Trabajo Práctico 1:
Conjunto de instrucciones MIPS

Santiago Aguilera, *Padrón Nro. 95795*
marquito.santi@gmail.com
Agustina Barbetta, *Padrón Nro. 96528*
agustina.barbetta@gmail.com
Manuel Porto, *Padrón Nro. 96587*
manu.porto94@hotmail.com

2do. Cuatrimestre de 2016
66.20 Organización de Computadoras
Facultad de Ingeniería, Universidad de Buenos Aires

30 de noviembre de 2016

Resumen

El presente trabajo práctico consiste en crear parte de una aplicación, en MIPS32, capaz de dibujar los conjuntos fractales de Julia sobre una imagen de formato PGM.

El código del trabajo se encuentra en el siguiente repositorio: https://github.com/saantiaguilera/fiuba-orga-pc-julia-set.

Índice

1. Descripción

El objetivo del presente trabajo es familiarizarse con el conjunto de instrucciones MIPS y el concepto de ABI, extendiendo un programa que resuelva el problema descripto en secciones posteriores.

2. Programa

El trabajo consiste una modificacion de una seccion de un programa introducido en el TP0, que se encargaba de dibujar el conjunto de Julia y sus vecindades. La logica de computo del fractal debera tener soporte nativo para MIPS32 sobre NetBSD/pmax.

2.1. Diseño

A continuacion se describen los distintos componentes de la solucion.

2.1.1. mips32_plot

La rutina `mips32_plot` es una traducción del código C a la arquitectura MIPS32. La misma se encarga de el cálculo de color para cada píxel de la imagen y la escritura del archivo `.pgm`.

Desde el punto de vista del usuario el programa funciona de manera idéntica al presentado en el primer trabajo. Sin embargo, para reducir la cantidad de `syscall`s utilizados se implemento un *buffer* que almacena una determinada cantidad de elementos y solamente cuando el mismo se llena o termina la rutina, realiza la escritura del archivo. La razón de esta implementación es que las llamadas realizadas para escribir el archivo son costosas en tiempo, por lo que se busca reducirlas implementando un *buffer* que escriba varios caracteres a la vez.

2.1.2. write_buffer

Para implementar la lógica de la carga del *buffer* y la escritura del archivo, se intentaron distintas soluciones. Una de ellas fue implementar la misma como una macro que se expandiría en su correspondiente código cada vez que fuera utilizada. Sin embargo, la versión de MIPS utilizada no soporta esta característica por lo que no fue posible implementar esta solución.

La solución alternativa consistió en dejar la lógica debajo de un *label* y saltar a ella cada vez que se necesite.

2.1.3. Stack Frame

8260	*a3*
8256	*a2*
8252	*a1*
8248	*a0 (param)*
8244	**fp**
8240	**gp**
8236	cpi
8232	cpr
8228	c
8224	y
8220	x
8216	si
8212	sr
8208	zi
8204	zr
8200	ci
8196	cr
8192	buff_len
8188	buff (end)
8184	
8180	
...	...
8	
4	
0	buff (start)

Cuadro 1: *Stack frame* de la función `mips32_plot()`. Las palabras 8248 a 8260 corresponden al *Stack frame* de la función `caller`, las palabras en negrita corresponden a la *General Register Save Area* y los restantes a la *Local and Temporary Variables Area*.

2.2. Compilación

El código fuente provisto por la cátedra provee los makefiles necesarios para compilar el ejecutable a partir de la versión en C con el archivo `mips32_plot.c`. Para poder compilar el código desarrollado se debió cambiar la definición en el archivo Makefile 6:

`SRCS = mips32_plot.c main.c mygetopt_long.c`

por

`SRCS = mips32_plot.S main.c mygetopt_long.c`

Luego deberán invocar la siguiente secuencia de comandos para limpiar los archivos temporales y generar los nuevos Makefiles:

```
> make clean
> make makefiles
> make
```

Para la compilación del programa se cuenta con un *Makefile*, el mismo cuenta con un *phony target* para limpiar los archivos generados (`make clean`) y el target *phony all* que (por más que sea *phony*) generará un ejecutable '`tp0`' y los archivos del *linker* necesarios (`make` o `make all`).

2.3. Pruebas

Se creó una *test-suite* en bash para testear casos borde de los parámetros y que el programa funcione correctamente.

Para correrla simplemente hay que compilar el proyecto previamente, darle permisos al *script* (de ser necesarios, `chmod u+x run_tests.sh`) y correrlo de forma normal (`bash run_tests.sh`). El mismo irá dando información sobre cada test corrido y su estado.

Las mismas se encargan de probar el pasaje de argumentos en casos particulares (pasar caracteres en vez de numeros, poner datos no respetando el formato, entre otros) y a su vez de correr el caso de exito. Actualmente, el programa cuenta con una limitacion, la cual es que, si se cambia el size del buffer a "1"(o un numero menor al size de un color, el cual puede ir entre 1 y 3 bytes), el buffer nunca podra llenarse y por consiguiente no se escribira nada. Por lo cual se tiene como pre condicion que el size del buffer sea siempre mayor o igual a 4 (porque consideramos tambien el salto de linea)

3. Conclusiones

La realización de este trabajo fue importante para entender y aplicar los temas vistos en clase sobre MIPS y su ABI. A su vez, se pudo ver detalladamente el funcionamiento de un programa assembly, y particularmente el funcionamiento del stack en el mismo.

4. Apéndices

4.1. Código fuente

El código a continuación se encuentra también en el repositorio del trabajo: `https://github.com/saantiaguilera/fiuba-orga-pc-julia-set`.

Listing 1: mips32_plot.S

```
#include <mips/regdef.h>
#include <sys/syscall.h>

#define ASCII_0 48

#define BUF_SZ 8192

#define STACK_SZ BUF_SZ+56

#define A0_OFFSET BUF_SZ+56
#define FP_OFFSET BUF_SZ+52
#define GP_OFFSET BUF_SZ+48
#define CPI_OFFSET BUF_SZ+44
#define CPR_OFFSET BUF_SZ+40
#define C_OFFSET BUF_SZ+36
```

```
16 #define Y_OFFSET BUF_SZ+32
   #define X_OFFSET BUF_SZ+28
18 #define SI_OFFSET BUF_SZ+24
   #define SR_OFFSET BUF_SZ+20
20 #define ZI_OFFSET BUF_SZ+16
   #define ZR_OFFSET BUF_SZ+12
22 #define CI_OFFSET BUF_SZ+8
   #define CR_OFFSET BUF_SZ+4
24 #define BUFF_LEN_OFFSET BUF_SZ
   #define BUFF_OFFSET 0
26
   #define PARAM_UL_RE 0
28 #define PARAM_UL_IM 4
   #define PARAM_LR_RE 8
30 #define PARAM_LR_IM 12
   #define PARAM_D_RE 16
32 #define PARAM_D_IM 20
   #define PARAM_CP_RE 24
34 #define PARAM_CP_IM 28
   #define PARAM_X_RES 32
36 #define PARAM_Y_RES 36
   #define PARAM_SHADES 40
38 #define PARAM_FD 44

40     .text
       .abicalls
42     .globl mips32_plot
       .ent mips32_plot
44

46 mips32_plot:
       ################################################################
48     # Push stack frame
       subu sp, sp, STACK_SZ   # push stack pointer down
50
       sw $fp, FP_OFFSET(sp)   # store $fp on mem(sp + FP_OFFSET)
52     sw gp, GP_OFFSET(sp)    # store gp on mem(sp + GP_OFFSET)

54     move $fp, sp            # $fp = sp

56     sw a0, A0_OFFSET($fp)   # store params on mem($fp + A0_OFFSET)

58     l.s $f0, PARAM_CP_RE(a0)
       s.s $f0, CPR_OFFSET($fp)    # store cpr on mem($fp + CPR_OFFSET)
60     #lw t0, PARAM_CP_RE(a0) # cpr = params->cp_re on t0
       #sw t0, CPR_OFFSET($fp) # store cpr on mem($fp + CPR_OFFSET)
62     l.s $f0, PARAM_CP_IM(a0)
       s.s $f0, CPI_OFFSET($fp)    # store cpr on mem($fp + CPR_OFFSET)
64     #lw t0, PARAM_CP_IM(a0) # cpi = params->cp_im on t2
       #sw t0, CPI_OFFSET($fp) # store cpi on mem($fp + CPI_OFFSET)
66     li t0, 0                # t0 = 0
       sw t0, C_OFFSET($fp)    # store c = 0 on mem($fp + C_OFFSET)
68     sw t0, X_OFFSET($fp)    # store x = 0 on mem($fp + X_OFFSET)
```

```
    sw t0, Y_OFFSET($fp)    # store y = 0 on mem($fp + Y_OFFSET)
    sw t0, SI_OFFSET($fp)   # store si = 0
    sw t0, SR_OFFSET($fp)   # store sr = 0
    sw t0, ZI_OFFSET($fp)   # store zi = 0
    sw t0, ZR_OFFSET($fp)   # store zr = 0
    # This change its not really necessary, so it can be reverted
    l.s $f0, PARAM_UL_IM(a0)    # ci = parms->UL_im on t0
    #lw t0, PARAM_UL_IM(a0) # ci = parms->UL_im on t0
    s.s $f0, CI_OFFSET($fp) # store ci on mem($fp + CI_OFFSET)
    #sw t0, CI_OFFSET($fp)  # store ci on mem($fp + CI_OFFSET)
    l.s $f0, PARAM_UL_RE(a0)    # cr = parms->UL_re on t0
    #lw t0, PARAM_UL_RE(a0) # cr = parms->UL_re on t0
    s.s $f0, CR_OFFSET($fp) # store cr on mem($fp + CR_OFFSET)
    #sw t0, CR_OFFSET($fp)  # store cr on mem($fp + CR_OFFSET)

    li t0, 0
    sw t0, BUFF_LEN_OFFSET($fp) # store buff_len = 0

    ################################################################

    ################################################################
    # Store PGM header on buffer, this is "P2\nx_res\ny_res\nshades\n"

    addiu t0, $fp, BUFF_OFFSET  # buff_addr on t0
    lw t1, BUFF_LEN_OFFSET($fp)     # buff_len on t1

title:
    li t2, 0            # i = 0
title_loop:
    li t3, 3
    bge t2, t3, x_res   # if (i >= 3) finish
    la t3, header
    addu t3, t3, t2
    lb t3, 0(t3)        # header[i]
    addu t4, t0, t1
    sb t3, 0(t4)        # store header[i] on buff_addr[buff_len]
    addiu t1, t1, 1     # buff_len++
    addiu t2, t2, 1     # i++
    j title_loop

x_res:
    lw t2, PARAM_X_RES(a0)  # load parms->x_res on t2
    move t7, t2             # t7 = t2
    li t3, 10
    li t4, -1               # t4 = x_res_len
x_res_len:
    div t7, t3              # x_res /= 10
    addi t4, t4, 1          # x_res_len++
    bne t7, zero, x_res_len # if not zero, loop
    add t1, t1, t4          # start at the end
loop_x:
    div  t2, t3             # x_res /= 10
    mflo t2
```

```
122     mfhi t5                 # get remainder
        addi t5, t5, ASCII_0    # convert to ASCII digit
124     addu t6, t0, t1         # t6 = buff_addr + buff_len
        sb   t5, 0(t6)          # store it
126     sub t1, t1, 1           # adjust buf ptr
        bne  t2, zero, loop_x   # if not zero, loop
128     add t1, t1, t4          # correct buf ptr so it points the x_res end
        addi t1, t1, 2          # two more so \n doesnt overwrite last char
130
        la t2, line_break       # load line_break addr
132     lb t2, 0(t2)            # load line_break
        addu t3, t0, t1
134     sb t2, 0(t3)            # store line_break on buff
        addiu t1, t1, 1         # buff_len++
136
    y_res:
138     lw t2, PARAM_Y_RES(a0)
        move t7, t2             # t7 = t2
140     li t3, 10
        li t4, -1               # t4 = y_res_len
142 y_res_len:
        div t7, t3              # y_res /= 10
144     addi t4, t4, 1          # y_res_len++
        bne t7, zero, y_res_len # if not zero, loop
146     add t1, t1, t4          # start at the end
    loop_y:
148     div  t2, t3             # y_res /= 10
        mflo t2
150     mfhi t5                 # get remainder
        addi t5, t5, ASCII_0    # convert to ASCII digit
152     addu t6, t0, t1         # t6 = buff_addr + buff_len
        sb   t5, 0(t6)          # store it
154     sub t1, t1, 1           # adjust buf ptr
        bne  t2, zero, loop_y   # if not zero, loop
156     add t1, t1, t4          # correct buf ptr so it points the x_res end
        addi t1, t1, 2          # two more so \n doesnt overwrite last char
158
        la t2, line_break       # load line_break addr
160     lb t2, 0(t2)            # load line_break
        addu t3, t0, t1
162     sb t2, 0(t3)            # store line_break on buff
        addiu t1, t1, 1         # buff_len++
164
    shades:
166     lw t2, PARAM_SHADES(a0)
        move t7, t2             # t7 = t2
168     li t3, 10
        li t4, -1               # t4 = shds_len
170 shds_len:
        div t7, t3              # shds /= 10
172     addi t4, t4, 1          # shds_len++
        bne t7, zero, shds_len  # if not zero, loop
174     add t1, t1, t4          # start at the end
```

```
loop_shds:
    div  t2, t3             # shds /= 10
    mflo t2
    mfhi t5                 # get remainder
    addi t5, t5, ASCII_0    # convert to ASCII digit
    addu t6, t0, t1         # t6 = buff_addr + buff_len
    sb   t5, 0(t6)          # store it
    sub t1, t1, 1           # adjust buf ptr
    bne t2, zero, loop_shds # if not zero, loop
    add t1, t1, t4          # correct buf ptr so it points the x_res end
    addi t1, t1, 2          # two more so \n doesnt overwrite last char

    la t2, line_break       # load line_break addr
    lb t2, 0(t2)            # load line_break
    addu t3, t0, t1
    sb t2, 0(t3)            # store line_break on buff
    addiu t1, t1, 1         # buff_len++


    sw t1, BUFF_LEN_OFFSET($fp)

    ################################################################

    ################################################################
    # Barremos la regián rectangular del plano complejo comprendida
    # entre (parms->UL_re, parms->UL_im) y (parms->LR_re, parms->LR_im).
    # El parámetro de iteración es el punto (cr, ci).



    li t4, 0            # y = 0 on t4
    sw t4, Y_OFFSET($fp)

    lw t0, A0_OFFSET($fp)   # parms on t0
    l.s $f0, PARAM_UL_IM(t0)    # ci = parms->UL_im
    s.s $f0, CI_OFFSET($fp)

UL_im_loop:
    lw t4, Y_OFFSET($fp)    # y on t4

    lw t0, A0_OFFSET($fp)   # parms on t0
    lw t6, PARAM_Y_RES(t0)  # parms->y_res on t6

    bge t4, t6, flush_end   # if (y < parms->y_res) continue

    li t7, 0            # x = 0 on t7
    sw t7, X_OFFSET($fp)    # store x on mem($fp + X_OFFSET)

    l.s $f0, PARAM_UL_RE(t0)    # cr = parms->UL_re
    s.s $f0, CR_OFFSET($fp)

UL_re_loop:
    lw t7, X_OFFSET($fp)    # x on t7
```

```
228
    lw t0, A0_OFFSET($fp)   # parms on t0
230     lw t6, PARAM_X_RES(t0)  # parms->x_res on t6
    bge t7, t6, UL_im_icr   # if (x < parms->x_res) continue
232

234     l.s $f0, CI_OFFSET($fp) # ci on f0
    mov.s $f2, $f0      # zi = ci
236     s.s $f2, ZI_OFFSET($fp)

238     l.s $f1, CR_OFFSET($fp) # cr on f1
    mov.s $f3, $f1      # zr = cr
240     s.s $f3, ZR_OFFSET($fp)

242     ################################################################
    # Determinamos el nivel de brillo asociado al punto
244     # (cr, ci), usando la fármula compleja recurrente
    # f = f^3 + c.
246
    li t9, 0                # c = 0 on t9
248     sw t9, C_OFFSET($fp)
shades_loop:
250     lw t0, A0_OFFSET($fp)   # parms on t0
    lw t6, PARAM_SHADES(t0) # parms->shades on t6
252
    lw t9, C_OFFSET($fp)
254     beq t9, t6, update_buffer   # if (c < parms->shades) continue

256     mul.s $f4, $f2, $f2     # zi * zi
    mul.s $f5, $f3, $f3     # zr * zr
258
    add.s $f6, $f4, $f5     # absz =  zr**2 + zi**2
260     li.s $f7, -4
    add.s $f6, $f6, $f7     # absz - 4 on t8
262     li.s $f8, 0
    c.le.s $f8, $f6
264     bc1t update_buffer

266     #### Save zr zi in stack
    s.s $f3, ZR_OFFSET($fp)
268     s.s $f2, ZI_OFFSET($fp)
    ####
270
    # sr = zr*zr - zi*zi + cpr
272     sub.s $f2, $f5, $f4     # zr**2 - zi**2 on t4
    l.s $f3, CPR_OFFSET($fp)
274     add.s $f2, $f2, $f3         # (zr**2 - zi**2) + cpr
    s.s $f2, SR_OFFSET($fp) # save sr in stack
276
    #### Restore zr zi
278     l.s $f3, ZR_OFFSET($fp)
    l.s $f2, ZI_OFFSET($fp)
280     ####
```

```
282     # si = 2 * zr * zi + cpi
        mul.s $f4, $f2, $f3     # (zr * zi)
284     li.s $f5, 2
        mul.s $f4, $f4, $f5
286     l.s $f5, CPI_OFFSET($fp)
        add.s $f4, $f4, $f5         # (2*zr*zi) + cpi
288     s.s $f4, SI_OFFSET($fp) # save si in stack

290     l.s $f3, SR_OFFSET($fp) # zr = sr
        l.s $f2, SI_OFFSET($fp) # zi = si
292
        addi t9, t9, 1          # ++c
294     sw t9, C_OFFSET($fp)    # save c in stack
        j shades_loop
296
        ################################################################
298
    update_buffer:
300     lw t1, BUFF_LEN_OFFSET($fp) # buff_len on t1

302     # Hay lugar para 2 bytes en el buffer?
        li t4, BUF_SZ
304     subu t4, t4, t1             # BUF_SZ - buff_len
        li t5, 4
306     bge t4, t5, store_shade     # if (BUF_SZ - buff_len >= 4) KB!! store

308 write_buffer:
        li v0, SYS_write            # system call for write to file
310     lw t0, A0_OFFSET($fp)       # parms on t0
        lw a0, PARAM_FD(t0)         # file descriptor
312     addiu a1, $fp, BUFF_OFFSET  # address of buffer from which to write
        lw a2, BUFF_LEN_OFFSET($fp) # buffer length
314     syscall                     # write to file
        bltz v0, io_error           # if v0 < 0, io error
316
        sw zero, BUFF_LEN_OFFSET($fp)   # reset buffer
318
    store_shade:
320     addiu t1, $fp, BUFF_OFFSET  # buff_addr on t1
        lw t2, BUFF_LEN_OFFSET($fp) # buff_len on t2
322     lw t3, C_OFFSET($fp)        # recover c from stack
        move t4, t3
324     li t5, 10
        li t6, -1                   # t6 = shade_len
326 shade_len:
        div t4, t5                  # shade /= 10
328     addi t6, t6, 1              # shade_len++
        bnez t4, shade_len          # if shade != 0, loop
330     add t2, t2, t6              # start at the end
    loop_shade:
332     div t3, t5                  # shade /= 10
        mflo t3
```

```
334     mfhi t7                     # get remainder
        addi t7, t7, ASCII_0        # ascii conversion
336     addu t8, t1, t2             # t8 = buff_addr (t1) + buff_len (t2)
        sb t7, 0(t8)                # store it
338     sub t2, t2, 1               # adjust buf ptr
        bnez t3, loop_shade         # if c != 0, loop
340     add t2, t2, t6              # correct buf ptr so it point at shade end
        addi t2, t2, 2              # two more so \n doesnt overwrite last char
342
        la t3, line_break
344     lb t3, 0(t3)
        addu t4, t1, t2
346     sb t3, 0(t4)                # buff_addr[buff_len] = \n
        addiu t2, t2, 1             # buff_len++
348
        sw t2, BUFF_LEN_OFFSET($fp) # update buff_len on stack
350
    UL_re_icr:
352     lw t7, X_OFFSET($fp)
        addi t7, t7, 1      # ++x
354     sw t7, X_OFFSET($fp)    # store x on mem($fp + X_OFFSET)

356     l.s $f1, CR_OFFSET($fp)
        l.s $f4, PARAM_D_RE(t0) # parms->d_re on t6
358     add.s $f1, $f1, $f4     # cr += parms->d_re
        s.s $f1, CR_OFFSET($fp) # store cr on mem($fp + CR_OFFSET)
360     j UL_re_loop

362 UL_im_icr:
        lw t4, Y_OFFSET($fp)
364     addi t4, t4, 1      # ++y
        sw t4, Y_OFFSET($fp)    # store y on mem($fp + Y_OFFSET)
366
        l.s $f4, PARAM_D_IM(t0) # parms->d_im on t6
368     l.s $f0, CI_OFFSET($fp)
        sub.s $f0, $f0, $f4     # ci -= parms->d_im
370     s.s $f0, CI_OFFSET($fp) # store ci on mem($fp + CI_OFFSET)
        j UL_im_loop
372
    flush_end:
374     li v0, SYS_write            # system call for write to file
        lw t0, A0_OFFSET($fp)       # parms on t0
376     lw a0, PARAM_FD(t0)         # file descriptor
        addiu a1, $fp, BUFF_OFFSET  # address of buffer from which to write
378     lw a2, BUFF_LEN_OFFSET($fp) # buffer length
        syscall                     # write to file
380     bltz v0, io_error           # if v0 < 0, io error
        j success
382
    io_error:
384     li v0, SYS_write
        li a0, 2
386     la a1, error_msg
```

```
    li a2, 11
388 syscall

390 li v0, -1           # return value -1
    j return
392
success:
394 li v0, 0            # return value 0

396 return:
    ##################################################################
398 # Pop stack frame
    move sp, $fp
400 lw $fp, FP_OFFSET(sp)   # loads mem(sp + FP_OFFSET) on $fp
    lw gp, GP_OFFSET(sp)    # loads mem(sp + GP_OFFSET) on gp
402 addiu sp, sp, STACK_SZ  # pop stack pointer up
    ##################################################################
404
    jr ra
406
    .end mips32_plot
408
    .data
410 header:
    .asciiz "P2\n"
412 line_break:
    .ascii "\n"
414 error_msg:
    .asciiz "i/o error.\n"
```

4.2. Enunciado

Universidad de Buenos Aires - FIUBA
66.20 Organización de Computadoras
Trabajo práctico 1: conjunto de instrucciones MIPS
2^o cuatrimestre de 2016

`$Date: 2016/10/02 22:23:34 $`

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, en la fecha estipulada, con una carátula que contenga los datos completos de todos los integrantes, un informe impreso de acuerdo con lo que mencionaremos en la sección 5, y con una copia digital de los archivos fuente necesarios para compilar el trabajo.

4. Descripción

Se trata de un modificar un programa que dibuje el conjunto de Julia y sus vecindades introducido en el *TP0* [1], en el cual la lógica de cómputo del fractal deberá tener soporte nativo para MIPS32 sobre NetBSD/pmax.

El código fuente con la versión inicial del programa, se encuentra disponible en [2]. El mismo deberá ser considerado como punto de partida de todas las implementaciones.

4.1. Soporte para MIPS

El entregable producido en este trabajo deberá implementar la lógica de cómputo del fractal en assembly MIPS32, con soporte nativo para NetBSD/pmax.

Para ello, cada grupo deberá tomar el código fuente de base para este TP, [2], y reescribir la función `mips32_plot()` sin cambiar su API. Esta función está ubicada en el archivo `mips32_plot.c`.

4.2. Casos de prueba

El informe trabajo práctico deberá incluir una sección dedicada a verificar el funcionamiento del código implementado. Para ello, será necesario escribir pruebas orientadas a probar el programa completo, ejercitando los casos más comunes de funcionamiento, los casos de borde, y también casos de error.

4.3. Compilación

El código fuente provisto por la cátedra provee los makefiles necesarios para compilar el ejecutable a partir de la versión en C con el archivo `mips32_plot.c`. Para poder compilar el código desarrollado deberán cambiar la definición en el archivo `Makefile.in` la línea número 6:

```
SRCS = mips32_plot.c main.c mygetopt_long.c
```

por

```
SRCS = mips32_plot.S main.c mygetopt_long.c
```

Luego deberán invocar la siguiente secuencia de comandos para limpiar los archivos temporales y generar los nuevos Makefiles:

```
$ make clean
$ make makefiles
$ make
```

4.4. Detalles de la implementación

Para optimizar los accesos a las llamadas a servicio del sistema (`syscalls`), deben utilizar un buffer de `BUF_SZ` bytes para escribir los datos de salida para luego ser enviados al archivo de salida. El tamaño `BUF_SZ` debe ser configurable, en tiempo de compilación, mediante un `#define`.

```
#ifndef BUF_SZ 8192
#define BUF_SZ 8192
#endif
```

Como podemos ver arriba, el valor por defecto de este parámetro es `8192` bytes.

5. Informe

El informe, a entregar en formarto impreso y digital[1] deberá incluir:

- Documentación relevante al diseño e implementación del código esarrollado para adaptar el programa. Incluír el diagrama de *stack frame* de las funciones implementadas en MIPS32.
- Documentación relevante al proceso de compilación: cómo obtener el ejecutable a partir de los archivos fuente. Especificar modificaciones realizadas a los archivos provistos por la cátedra si es que los hubo.
- Las corridas de prueba, con los comentarios pertinentes.[2]
- El código fuente, en lenguaje C (y MIPS32 donde corresponda)
- Este enunciado.

[1] En CD, DVD o memoria flash.

[2] Las pruebas provistas deben ejecutarse correctamente en NetBSD sobre MIPS32 sin modificación alguna.

6. Fecha de entrega

La fecha de vencimiento será el Martes 01/11.

Referencias

[1] Trabajo Práctico 0, 2do cuatrimestre de 2016.
`https://groups.yahoo.com/neo/groups/orga-comp/files/TPs/tp0-2016-2q.pdf`

[2] Código fuente con el esqueleto del trabajo práctico.
`https://drive.google.com/open?id=0B93s6e6NY_j1TFV2TFBqbUNKZ3M`

Referencias

[1] *GXemul*
http://gavare.se/gxemul/

[2] *The NetBSD project*
http://www.netbsd.org/

[3] *Conjunto de Julia*
https://es.wikipedia.org/wiki/Conjunto_de_Julia