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
```

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);

int readFromStdInput(int argumentCount);

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(FILE* fp,unsigned char *bloques,long long int
    tamanioOriginal, long long int longitudRelleno);
void showCheckSum(unsigned char *result);

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;

    if (readFromStdInput(argc)) {
        int c;

        fp = fopen("archivoAuxiliar.txt", "w+");
        while ((c = fgetc(stdin)) != EOF) {
            fputc(c, fp);
        };

        start = setFileSize(fp, &length);
        long long int longitudRelleno = calcularRelleno(length);
        char *bloques = malloc(longitudRelleno/8);
        asignarDatos(fp,bloques,length,longitudRelleno); // se
            almacena el tamanioOrginal al final
        sha1(result,bloques, length);
        showCheckSum(result);

        fclose (fp);
        remove("archivoAuxiliar.txt");

        return 0;

```

```
55      } else if (argc >= 2) {     // Parse arguments
56          param = *(argv + 1);
57          if ((strcmp(param, "-h") == 0) || (strcmp(param, "--help")
                == 0) ) {
58              printHelp();
59          }
60          else if ((strcmp(param, "-V") == 0) || (strcmp(param, "--
                version") == 0)) {
61              printVersion();
62          }
63      }
64
65      // Search for files
66      for(i = 1; i < argc; i++) {
67          if (*argv[i] != '-') {
68              files[fileCount++] = argv[i];
69          }
70      }
71
72      // Process each file
73      for(i = 0; i < fileCount; i++) {
74          result = malloc(HASH_LENGTH);
75
76          if (NULL == (fp = fopen(files[i],"r"))) {
77              fprintf(stderr, "Files '%s' doesn't exist.", files[i]);
78              exit(ERROR_OPEN_FILE);
79          }
80
81          start = setFileSize(fp, &length);
82          long long int longitudRelleno = calcularRelleno(length);
83          char *bloques = malloc(longitudRelleno/8);
84
85          fp = fopen(files[i],"r");
86          asignarDatos(fp,bloques,length,longitudRelleno); // se
                almacena el tamanioOrginal al final
87          sha1(result,bloques, length);
88      }
89
90      showCheckSum(result);
91
92      return 0;
93  }
94
95  char* setFileSize(FILE* fp, long long int *length) {
96      int character;
97      char *start;
98      int n = 0;
99
100     fseek(fp, 0, SEEK_END);
101     *length = ftell(fp)*8;  //devuelve el tamanio en bits
102
103     fseek(fp, 0, SEEK_SET);
104
105     start = malloc(1);
106     start = realloc(start, (*length) );
107
108     while ((character = fgetc(fp)) != EOF) {
109         start[n++] = (char)character;
110     }
111     start[n] = '\0';
112
113     return start;
```

```
114 }
115
116 void printHelp()
117 {
118     fprintf(stdout, "$ tp1 -h\n");
119     fprintf(stdout, "Usage:\n");
120     fprintf(stdout, "  tp1 -h\n");
121     fprintf(stdout, "  tp1 -V\n");
122     fprintf(stdout, "  tp1 [file...]\n");
123     fprintf(stdout, "Options:\n");
124     fprintf(stdout, "  -V, --version    Print version and quit.\n")
        ;
125     fprintf(stdout, "  -h, --help       Print this information and
        quit.\n\n");
126     fprintf(stdout, "Examples:\n");
127     fprintf(stdout, "  tp1 foo\n");
128     fprintf(stdout, "  echo \"hello\" | tp1\n\n");
129 }
130
131 int sha1(unsigned char *resultado, char *bloques, unsigned long
    long longitudOriginal)
132 {
133     long long int longitudRelleno = calcularRelleno(
        longitudOriginal);
134     //prepocesamiento
135     long long int cantBloques = longitudRelleno/512; //longArchivo/
        tamanio bloque   bloque = 512bits
136
137     ////procesar el bloque en 4 rondas de 20 pasos cada ronda
138     //la memoria temporal cuenta con 5 regstros ABCDE
139     unsigned  A=0x67452301;
140     unsigned  B=0xEFCDAB89;
141     unsigned  C=0x98BADCFE;
142     unsigned  D=0x10325476;
143     unsigned  E=0xC3D2E1F0;
144
145     //unsigned int trozos[80]; //big endian
146     unsigned int *trozos = malloc(80*sizeof(int));
147
148     // int indice = 0;
149     int i;
150     unsigned a = 0;
151     unsigned b = 0;
152     unsigned c = 0;
153     unsigned d = 0;
154     unsigned e = 0;
155
156     while(cantBloques--)
157     {
158         cargarTrozos(bloques, trozos);
159
160         a = A;
161         b = B;
162         c = C;
163         d = D;
164         e = E;
165
166         algoritmoSha1(trozos,&a,&b,&c,&d,&e);
167
168         A += a;
169         B += b;
170         C += c;
```

```
171         D += d;
172         E += e;
173 
174     }
175 
176     // hh = (h0 leftshift 128) or (h1 leftshift 96) or (h2
        leftshift 64) or (h3 leftshift 32) or h4
177     for(i = 0;i<4;i++)
178     {
179         resultado[i]    = (A>>(24-8*i));
180         resultado[i+4]  = (B>>(24-8*i));
181         resultado[i+8]  = (C>>(24-8*i));
182         resultado[i+12] = (D>>(24-8*i));
183         resultado[i+16] = (E>>(24-8*i));
184     }
185 
186     return 0;
187 }
188 
189 
190 //un bloque tiene 64 bytes
191 void asignarDatos(FILE *fp, unsigned char *bloques,long long int
        tamanioOriginal, long long int longitudRelleno)
192 {
193     // FILE *fp = fopen(file,"r");
194 
195     char caracter;
196     int indice =0;
197     int i =0;
198     int cantBitsRelleno = 0;
199     char relleno = 0x80; //minimo relleno
200     char rellenoCero = 0x00;
201     unsigned mascara = 0x00000000000000FF;
202 
203     while( (caracter = getc(fp)) != EOF)
204     {
205             *(bloques+indice) = caracter;
206             indice++;
207     }
208 
209     cantBitsRelleno = (longitudRelleno - tamanioOriginal);
210     cantBitsRelleno = cantBitsRelleno - 64 - 8;//restamos 64 bits
        de tamanio y los 8 bits basicos de relleno;
211     *(bloques+indice) = relleno;
212     indice++;
213 
214     for(i = 0;i<(cantBitsRelleno/8);i++)
215     {
216         *(bloques+indice) = rellenoCero;
217         indice++;
218     }
219 
220     for(i = 0;i<8;i++)
221     {
222         *(bloques+indice) = (tamanioOriginal>>(56-8*i)) & mascara;
223         indice++;
224     }
225 
226 }
227 
228 //devuleve el tamanio en bits
229 long long int calcularTamanioArchivo(char* nombreFile)
```

```
230 {
231     long long int tamanio =0;
232     FILE *fp = fopen(nombreFile,"r");
233     fseek(fp,0,SEEK_END);
234     tamanio = ftell(fp)*8;
235     return tamanio;
236 }
237
238
239 unsigned int leftrotate(unsigned int valor,int desplazamiento)
240 {
241     desplazamiento %=32;
242     unsigned retorno;
243     retorno = (valor << desplazamiento) |  (valor >> (32 -
            desplazamiento));
244     return retorno;
245 }
246
247 void printVersion()
248 {
249     fprintf(stdout, "Copyright (c) 2015\n");
250     fprintf(stdout, "Conjunto de instrucciones MIPS. v1.0.0\n\n");
251 }
252
253 void printError(char* msgError, int codeError)
254 {
255     fprintf(stderr, "%s\n", msgError);
256     exit(codeError);
257 }
258
259 void showCheckSum(unsigned char *result) {
260     int i;
261     for(i = 0; i < 20; i++) {
262         unsigned char aux = result[i];
263         aux<<=4;
264         aux>>=4;
265         printf("%x",(result[i]>>4));
266         printf("%x",aux);
267     }
268     printf("\n");
269 }
270
271 int readFromStdInput(int argumentCount) {
272     return (int)(argumentCount < 2);
273 }
```

../src/main.c