,
        uint32_t lineas, const char *nombreDelOutput)
134 {
135     bool outputPrevio = false;
136     if (lFlag)
137     {
138         printf("%s%u", outputPrevio ? "\t" : "", lineas);
139         outputPrevio = true;
140     }
141     if (wFlag)
```

```
142     {
143         printf("%s%u", outputPrevio ? "\t" : "", palabras);
144         outputPrevio = true;
145     }
146     if (cFlag)
147     {
148         printf("%s%u", outputPrevio ? "\t" : "", caracteres);
149         outputPrevio = true;
150     }
151     if (nombreDelOutput != NULL)
152     {
153         printf("%s%s", outputPrevio ? "\t" : "", nombreDelOutput);
154     }
155     printf("\n");
156 }
157 
158 int main(int argc, char **argv)
159 {
160     // usamos argv[0] como el nombre del aplicacion
161     // pero solo queremos el archivo, no la ruta
162     const char *nuestroNombre = basename(argv[0]);
163 
164     bool lFlag = false;
165     bool wFlag = false;
166     bool cFlag = false;
167     const char *iValue = NULL;
168 
169     // clear errors
170     opterr = 0;
171 
172     // parse short options
173     while (1)
174     {
175         // obtener el siguinete argumento
176         int option_index = 0;
177         int c = getopt_long(argc, argv, "hVlwci:", long_options, &option_index);
178 
179         if (c == -1)
180         {
181             // no hay mas
182             break;
183         }
184 
185         switch (c)
186         {
187             case 'h':
188             {
189                 usage(nuestroNombre);
190                 // no siguimos desupes de -h
191                 return 0;
192             }
193             case 'V':
194             {
195                 printf("%s: Version %u.%u\n", nuestroNombre, MAJOR_VERSION, MINOR_VERSION);
196                 // no siguimos desupes de -V
197                 return 0;
198             }
199             case 'l':
200             {
201                 lFlag = true;
202                 break;
203             }
204             case 'w':
205             {
206                 wFlag = true;
207                 break;
208             }
209             case 'c':
210             {
211                 cFlag = true;
212                 break;
213             }
214             case 'i':
```

```
215             {
216                 iValue = optarg;
217                 break;
218             }
219             case '?':
220             {
221                 if (optopt == 'i')
222                 {
223                     fprintf(stderr, "Option '-%c' requires an argument.\n\n", optopt);
224                 }
225                 else if (isprint(optopt))
226                 {
227                     // es un argumento, pero no es uno que esperamos
228                     fprintf (stderr, "Unknown option '-%c'.\n\n", optopt);
229                 }
230                 else
231                 {
232                     // solo mostra el usage
233                 }
234                 usage(nuestroNombre);
235                 return 1;
236             }
237             default:
238             {
239                 // porque estamos aqui?
240                 usage(nuestroNombre);
241                 return 1;
242             }
243         }
244     }
245 
246     // si ningun de -l -w o -c esta especificado usamos todos
247     if (!lFlag && !wFlag && !cFlag)
248     {
249         lFlag = true;
250         wFlag = true;
251         cFlag = true;
252     }
253 
254     bool err = false;
255     uint32_t totalLineas = 0;
256     uint32_t totalPalabras = 0;
257     uint32_t totalcaracteres = 0;
258     uint32_t totalArchivos = 0;
259 
260     // si no hay archivos especificado usamos de stdin
261     if (iValue == NULL && (optind >= argc))
262     {
263         parseStream(stdin, &totalcaracteres, &totalPalabras, &totalLineas);
264         totalArchivos = 1;
265 
266         output(cFlag, wFlag, lFlag, totalcaracteres, totalPalabras, totalLineas, NULL);
267     }
268     else
269     {
270         // primero el archivo especificado con -i (si hay uno)
271         if (iValue != NULL)
272         {
273             FILE *stream = fopen(iValue, "rb");
274             if (stream == NULL)
275             {
276                 fprintf(stderr, "%s: %s: No such file or directory\n", nuestroNombre, iValue);
277                 err = true;
278             }
279             else
280             {
281                 uint32_t chars;
282                 uint32_t palabras;
283                 uint32_t lineas;
284                 parseStream(stream, &chars, &palabras, &lineas);
285                 fclose(stream);
286                 output(cFlag, wFlag, lFlag, chars, palabras, lineas, iValue);
287 
```

```
288                 totalcaracteres    += chars;
289                 totalPalabras       += palabras;
290                 totalLineas         += lineas;
291             }
292 
293             // conta el archivo si existe o no
294             totalArchivos++;
295         }
296 
297         // despues cada argumento que no es un opcion
298         int index;
299         for (index = optind; index < argc; index++)
300         {
301             FILE *stream = fopen(argv[index], "rb");
302             if (stream == NULL)
303             {
304                 fprintf(stderr, "%s: %s: No such file or directory\n", nuestroNombre, argv[
    index]);
305                 err = true;
306             }
307             else
308             {
309                 uint32_t chars;
310                 uint32_t palabras;
311                 uint32_t lineas;
312                 parseStream(stream, &chars, &palabras, &lineas);
313                 fclose(stream);
314                 output(cFlag, wFlag, lFlag, chars, palabras, lineas, argv[index]);
315 
316                 totalcaracteres    += chars;
317                 totalPalabras       += palabras;
318                 totalLineas         += lineas;
319             }
320 
321             // conta el archivo si existe o no
322             totalArchivos++;
323         }
324     }
325 
326     if (totalArchivos > 1)
327     {
328         output(cFlag, wFlag, lFlag, totalcaracteres, totalPalabras, totalLineas, "total");
329     }
330 
331     // return 0 si no habia errores, o 1 si habia
332     return err ? 1 : 0;
333 }
```

11. Código Makefile

11.1. Makefile

```
TARGET = tp0

MACHINE = $(shell uname -m)

LIBS =
CC = gcc
CFLAGS = -Wall
SOLO_ASM_FLAGS = -O0 -S

# solo podriamos usar mrnames en MIPS
ifeq ($(MACHINE),pmax)
        SOLO_ASM_FLAGS += -mrnames
endif

define HACE_PRUEBA
        $(1) wc $(2) $(4) >> wc_out 2>&1
        $(1) ./$(TARGET) $(3) $(4) >> tp0_out 2>&1
endef

default: $(TARGET)
all: default

OBJECTS = $(patsubst %.c, %.o, $(wildcard *.c))
ASSEMBLER = $(patsubst %.c, %.s, $(wildcard *.c))
HEADERS = $(wildcard *.h)

%.o: %.c $(HEADERS)
        $(CC) $(CFLAGS) -c $< -o $@

%.s: %.c $(HEADERS)
        $(CC) $(CFLAGS) $(SOLO_ASM_FLAGS) -c $<

$(TARGET): $(OBJECTS)
        $(CC) $(OBJECTS) -Wall $(LIBS) -o $@

asm: $(ASSEMBLER)

prueba: $(TARGET)
        -rm -f wc_out tp0_out
        # archivos normales
        -$(call HACE_PRUEBA, , -lwm, -i, pruebas/cyclopedia.txt)
        -$(call HACE_PRUEBA, , -lwm, -lwc, pruebas/beowulf.txt)
        -$(call HACE_PRUEBA, , -lwm, , pruebas/elquijote.txt)
        # solo lineas
        -$(call HACE_PRUEBA, , -l, -l, pruebas/cyclopedia.txt)
        -$(call HACE_PRUEBA, , -l, -li, pruebas/beowulf.txt)
        -$(call HACE_PRUEBA, , -l, -l, pruebas/elquijote.txt)
        # solo palabras
        -$(call HACE_PRUEBA, , -w, -w, pruebas/cyclopedia.txt)
        -$(call HACE_PRUEBA, , -w, -w, pruebas/beowulf.txt)
        -$(call HACE_PRUEBA, , -w, -wi, pruebas/elquijote.txt)
        # solo caracteres
        -$(call HACE_PRUEBA, , -m, -c, pruebas/cyclopedia.txt)
        -$(call HACE_PRUEBA, , -m, -c, pruebas/beowulf.txt)
        -$(call HACE_PRUEBA, , -m, -c, pruebas/elquijote.txt)
        # multiples archivos
        -$(call HACE_PRUEBA, , -lwm, -i, pruebas/cyclopedia.txt pruebas/beowulf.txt pruebas/elquijote.txt
        -$(call HACE_PRUEBA, , -lwm, , pruebas/cyclopedia.txt pruebas/beowulf.txt pruebas/elquijote.txt)
        # wildcard
        -$(call HACE_PRUEBA, , -lwm, , pruebas/*.txt)
        # archivos que no existe
        -$(call HACE_PRUEBA, , -lwm, , pruebas/esto/archivo/no.existe)
        -$(call HACE_PRUEBA, , -lwm, , pruebas/esto/archivo/no.existe este.tampoco)
        # archivos / rutas con caracteres especiales
        -$(call HACE_PRUEBA, , -lwm, , pruebas/abc\ def)
        -$(call HACE_PRUEBA, , -lwm, , pruebas/\"\")
        -$(call HACE_PRUEBA, , -lwm, , "pruebas/alice.txt")
        # archivos binarios
        -$(call HACE_PRUEBA, , -lwm, , pruebas/testBin)
```

```
	# stdin
	-$(call HACE_PRUEBA, cat pruebas/*.txt |, -lwm, -lwc)
	-$(call HACE_PRUEBA, cat pruebas/*.txt |, -lwm, )
	# newlines
	-$(call HACE_PRUEBA, cat pruebas/newlines.txt |, -lwm, -lwc)
	@# Estoy poniendo esto aqui para estar cerca las otras pruebas
	@# Estos estan pruebas que no puedo automizar
	# TODO manualmente
	# Archivos infinitos
	# ./tp0 /dev/zero
	# stdin desde usario
	# ./tp0

clean:
	-rm -f *.o
	-rm -f *.s
	-rm -f $(TARGET)
	-rm -f wc_out tp0_out

.PHONY: default all asm prueba clean
```

12. Enunciado

**Ver hojas anexadas*