U.B.A. FACULTAD DE INGENIERÍA

Departamento de Electrónica

Organización de computadoras 66-20

TRABAJO PRÁCTICO #0

Infraestructura básica

Curso: 2018 - 2do Cuatrimestre

Turno: Martes

GRUPO N°	
Integrantes	**Padrón**
Verón, Lucas	89341
Gamarra Silva, Cynthia Marlene	92702
Gatti, Nicolás	93570
Fecha de entrega:	18-09-2018
Fecha de aprobación:	
Calificación:	
Firma de aprobación:	

Observaciones:

Índice

1. Enunciado del trabajo práctico

66:20 Organización de Computadoras
Trabajo práctico #0: Infraestructura básica
2^{do} cuatrimestre de 2018

$Date: 2018/09/08 23:16:30 $

1. Objetivos

Familiarizarse con las herramientas de software que usaremos en los siguientes trabajos, implementando un programa (y su correspondiente documentación) que resuelva el problema piloto que presentaremos más abajo.

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.

Además, es necesario que el trabajo práctico incluya (entre otras cosas, ver sección 6), la presentación de los resultados obtenidos explicando, cuando corresponda, con fundamentos reales, las causas o razones de cada resultado obtenido.

El informe deberá respetar el modelo de referencia que se encuentra en el grupo[1], y se valorarán aquellos escritos usando la herramienta TEX / LATEX.

4. Recursos

Usaremos el programa GXemul [1] para simular el entorno de desarrollo que utilizaremos en este y otros trabajos prácticos, una máquina MIPS corriendo una versión reciente del sistema operativo NetBSD [2].

En la clase del 21/8 hemos repasado los pasos necesarios para la instalación y configuración del entorno de desarrollo.

[1] `http://groups.yahoo.com/group/orga-comp`

5. Programa

Se trata de escribir, en lenguaje C, un programa para codificar y decodificar información en formato base 64: el programa recibirá, por línea de comando, los archivos o *streams* de entrada y salida, y la acción a realizar, codificar (acción por defecto) o decodificar. De no recibir los nombres de los archivos (o en caso de recibir `-` como nombre de archivo) usaremos los *streams* estándar, `stdin` y `stdout`, según corresponda. A continuación, iremos leyendo los datos de la entrada, generando la salida correspondiente. De ocurrir errores, usaremos `stderr`. Una vez agotados los datos de entrada, el programa debe finalizar adecuadamente, retornando al sistema operativo.

Estrictamente hablando, base 64 es un grupo de esquemas de codificación similares. En nuestra implementación, estaremos siguiendo particularmente el esquema establecido en [3], con el siguiente agregado: si se recibe una secuencia de caracteres inválida en la decodificación, debe asumirse como una condición de error que el programa deberá reportar adecuadamente y detener el procesamiento en ese punto.

5.1. 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.
  -a, --action      Program action: encode (default) or decode.
Examples:
  tp0 -a encode -i ~/input -o ~/output
  tp0 -a decode
```

Codificamos un archivo vacío (cantidad de bytes nula):

```
$ touch /tmp/zero.txt
$ tp0 -a encode -i /tmp/zero.txt -o /tmp/zero.txt.b64
$ ls -l /tmp/zero.txt.b64
-rw-r--r--  1 user group 0 2018-09-08 16:21 /tmp/zero.txt.b64
```

Codificamos el caracter ASCII `M`,

```
$ echo -n M | tp0
TQ==
```

Codificamos los caracteres ASCII `M` y `a`,

```
$ echo -n Ma | tp0
TWE=
```

Codificamos `M a n`,

```
$ echo -n Man | tp0
TWFu
```

Codificamos y decodificamos:

```
$ echo Man | tp0 | tp0 -a decode
Man
```

Verificamos bit a bit:

```
$ echo xyz | tp0 | tp0 -a decode | od -t c
0000000    x   y   z  \n
0000004
```

Codificamos 1024 bytes, para verificar que el programa genere líneas de no mas de 76 unidades de longitud:

```
$ yes | head -c 1024 | tp0 -a encode
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK
...
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5Cg==
```

Verificamos que la cantidad de bytes decodificados, sea 1024:

```
$ yes | head -c 1024 | tp0 -a encode | tp0 -a decode | wc -c
1024
```

Generamos archivos de tamaño creciente, y verificamos que el procesamiento de nuestro programa no altere los datos:

```
$ n=1;
$ while :; do
>         head -c $n </dev/urandom >/tmp/in.bin;
>         tp0 -a encode -i /tmp/in.bin  -o /tmp/out.b64;
>         tp0 -a decode -i /tmp/out.b64 -o /tmp/out.bin;
>         if diff /tmp/in.bin /tmp/out.bin; then :; else
>                 echo ERROR: $n;
>                 break;
>         fi
>         echo ok: $n;
>         n="`expr $n + 1`";
>         rm -f /tmp/in.bin /tmp/out.b64 /tmp/out.bin
> done
ok: 1
ok: 2
ok: 3
...
```

6. Informe

El informe deberá incluir al menos las siguientes secciones:

- Documentación relevante al diseño e implementación del programa;
- Comando(s) para compilar el programa;
- Las corridas de prueba, con los comentarios pertinentes;
- El código fuente, en lenguaje C, el cual también deberá entregarse en formato digital compilable (incluyendo archivos de entrada y salida de pruebas);
- El código MIPS32 generado por el compilador;
- Este enunciado.

El informe deberá entregarse en formato impreso y digital.

Referencias

[1] GXemul, `http://gavare.se/gxemul/`.

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

[3] RFC 2045: Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies; sección 6.8, Base64 Content-Transfer-Encoding. `http://tools.ietf.org/html/rfc2045#section-6.8`.

[4] Base64 (Wikipedia). `http://en.wikipedia.org/wiki/Base64`.

1.1. Diseño e implementación

El programa que contiene la lógica de codificador y decodificador se encuentra en el archivo ***encode.c***. El codificador transforma expresiones con caracteres ASCII en base64, mientras que el decodificador hace el proceso inverso. Ambas funciones se implementan con diversos ciclos y recorridos, así como utilización de instrucciones aritmético lógicas (sumas, &, shifts, etc) que relacionan la codificación base 64 con ASCII.

Se creó un achivo ***file.c*** para el manejo de los archivos a codificar/decodificar. Además se creó otro archivo llamado ***command.c*** cuya función es tomar los parámetros del programa y realizar las acciones pertinentes a la misma.

Específicamente el programa se estructura en los siguientes pasos:

- Análisis de las parámetros de la línea de comandos: se analizan las opciones ingresadas por la línea de comandos utilizando la función `getopt_long()`, la cual puede procesar cada opción que es leída de forma simplificada. Se extraen los argumentos de cada opción y se los guarda dentro de una estructura para su posterior acceso del tipo `CommandOptions` cuya definición es

```
1         typedef struct {
2                 File input;
3                 File output;
4                 const char* input_route;
5                 const char* output_route;
6                 char error;
7                 char encode_opt;
8         } CommandOptions;
9
```

  En caso de que no se encuentre alguna opción, se muestra el mensaje de ayuda al usuario para que identifique el prototipo de cómo debe ejecutar el programa.

- Validación de opciones: a medida que se va analizando cada opción de la línea de comandos, se valida cada una de ellas. Si se ingresó algún parámetro no válido para el programa o si se encuentró un error se lo informa al usuario por pantalla y se aborta la ejecución del programa. Se utiliza para ello se la función `CommandErrArg()` cuyo resultado es:

```
1                 fprintf(stderr, "Invalid Arguments\n");
2                 fprintf(stderr,"Options:\n");
3                 fprintf(stderr,"  -V, --version    Print version and quit.\n");
4                 fprintf(stderr,"  -h, --help       Print this information.\n");
5                 fprintf(stderr,"  -i, --input      Location of the input file.\n");
6                 fprintf(stderr,"  -o, --output     Location of the output file.\n");
7                 fprintf(stderr,"  -a, --action     Program action: encode (default) or
    decode.\n");
8                 fprintf(stderr,"Examples:\n");
9                 fprintf(stderr,"  tp0 -a encode -i ~/input -o ~/output\n");
10                fprintf(stderr,"  tp0 -a decode\n");
11
```

  Para el caso en que no hubo errores a la validación de los argumentos se procede a llamar a las funciones correspondientes a:

  - `Mensaje de ayuda:` Función `CommandVersion()`
  - `Mensaje de versión:` Función `CommandHelp()`
  - `Input file :` Función `CommandSetInput()` que guarda la entrada del archivo donde será leído el texto.
  - `Output file:` Función CommandSetOutput() que guarda la entrada del archivo de salida donde se escribirá el texto codificado.
  - `Acción del programa a ejecutar:` Función CommandSetEncodeOpt() que setea la variable $opt->encode\_opt$ indicando si es una operación de ENCODE o DECODE respectivamente.

- Encode/Decode: una vez que se procesó correctamente las opciones de la línea de comandos se procede a llamar a la función `_CommandEncodeDecode()` que ejecutará la operación de ENCODE o DECODE dependiendo del argumento pasado en la línea de comandos. La operación de DECODE se ejecuta el siguiente código:

```
while (!FileEofReached(&opt->input) && !CommandHasError(opt)) {
            unsigned int read = FileRead(&opt->input, buf_encoded, 4);
            if (read > 0) {   // Solo es 0 si alcance el EOF
                if (read != 4) {  //Siempre debo leer 4 sino el formato es incorrecto
                    fprintf(stderr, "Longitud de archivo no es multiplo de 4\n");
                    CommandSetError(opt);
                } else {
                    ++count;
                    if (count == 18) { // 19 * 4 = 76 bytes
                        unsigned char aux;
                        FileRead(&opt->input, &aux, 1);
                        count = 0;
                    }
                    if (Decode(buf_encoded, buf_decoded)) {
                        char aux = 0;
                        if (buf_encoded[2] == '=')
                            ++aux;
                        if (buf_encoded[3] == '=')
                            ++aux;

                        FileWrite(&opt->output, buf_decoded, 3 - aux);
                    } else {
                        fprintf(stderr, "Caracteres invalidos en archivo codificado: ");
                        unsigned int i;
                        for (i = 0; i < 4; ++i)
                            fprintf(stderr, "%c", buf_encoded[i]);
                        CommandSetError(opt);
                    }
                }
            }


        }

```

Básicamente lo que se realiza es la lectura del archivo para procesarlo teniendo en cuenta la longitud del archivo a procesar y el padding a decodificar. La función `Decode()` retorna un buffer de 3 caracteres con el decode de 4 caracteres en base64. Se debe cumplir:

* Pre: el buffer input contiene 4 caracteres. El buffer output tiene por lo menos 3 caracteres * Post: retorna un buffer de 3 byte con los caracteres en ASCII. retorna 0 si error 1 si ok.

La operación de ENCODE se ejecuta el siguiente código:

```
        while(!FileEofReached(&opt->input)){
            memset(buf_decoded, 0, 3);
            unsigned int read = FileRead(&opt->input, buf_decoded, 3);
            if (read > 0) {
                Encode(buf_decoded, read, buf_encoded);
                FileWrite(&opt->output, buf_encoded, 4);
                ++count;
                if (count == 18) { // 19 * 4 = 76 bytes
                    FileWrite(&opt->output, (unsigned char *) "\n", 1);
                    count = 0;
                }

            }
        }
```

Básicamente lo que se realiza es la lectura del archivo para procesarlo en la función `Encode()`e n donde recibe 3 caracteres en buffer y los convierte en 4 caracteres codificados en output. Se debe cumplir:

* Pre: el buffer contiene length caracteres (1 a 3) y todos los caracteres son validos * Post: retorna un buffer de 4 byte con los caracteres en base64.

1.2. Parámetros del programa

Se detallan a continuación los parámetros del programa

- -h: Visualiza la ayuda del programa, en la que se indican los parámetros y sus objetivos.
- -V: Indica la versión del programa.
- -i: Archivo de entrada del programa.
- -o: Archivo de salida del programa.
- -a: Acción a llevar a cabo: codificación o decodificación.

Se indica a continuación detalles respecto a los parámetros:

- Si no se explicitan -i y -o, se utilizarán stdin y stdout, respectivamente.
- -V es una opción "show and quit". Si se explicita este parámetro, sólo se imprimirá la versión, aunque el resto de los parámetros se hayan explicitado.
- -h también es de tipo "show and quit z se comporta de forma similar a -V.
- en caso de que se use la entrada estándar (con comando echo texto | ./tp0 -a encode) y luego se especifique un archivo de salida con -i, prevalecerá el establecido por parámetro.

1.3. Compilación del programa

Para obtener un ejecutable, se creó un archivo `makefile` cuyo contenido es:

```
CC = gcc
CFLAGS = -o0 -g -Wall -Werror -pedantic -std=c99

OBJECTS = command.o encode.o file.c
EXEC = tp0

VALGRIND = valgrind --track-origins=yes --leak-check=full
VALGRIND-V = $(VALGRIND) -v

all: $(EXEC)

command.o: command.c command.h
	$(CC) $(CFLAGS) -c command.c -o command.o
encode.o: encode.c encode.h
	$(CC) $(CFLAGS) -c encode.c -o encode.o
file.o: file.c file.h
	$(CC) $(CFLAGS) -c file.c -o file.o

$(EXEC): $(OBJECTS)
	$(CC) $(CFLAGS) $(OBJECTS) main.c -o $(EXEC) -lm

run: $(EXEC)
	./$(EXEC)

valgrind: $(EXEC)
	$(VALGRIND) ./$(EXEC)
```

```
27
28 valgrind-verb: $(EXEC)
29         $(VALGRIND-V) ./$(EXEC)
30
31 clean:
32         rm -f *.o $(EXEC)
33
```

Para ejecutarlo, posicionarse en el directorio `src/` y ejecutar el siguiente comando:

```
1 $ make
```

Para proceder a la ejecución del programa, se debe llamar a:

```
1 $ ./tp0
```

seguido de los parámetros que se desee modificar, los cuales se indicaron en la sección 1.2.

En caso de ser entrada estándar (stdin) se podrá ejecutar de la siguiente forma:

```
1 $ echo texto | ./tp0 -a encode
```

También en este caso, se indican a continuación los parámetros a usar.

Para el caso de hacerlo en el emulador GXemul que provee la cátedra, utilizando la máquina virtual que contiene el sistema operativo NetBSD, no se utilizó el archivo Makefile, la compilación se realizó con la herramienta `gcc`.

2. Pruebas realizadas

2.1. Pruebas con archivo bash test.sh

Se realizan corridas de prueba con el siguiente script

```
#!/bin/bash

n=1
while :; do
        head -c $n </dev/urandom >/tmp/in.bin;
        ./tp0 -a encode -i /tmp/in.bin -o /tmp/out.b64;
        ./tp0 -a decode -i /tmp/out.bin -o /tmp/out.bin;
        if diff /tmp/in.bin /tmp/out.bin; then :; else
                echo ERROR: $n;
                break;
        fi
        echo ok: $n;
        n="`expr $n + 1`";
        rm -f /tmp/in.bin /tmp/out.b64 /tmp/out.bin o;
done
```

El cual no presenta errores en ninguna de las corridas llevadas a cabo.

Todas las pruebas que se presentan a continuación, están codificadas en los archivos de prueba ***.txt de forma que puedan ejecutarse y comprobar los resultados obtenidos.

Se indicaran a continuación lo siguiente: comandos para ejecutarlas, líneas de código que las componen y resultado esperado.

2.1.1. Generales

- Mensaje de ayuda

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

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.
  -a, --action     Program action: encode (default) or decode.
Examples:
  tp0 -a encode -i ~/input -o ~/output
  tp0 -a decode
```

- Mensaje de version

```
$ ./tp0 -V  o ./tp0 --version
Version: 0.1

```

- Archivo de entrada no válido

```
$ ./tp0 -i archivoInvalido.txt

Invalid Arguments
Options:
  -V, --version    Print version and quit.
  -h, --help       Print this information.
  -i, --input      Location of the input file.
```

```
8    -o, --output      Location of the output file.
9    -a, --action      Program action: encode (default) or decode.
10 Examples:
11   tp0 -a encode -i ~/input -o ~/output
12   tp0 -a decode
13
14
15
```

Referencias

[1] Base64 (Wikipedia) http://en.wikipedia.org/wiki/Base64

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

[3] Kernighan, B. W. - Ritchie, D. M. - *C Programming Language* - 2nd edition - Prentice Hall - 1988.

[4] *GNU Make* - https://www.gnu.org/software/make/

[5] *Valgrind* - http://valgrind.org/

A. Código fuente

A.0.1. main.c

```
/**
 * Created by gatti2602 on 12/09/18.
 * Main
 */

#define ERROR 0
#define FALSE 0
#define TRUE 1

#include <getopt.h>
#include "command.h"

int main(int argc, char** argv) {
    struct option arg_long[] = {
            {"input",   required_argument,  NULL,  'i'},
            {"output",  required_argument,  NULL,  'o'},
            {"action",  required_argument,  NULL,  'a'},
            {"help",    no_argument,        NULL,  'h'},
            {"version", no_argument,        NULL,  'V'},
    };
    char arg_opt_str[] = "i:o:a:hV";
    int arg_opt;
    int arg_opt_idx = 0;
    char should_finish = FALSE;

    CommandOptions cmd_opt;
    CommandCreate(&cmd_opt);

    if(argc == 1)
        CommandSetError(&cmd_opt);

    while((arg_opt =
                   getopt_long(argc, argv, arg_opt_str, arg_long, &arg_opt_idx)) != -1 && !
    should_finish) {
        switch(arg_opt){
                case 'i':
                        CommandSetInput(&cmd_opt, optarg);
                        break;
                case 'o':
                CommandSetOutput(&cmd_opt, optarg);
                break;
                case 'h':
                        CommandHelp();
                should_finish = TRUE;
                break;
                case 'V':
                        CommandVersion();
                should_finish = TRUE;
                break;
                case 'a':
                    CommandSetEncodeOpt(&cmd_opt, optarg);
                                break;
                default:
                        CommandSetError(&cmd_opt);
                        break;
        }
    }

    if(should_finish)
        return 0;

    if(!CommandHasError(&cmd_opt)) {
```

```
62          CommandProcess(&cmd_opt);
63      } else {
64          CommandErrArg();
65          return 1;
66      }
67      return 0;
68  }
```

A.0.2. Assembly

```
        .file   1 "main.c"
        .section .mdebug.abi32
        .previous
        .abicalls
        .rdata
        .align  2
$LC0:
        .ascii  "input\000"
        .align  2
$LC1:
        .ascii  "output\000"
        .align  2
$LC2:
        .ascii  "action\000"
        .align  2
$LC3:
        .ascii  "help\000"
        .align  2
$LC4:
        .ascii  "version\000"
        .data
        .align  2
$LC5:
        .word   $LC0
        .word   1
        .word   0
        .word   105
        .word   $LC1
        .word   1
        .word   0
        .word   111
        .word   $LC2
        .word   1
        .word   0
        .word   97
        .word   $LC3
        .word   0
        .word   0
        .word   104
        .word   $LC4
        .word   0
        .word   0
        .word   86
        .globl  memcpy
        .rdata
        .align  2
$LC6:
        .ascii  "i:o:a:hV\000"
        .text
        .align  2
        .globl  main
        .ent    main
main:
        .frame  $fp,200,$ra             # vars= 152, regs= 3/0, args= 24, extra= 8
        .mask   0xd0000000,-8
        .fmask  0x00000000,0
        .set    noreorder
        .cpload $t9
        .set    reorder
        subu    $sp,$sp,200
        .cprestore 24
        sw      $ra,192($sp)
        sw      $fp,188($sp)
        sw      $gp,184($sp)
        move    $fp,$sp
```

```
66          sw      $a0,200($fp)
67          sw      $a1,204($fp)
68          addu    $v0,$fp,32
69          la      $v1,$LC5
70          move    $a0,$v0
71          move    $a1,$v1
72          li      $a2,80                  # 0x50
73          la      $t9,memcpy
74          jal     $ra,$t9
75          lw      $v0,$LC6
76          sw      $v0,112($fp)
77          lw      $v0,$LC6+4
78          sw      $v0,116($fp)
79          lbu     $v0,$LC6+8
80          sb      $v0,120($fp)
81          sw      $zero,132($fp)
82          sb      $zero,136($fp)
83          addu    $v0,$fp,144
84          move    $a0,$v0
85          la      $t9,CommandCreate
86          jal     $ra,$t9
87          lw      $v1,200($fp)
88          li      $v0,1                   # 0x1
89          bne     $v1,$v0,$L18
90          addu    $v0,$fp,144
91          move    $a0,$v0
92          la      $t9,CommandSetError
93          jal     $ra,$t9
94  $L18:
95          .set    noreorder
96          nop
97          .set    reorder
98  $L19:
99          addu    $v1,$fp,112
100         addu    $v0,$fp,132
101         sw      $v0,16($sp)
102         lw      $a0,200($fp)
103         lw      $a1,204($fp)
104         move    $a2,$v1
105         addu    $a3,$fp,32
106         la      $t9,getopt_long
107         jal     $ra,$t9
108         sw      $v0,128($fp)
109         lw      $v1,128($fp)
110         li      $v0,-1                  # 0xffffffffffffffff
111         beq     $v1,$v0,$L20
112         lb      $v0,136($fp)
113         bne     $v0,$zero,$L20
114         lw      $v0,128($fp)
115         addu    $v0,$v0,-86
116         sw      $v0,180($fp)
117         lw      $v1,180($fp)
118         sltu    $v0,$v1,26
119         beq     $v0,$zero,$L29
120         lw      $v0,180($fp)
121         sll     $v1,$v0,2
122         la      $v0,$L30
123         addu    $v0,$v1,$v0
124         lw      $v0,0($v0)
125         .cpadd  $v0
126         j       $v0
127         .rdata
128         .align  2
129 $L30:
130         .gpword $L27
131         .gpword $L29
132         .gpword $L29
```

```
133         .gpword $L29
134         .gpword $L29
135         .gpword $L29
136         .gpword $L29
137         .gpword $L29
138         .gpword $L29
139         .gpword $L29
140         .gpword $L29
141         .gpword $L28
142         .gpword $L29
143         .gpword $L29
144         .gpword $L29
145         .gpword $L29
146         .gpword $L29
147         .gpword $L29
148         .gpword $L26
149         .gpword $L24
150         .gpword $L29
151         .gpword $L29
152         .gpword $L29
153         .gpword $L29
154         .gpword $L29
155         .gpword $L25
156         .text
157 $L24:
158         addu    $v0,$fp,144
159         move    $a0,$v0
160         lw      $a1,optarg
161         la      $t9,CommandSetInput
162         jal     $ra,$t9
163         b       $L19
164 $L25:
165         addu    $v0,$fp,144
166         move    $a0,$v0
167         lw      $a1,optarg
168         la      $t9,CommandSetOutput
169         jal     $ra,$t9
170         b       $L19
171 $L26:
172         la      $t9,CommandHelp
173         jal     $ra,$t9
174         li      $v0,1                   # 0x1
175         sb      $v0,136($fp)
176         b       $L19
177 $L27:
178         la      $t9,CommandVersion
179         jal     $ra,$t9
180         li      $v0,1                   # 0x1
181         sb      $v0,136($fp)
182         b       $L19
183 $L28:
184         addu    $v0,$fp,144
185         move    $a0,$v0
186         lw      $a1,optarg
187         la      $t9,CommandSetEncodeOpt
188         jal     $ra,$t9
189         b       $L19
190 $L29:
191         addu    $v0,$fp,144
192         move    $a0,$v0
193         la      $t9,CommandSetError
194         jal     $ra,$t9
195         b       $L19
196 $L20:
197         lb      $v0,136($fp)
198         beq     $v0,$zero,$L31
199         sw      $zero,176($fp)
```

```
200         b       $L17
201 $L31:
202         addu    $v0,$fp,144
203         move    $a0,$v0
204         la      $t9,CommandHasError
205         jal     $ra,$t9
206         bne     $v0,$zero,$L32
207         addu    $v0,$fp,144
208         move    $a0,$v0
209         la      $t9,CommandProcess
210         jal     $ra,$t9
211         b       $L33
212 $L32:
213         la      $t9,CommandErrArg
214         jal     $ra,$t9
215         li      $v0,1                   # 0x1
216         sw      $v0,176($fp)
217         b       $L17
218 $L33:
219         sw      $zero,176($fp)
220 $L17:
221         lw      $v0,176($fp)
222         move    $sp,$fp
223         lw      $ra,192($sp)
224         lw      $fp,188($sp)
225         addu    $sp,$sp,200
226         j       $ra
227         .end    main
228         .size   main, .-main
229         .ident  "GCC: (GNU) 3.3.3 (NetBSD nb3 20040520)"
```