Universidad de Buenos Aires - FIUBA

66:20 Organización de Computadoras

Trabajo práctico 1: Conjunto de instrucciones MIPS

Fonseca, Matias (98591)

Bacigaluppo, Ivan (98064)

Irrazabal, Diego (98125)

8 de Mayo del 2018

Índice

1. Documentación

a. Diseño

En este TP el objetivo es familiarizarse con el conjunto de instrucciones de MIPS y la ABI. La idea de la aplicación es la misma que el tp0, dibujar el conjunto Julia y sus vecindades, pero esta vez se nos entregó una implementación en ya hecha en C. El trabajo se basó en implementar en MIPS la función *mips32_plot()* que es la encargada escribir en el archivo los valores de los fractales. Originalmente, el programa debía poder seleccionar de forma dinámica la implementación a usar, en C o en MIPS, pero esto no funcionó debido a algunos errores de compilación. Entonces, lo que hicimos fue reemplazar la implementación en C por la nuestra en MIPS.

Comenzamos el trabajo implementando en forma separada una función que imprima los encabezados del archivo. Así, *print_headers()* toma los parámetros de resolución y cantidad máxima de sombreado y lo escribe en el archivo. A su vez, también imprime una *string* requerida por el formato *pgm*, agregada de forma manual en la función. Todo esto lo hace por medio de dos funciones: *print_number()* toma un número y llama a otra función *inttostr()*. Esta guarda en un buffer de caracteres cada uno de sus digitos, pasados a *char*. Esto debemos hacerlo debido a que la *syscall* escribe en archivos información solo a partir de un buffer, y por eso debemos guardar en este cada número para imprimirlo; la otra función es print_barran(). Como dice su nombre lo único que hace es imprimir el carácter de cambio de línea '\n' en el archivo.

El siguiente paso fue implementar el algoritmo. Nuestra función *get_shades() es* la encargada de encapsular esto. La función trabaja usando los mismos 3 ciclos de la implementación en C, llamando a las mismas *print_number()* y *print_barran()* al final de cada vuelta para imprimir el valor calculado.

b. Implementación

En esta sección describiremos únicamente las funciones trabajadas. Es decir, no tomamos en cuenta la parte del código ya implementada entregada junto con el enunciado, que después de todo no ha sido tocada.

- `void mips32_plot(param_t *parms);`

  Implementada en C. Recibe el struct de paramétros implementado por la cátedra. Llama a la implementación del algoritmo en MIPS. No tiene valor de retorno.

- `int print_header(int fd, int x_res, int y_res, int shades);`

  Implementada en asm MIPS. Recibe el f*ile descriptor* del archivo abierto, el ancho y alto de la resolución, y el nivel máximo de sombreado. Imprime los encabezados del archivo pgm. Devuelve 1 si no hubo problemas, un valor negativo (representativo del código) si hubo algún error.

- `int print_number(int fd, int número);`

  Implementada en asm MIPS. Recibe el *file descriptor* del archivo abierto y el número a imprimir. Devuelve 1 si no hubo problemas, un valor negativo (representativo del código) si hubo algún error.

- `int print_barran(int fd);`

  Implementada en asm MIPS. Recibe el *file descriptor* del archivo abierto. Imprime en el archivo el carácter de cambio de línea '\n'. Devuelve 1 si no hubo problemas, un valor negativo (representativo del código) si hubo algún error.

- `char* inttostr(int número);`

  Implementada en asm MIPS. Recibe un número entero. Convierte el número en una cadena de caracteres dígito a dígito. Devuelve un puntero al buffer donde se almacenó la *string*.

- `int get_shades(param_t* params, int fd);`

  Implementada en asm MIPS. Recibe el struct de parámetros y el *file descriptor* del archivo ya abierto. Procesa los valores del struct con el algoritmo de vecindades de Julia e imprime en el archivo los encabezados y las sombras calculadas que generen la imagen, que deberá ser formato pgm para ser visualizada. Devuelve 1, o un valor negativo en caso de error.

c. Compilación

Para el proceso de compilación de la función implementada en Mips, se utilizó Gxemul para emular un procesador MIPS R3000 con una imagen de sistema operativo NETBSD 3.0.
Este se ejecutó con soporte X11, mediante la generación de un túnel entre el HostOS y el GuestOS con SSH/SSHD. La transferencia de archivos (los necesarios para compilar el tp) se hizo con la utilidad SCP.
Para realizar la compilación de mips32 plot.S junto con el main.c y demás archivos necesarios se utilizó el makefile provisto por la cátedra. Para esto se emite el comando "make", obteniendo de esta manera el ejecutable.
El archivo de salida se puede enviar al sistema operativo HostOS mediante SCP y
así corroborar el resultado del mismo.

2. Casos de Pruebas

a. Makefile

Se pueden correr pruebas utilizando el makefile:

$ make test

gcc −g −Wall −o tp0 tp0.c

#TEST imagen por defecto. Se crea archivo en /tests/

./$(OUTPUT) -o $(TESTS_DIR)/uno.pgm

#TEST imagen 2.

./$(OUTPUT) -o $(TESTS_DIR)/dos.pgm -c 0.282-0.007i -w 0.005 -H 0.005

#TEST Generar imagen con nombre fractales.pgm en la test

./$(OUTPUT) -o $(TESTS_DIR)/fractales.pgm

#TEST imprimir por pantalla 0

./$(OUTPUT) -c 10+0i -r 1x1

#TEST complejo incorrecto

./$(OUTPUT) -s 0+1 -o $(TESTS_DIR)/complejo_incorrecto.pgm

3. Conclusión

Al final de este TP pudimos observar los usos de cada distinta herramienta. Vimos como gxemul nos posibilita levantar una imagen virtual de otro sistema operativo dentro de nuestra propia máquina. Esta imagen sea NetBSD que utilizamos para poder trabajar con otras arquitecturas que nuestro sistema operativo no permitía, como en particular assembly MIPS. De esta manera pudimos implementar en MIPS el cómputo de los fractales. Aprendimos a programar en muy bajo nivel (para esta arquitectura de mips32) utilizando las distintas variaciones del set de instrucciones RISC, para guardar y cargar en memoria, incluyendo el trabajo con números flotantes que tienen sus propios registros y hasta sus propias instrucciones, según sean doubles o singles. Utilizamos además las convenciones de la ABI para el llamado de funciones teniendo en cuenta los registros que deben ser guardados en el stackframe de cada función antes y después de llamar a otra y al ser llamada.