Trabajo Práctico 1:
Conjunto de instrucciones MIPS

Jimenez, Ruben, *Padrón Nro. 92.402*
rbnm.jimenez@gmail.com

Reyero, Felix, *Padrón Nro. 92.979*
felixcarp@gmail.com

Suárez, Emiliano, *Padrón Nro. 78.372*
emilianosuarez@gmail.com

Primera Entrega: *30/04/2015*

1er. Cuatrimestre de 2015
66.20 Organización de Computadoras – Práctica Jueves
Facultad de Ingeniería, Universidad de Buenos Aires

Resumen

Se implementó una versión simplificada del programa **sha1** de UNIX. Para nuestra implementación.

1. Introducción

Este Trabajo Práctico pretende familiarizarse con la programación en assembly y el concepto de ABI.

Para ello, implementaremos el algoritmo **sha1** de UNIX en código *Assembly*, mientras que la interpretación de argumentos del programa y la lectura de archivos será realizada en lenguaje *C*.

Además, se utilizará *GXemul* para simular una máquina *MIPS* corriendo una versión reciente del sistema operativo *NetBSD*.

El programa implementado, muestra por *stdout* un checksum generado a partir del contenido de los archivos archivos pasados por parámetro. En caso de no especificarse algún archivo, se mostrarán un checksum a partir de lo ingresado por *stdin*.

2. Diseño e Implementación

Se implementó un programa que realiza la lectura a través del stdin o a través de archivos que se reciben por parámetro.

El comando acepta 2 parámetros para mostrar la Ayuda y la Versión del programa:

```
$ ./sha -h
$ ./sha --help
```

Para desplegar la ayuda del comando. Y los siguientes comandos para mostrar la versión:

```
$ ./sha -V
$ ./sha --version
```

Inicialmente el programa revisa la cadena de parametros ingresada y determina si el checksum debe generarse a partir de lo ingresado por *stdin* o a través del contenido del (o los) archivo(s).

Para este última opción, se procesan los archivo de uno por vez, y para cada uno de ellos se genera el checksum a partir de sus datos.

2.1. Compilación

Para compilar el programa se debe abrir una terminal en la carpeta donde están alojados los archivos fuentes (*src/*) y se ejecuta el siguiente comando:

```
../src$ gcc -g -O0 -Wall main.c algoritmo.S trozo.S relleno.S -o sha
../src$ gcc -g -O0 -Wall main.c algoritmo.S trozo.S relleno.S -o sha # borrar esta linea
```

Para generar el ejecutable *sha*.

2.2. Arquitectura

El trabajo está dividido en los siguientes archivos:

main.c
: Cuerpo principal del programa, donde se realiza la lectura de los parámetros de los archivos.

relleno.S
: Donde se encuentra la implementación en *Assembly* de la función *calcularRelleno*.

trozo.S
: Donde se encuentra la implementación en *Assembly* de la función *cargarTrozos*.

algoritmo.S
: Donde se encuentra la implementación en *Assembly* de la función *algoritmoSha1*.

Las funciones *calcularRelleno*, *cargarTrozos* y *algoritmoSha1*, son llamadas desde la función *sha1* de *main.c*.

Los stacks de cada una de ellas que pueden observase a continuación:

calcularRelleno	
Dir Mem	Valor
44	
40	ra
36	fp
32	gp
28	longOrig
24	longOrig
20	longRelleno
16	longRelleno
12	a3
8	a2
4	a1
0	a0

cargarTrozos	
Dir Mem	Valor
36	
32	ra
28	fp
24	gp
20	mascara
16	i
12	a3
8	a2
4	a1
0	a0

algoritmoSha1	
Dir Mem	Valor
44	
40	ra
36	fp
32	gp
28	
24	temp
20	k
16	i
12	a3
8	a2
4	a1
0	a0

Registro	Valor
s0	bloques
s1	trozos
s2	a
s3	b
s4	c
s5	d
s6	e
s7	f

3. Casos de Prueba

Algunos de los casos de pruebas realizados, pueden observarse a continuación:

```
root@:/home/gxemul/tprub/div# sha1 hola
SHA1 (hola) = ef443fee4da6bfb41651930de7ad99f29ed9f079
root@:/home/gxemul/tprub/div# ./sha hola
ef443fee4da6bfb41651930de7ad99f29ed9f079

root@:/home/gxemul/tprub/div# sha1 vacio
SHA1 (vacio) = da39a3ee5e6b4b0d3255bfef95601890afd80709
root@:/home/gxemul/tprub/div# ./sha vacio
da39a3ee5e6b4b0d3255bfef95601890afd80709

root@:/home/gxemul/tprub/div# sha1 prueba-dog
SHA1 (prueba-dog) = 2fd4e1c67a2d28fced849ee1bb76e7391b93eb12
root@:/home/gxemul/tprub/div# ./sha prueba-dog
2fd4e1c67a2d28fced849ee1bb76e7391b93eb12

root@:/home/gxemul/tprub/div# sha1 prueba-cog
SHA1 (prueba-cog) = de9f2c7fd25e1b3afad3e85a0bd17d9b100db4b3
root@:/home/gxemul/tprub/div# ./sha prueba-cog
de9f2c7fd25e1b3afad3e85a0bd17d9b100db4b3
```

4. Conclusiones

El presente trabajo permitió la familiarización con las herramientas de compilación de código C y código assembly en un entorno que emula la arquitectura MIPS 32, asegurando la portabilidad del programa.

5. Apéndice

5.1. Código Fuente: main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define ERROR_OPEN_FILE 10
#define MAX_COUNT_FILES 50
#define HASH_LENGTH 20

void printHelp();
void printVersion();
void printError(char* msgError, int codeError);
char* setFileSize(FILE* fp, long long int *length);

long long int calcularRelleno(long long int longitudOriginal);
void cargarTrozos(char *bloque,unsigned int *Trozos);
void algoritmoSha1(unsigned int *Trozos,unsigned *a,unsigned *b,
    unsigned *c,unsigned *d,unsigned *e);

int sha1(unsigned char *resultado, char *nombre_archivo, unsigned
    long long longitudOriginal);
unsigned int leftrotate(unsigned int valor,int desplazamiento);
void asignarDatos(char *file,unsigned char *bloques,long long int
    tamanioOriginal, long long int longitudRelleno);

int main(int argc, char* argv[]) {
    int i = 0;
    char* param;
    char* files[MAX_COUNT_FILES];
    unsigned char *result;
    int fileCount = 0;
    FILE* fp;
    char *start;
    long long int length;

    // Parse arguments
    if (argc >= 2) {
        param = *(argv + 1);
        if ((strcmp(param, "-h") == 0) || (strcmp(param, "--help")
            == 0) ) {
            printHelp();
        }
        else if ((strcmp(param, "-V") == 0) || (strcmp(param, "--
            version") == 0)) {
            printVersion();
        }
    }

    // Search for files
    for(i = 1; i < argc; i++) {
        if (*argv[i] != '-') {
            files[fileCount++] = argv[i];
        }
    }

    // Process each file
    for(i = 0; i < fileCount; i++) {
        result = malloc(HASH_LENGTH);

```

```
54          if (NULL == (fp = fopen(files[i],"r"))) {
55              fprintf(stderr, "Files '%s' doesn't exist.", files[i]);
56              exit(ERROR_OPEN_FILE);
57          }
58
59          start = setFileSize(fp, &length);
60          long long int longitudRelleno = calcularRelleno(length);
61          char *bloques = malloc(longitudRelleno/8);
62          asignarDatos(files[i],bloques,length,longitudRelleno); //
                se almacena el tamanioOrginal al final
63          sha1(result,bloques, length);
64      }
65
66      for(i = 0; i < 20; i++) {
67          unsigned char aux = result[i];
68          aux<<=4;
69          aux>>=4;
70          printf("%x",(result[i]>>4));
71          printf("%x",aux);
72      }
73      printf("\n");
74
75      return 0;
76  }
77
78  char* setFileSize(FILE* fp, long long int *length) {
79      int character;
80      char *start;
81      int n = 0;
82
83      fseek(fp, 0, SEEK_END);
84      *length = ftell(fp)*8;  //devuelve el tamanio en bits
85
86      fseek(fp, 0, SEEK_SET);
87
88      start = malloc(1);
89      start = realloc(start, (*length) );
90
91      while ((character = fgetc(fp)) != EOF) {
92          start[n++] = (char)character;
93      }
94      start[n] = '\0';
95
96      return start;
97  }
98
99  void printHelp()
100 {
101     fprintf(stdout, "$ tp1 -h\n");
102     fprintf(stdout, "Usage:\n");
103     fprintf(stdout, "  tp1 -h\n");
104     fprintf(stdout, "  tp1 -V\n");
105     fprintf(stdout, "  tp1 [file...]\n");
106     fprintf(stdout, "Options:\n");
107     fprintf(stdout, "  -V, --version    Print version and quit.\n")
            ;
108     fprintf(stdout, "  -h, --help       Print this information and
            quit.\n\n");
109     fprintf(stdout, "Examples:\n");
110     fprintf(stdout, "  tp1 foo\n");
111     fprintf(stdout, "  echo \"hello\" | tp1\n\n");
112 }
```

```
113
114 int sha1(unsigned char *resultado, char *bloques, unsigned long
        long longitudOriginal)
115 {
116     long long int longitudRelleno = calcularRelleno(
            longitudOriginal);
117     //prepocesamiento
118     long long int cantBloques = longitudRelleno/512; //longArchivo/
            tamanio bloque    bloque = 512bits
119
120     ////procesar el bloque en 4 rondas de 20 pasos cada ronda
121     //la memoria temporal cuenta con 5 regstros ABCDE
122     unsigned  A=0x67452301;
123     unsigned  B=0xEFCDAB89;
124     unsigned  C=0x98BADCFE;
125     unsigned  D=0x10325476;
126     unsigned  E=0xC3D2E1F0;
127
128     //unsigned int trozos[80]; //big endian
129     unsigned int *trozos = malloc(80*sizeof(int));
130
131     // int indice = 0;
132     int i;
133     unsigned a = 0;
134     unsigned b = 0;
135     unsigned c = 0;
136     unsigned d = 0;
137     unsigned e = 0;
138
139     while(cantBloques--)
140     {
141         cargarTrozos(bloques, trozos);
142
143         a = A;
144         b = B;
145         c = C;
146         d = D;
147         e = E;
148
149         algoritmoSha1(trozos,&a,&b,&c,&d,&e);
150
151         A += a;
152         B += b;
153         C += c;
154         D += d;
155         E += e;
156
157     }
158
159     // hh = (h0 leftshift 128) or (h1 leftshift 96) or (h2
            leftshift 64) or (h3 leftshift 32) or h4
160     for(i = 0;i<4;i++)
161     {
162         resultado[i]    = (A>>(24-8*i));
163         resultado[i+4]  = (B>>(24-8*i));
164         resultado[i+8]  = (C>>(24-8*i));
165         resultado[i+12] = (D>>(24-8*i));
166         resultado[i+16] = (E>>(24-8*i));
167     }
168
169     return 0;
170 }
```

```
171
172
173 //un bloque tiene 64 bytes
174 void asignarDatos(char *file,unsigned char *bloques,long long int
        tamanioOriginal, long long int longitudRelleno)
175 {
176     FILE *fp = fopen(file,"r");
177     char caracter;
178     int indice =0;
179     int i =0;
180     int cantBitsRelleno = 0;
181     char relleno = 0x80; //minimo relleno
182     char rellenoCero = 0x00;
183     unsigned mascara = 0x00000000000000FF;
184
185     while( (caracter = getc(fp)) != EOF)
186     {
187             *(bloques+indice) = caracter;
188             indice++;
189     }
190
191     cantBitsRelleno = (longitudRelleno - tamanioOriginal);
192     cantBitsRelleno = cantBitsRelleno - 64 - 8;//restamos 64 bits
            de tamanio y los 8 bits basicos de relleno;
193     *(bloques+indice) = relleno;
194     indice++;
195
196     for(i = 0;i<(cantBitsRelleno/8);i++)
197     {
198         *(bloques+indice) = rellenoCero;
199         indice++;
200     }
201
202     for(i = 0;i<8;i++)
203     {
204         *(bloques+indice) = (tamanioOriginal>>(56-8*i)) & mascara;
205         indice++;
206     }
207
208 }
209
210 //devuleve el tamanio en bits
211 long long int calcularTamanioArchivo(char* nombreFile)
212 {
213     long long int tamanio =0;
214     FILE *fp = fopen(nombreFile,"r");
215     fseek(fp,0,SEEK_END);
216     tamanio = ftell(fp)*8;
217     return tamanio;
218 }
219
220
221 unsigned int leftrotate(unsigned int valor,int desplazamiento)
222 {
223     desplazamiento %=32;
224     unsigned retorno;
225     retorno = (valor << desplazamiento) |  (valor >> (32 -
            desplazamiento));
226     return retorno;
227 }
228
229 void printVersion()
```

```
230 {
231     fprintf(stdout, "Copyright (c) 2015\n");
232     fprintf(stdout, "Conjunto de instrucciones MIPS. v1.0.0\n\n");
233 }
234
235 void printError(char* msgError, int codeError)
236 {
237     fprintf(stderr, "%s\n", msgError);
238     exit(codeError);
239 }
```

../src/main.c