[66.20] ORGANIZACIÓN DE COMPUTADORAS

TRABAJO PRÁCTICO 0

2DO CUATRIMESTRE 2018

Infraestructura básica

AUTORES

Álvarez, Natalia Nayla. - #xx.xxx
<xxxx@xx.xx>
Husain, Ignacio Santiago. - #90.117
<santiago.husain@gmail.com>
Verstraeten, Federico. - #92.892
<federico.verstraeten@gmail.com>

CÁTEDRA

Dr. Ing. Hamkalo, José Luis.

CURSO

Ing. Santi, Leandro.

FECHA DE ENTREGA xx/xx/2018

FECHA DE APROBACIÓN

CALIFICACIÓN

FIRMA DE APROBACIÓN

Índice

1. Enunciado del trabajo práctico

66:20 Organización de Computadoras
Trabajo práctico #0: Infraestructura básica
1^{er} cuatrimestre de 2017

`$Date: 2017/03/21 22:54:33 $`

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 7/3 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 2017-03-19 15:14 /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
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK
...
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5Cg==
```

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.

7. Fechas

- Entrega: 28/3/2017;
- Vencimiento: 10/4/2017.

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`.

2. Objetivos

3. Diseño e implementación del programa

Para código en texto usar `CODE_1`. Para código en bloque usar

```
typedef struct myType_t {
int a, b;
float a1, a2;
FILE *outputStream;
} myType_t;
```

El código fuente se encuentra en el apéndice, tanto en lenguaje C (A.0.1) como en MIPS32 (A.0.4).

Para poner figuras, usar y se las cita con 3.1.

Figura 3.1: fig1.

4. Compilación del programa

Debido al requerimiento de utilizar el programa en una computadora con arquitectura MIPS32, se utiliza el emulador GXemul que provee la cátedra, utilizando una máquina virtual que contiene el sistema operativo NetBSD con las herramientas necesarias para compilar el programa desarrollado; `gcc` y `make`.

Para obtener un ejecutable, se creó un archivo `makefile` cuyo contenido se puede ver en la sección A.0.2. Para ejecutarlo, posicionarse en el directorio `src/` y ejecutar el siguiente comando:

```
$ make
```

El mismo generará el código assembler en el archivo “main.s” y el ejecutable con nombre `tp0`. En caso de requerir que el código assembly contenga los nombres de los registros, ejecutar el siguiente comando:

```
$ make ARGS=-mrnames
```

5. Pruebas

Se realizaron pruebas de caja negra para determinar si el programa funciona como se espera.

5.1. Ejecución con parámetros por defecto

5.2. Pruebas de errores en las opciones

Se creó un script en `Bash` para automatizar las pruebas del programa. En el mismo se prueban diferentes combinaciones de las opciones de entrada para verificar si el programa es capaz de detectar errores. El código del script se encuentra en la sección A.0.3, y está compuesto por X tests. Los Y primeros son validaciones utilizando opciones y parámetros inválidos, donde se verifica que al intentar ejecutarlo, el programa termine y retorne un mensaje que indique el motivo de la ejecución fallida. El último test se corresponde con ejecuciones que retornan un código de éxito. Dado que el programa en este caso solo retorna un solo código, el script devuelve un mensaje de éxito, indicando la correcta ejecución del programa y que se generó correctamente el archivo de salida.

La salida del script se divide en 7 secciones, cada una con un encabezado indicando el inicio del nuevo test, y 3 o más líneas por cada test. La primer línea del test es el comando ejecutado. La segunda indica si el test fue exitoso o no mediante la etiqueta `PASSED/NOT EQUAL`, y las siguientes líneas son los resultados que produce el programa (mensajes de error, etc...). Por ejemplo, para la prueba de la opción "-file", se tiene lo siguiente:

```
1  ./tp0 -o .
2 PASSED :
3 ERROR: Can't open output stream.
4 ERROR: Program exited with errors. = ERROR: Can't open output stream.
5 ERROR: Program exited with errors.
```

donde se ve que el test fue satisfactorio.

5.2.1. Salida de los tests automatizados

A continuación se muestran las salidas de los tests automatizados, concluyendo que se verifica el correcto funcionamiento del programa.

```
1 ######################################
2 TEST VALIDACION PARAMETRO 'FILE'
3 ######################################
4   ./tp0 -o .
5 PASSED :
6 ERROR: Can't open output stream.
7 ERROR: Program exited with errors. = ERROR: Can't open output↩
      stream.
8 ERROR: Program exited with errors.
```

5.3. Prueba de fuga de memoria

El último tipo de prueba es el de la verificación de fugas de memoria. Debido al uso de memoria dinámica en el programa, se utilizó la herramienta `valgrind` [3] para verificar la existencia de dichas fugas. El comando utilizado fue

```
valgrind --tool=memcheck --leak-check=full --show-leak-kinds=all -v ./tp0 ARGUMENTOS
```

y el resultado del mismo fue

```
==21530==
==21530== HEAP SUMMARY:
==21530==     in use at exit: 0 bytes in 0 blocks
==21530==   total heap usage: 3 allocs, 3 frees, 3,691,048 bytes allocated
==21530==
==21530== All heap blocks were freed -- no leaks are possible
==21530==
==21530== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==21530== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

concluyendo que no existen fugas de memoria.

Es importante aclarar que la verificación se realizó en la máquina `host` utilizando un procesador Intel i3-6100 (x86) y el sistema operativo Ubuntu 16.04.4. Si bien `valgrind` soporta la arquitectura MIPS32, en la máquina virtual no se dispone de esta herramienta, con lo cual no fue posible realizar la verificación de fuga de memoria en dicha máquina virtual.

6. Conclusiones

Referencias

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

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

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

[4] *Apuntes del curso 66.20 Organización de Computadoras* - Cátedra Hamkalo - Facultad de Ingeniería de la Universidad de Buenos Aires.

A. Código fuente

A.0.1. main.c

```
/* ------------------------------------------------------------
 @Title:   FIUBA - 66.20 Organización de Computadoras.
 @Project: TP0 - Infraestructura básica.
------------------------------------------------------------
 @Filename: main.c
------------------------------------------------------------
 @Authors:
    Álvarez, Natalia Nayla.
          xxx at xxx dot xxx
    Husain, Ignacio Santiago.
          santiago.husain at gmail dot com
    Verstraeten, Federico.
          federico.verstraeten at gmail dot com

 @Date:               07-Sep-2018 12:55:41 pm
 @Last modified by:   santiago
 @Last modified time: 07-Sep-2018 12:57:03 pm

 @Copyright(C):
    This file is part of 'TP0 - Infraestructura básica.'.
    Unauthorized copying or use of this file via any medium
    is strictly prohibited.
------------------------------------------------------------

Check memory leaks with the following command:
valgrind --tool=memcheck --leak-check=full \
--show-leak-kinds=all -v ./tp0

------------------------------------------------------------ */
#include <getopt.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char **argv) {

  return EXIT_SUCCESS;
}
```

A.0.2. makefile

```
# ------------------------------------------------------------
# @Title:   FIUBA - 66.20 Organización de Computadoras.
# @Project: TP0 - Infraestructura básica.
# ------------------------------------------------------------
# @Filename: makefile
# ------------------------------------------------------------
# @Authors:
#    Álvarez, Natalia Nayla.
#          xxx at xxx dot xxx
#    Husain, Ignacio Santiago.
#          santiago.husain at gmail dot com
#    Verstraeten, Federico.
#          federico.verstraeten at gmail dot com
#
# @Date:               07-Sep-2018 12:57:50 pm
# @Last modified by:   santiago
# @Last modified time: 07-Sep-2018 1:54:59 pm
#
# @Copyright (C):
#    This file is part of 'TP0 - Infraestructura básica.'.
#    Unauthorized copying or use of this file via any medium
#    is strictly prohibited.
# ------------------------------------------------------------
#
# The source files must have .c extension.
# The object code must have .o extension.
# The header files must have .h extension.
#
# To compile, execute 'make'.
# To clean all the compilation files, issue the command
# 'make clean'.
#
# ------------------------------------------------------------

# List all the header and object files separated by a blank
# space.
_DEPS =
_SRC = main.c
_OBJ = main.o
# ------------------------------------------------------------
# Configuration.
CC = gcc
CFLAGS = -Wall -I. -O0
OUTPUT1 = tp0
```

```
45 # ------------------------------------------------------------
46 # Processing.
47 $(OUTPUT1): $(_OBJ)
48     $(CC) $(CFLAGS) -o $(OUTPUT1) $(_OBJ)
49
50 assembly:
51     $(CC) $(CFLAGS) -S $(_SRC) $(ARGS)
52
53 .PHONY: clean
54
55 clean:
56     rm -f ./*.o *~ core ./*~ ./*.s
57     rm -f $(OUTPUT1)
```

A.0.3. run_tests.sh

```
#!/bin/bash

RED="\e[31m"
GREEN="\e[32m"
CYAN="\e[96m"
YELLOW="\e[93m"
BOLD="\033[1m"
DEFAULT="\e[0m"
EXEC='./tp0'


function header() {
  echo -e "$CYAN########################################$DEFAULT"
  echo -e "$CYAN$1$DEFAULT"
  echo -e "$CYAN########################################$DEFAULT"
}

function msg_true () {
  echo -e "$GREEN\0PASSED $DEFAULT:\n$1 = $GREEN $2 $DEFAULT"
}

function msg_false () {
  echo -e "$RED\0NOT EQUAL $DEFAULT:\n$1 = $YELLOW $2 $DEFAULT"
  echo -e "EXPECTED:\n$1"
}

function msg() {
  echo -e "  $BOLD$1$DEFAULT"
}

function success_msg() {
  echo -e "  $GREEN$1$DEFAULT"
}

function error_msg() {
  echo -e "  $RED$1$DEFAULT"
}

EXPECTED_OUTPUT_FILE=("ERROR: Can't open output stream.
ERROR: Program exited with errors.")

EXPECTED_OUTPUT_VALID_PARAMETERS=("SUCCESS: Output file has been written successfully")
```

```
43
44 function test_parameter_file(){
45   header "TEST VALIDACION PARAMETRO 'FILE'"
46
47     commands=(  "-o ./")
48
49     for i in "${commands[@]}"
50     do
51
52         msg "$EXEC $i"
53
54         ACTUAL_OUTPUT=$($EXEC $i 2>&1)
55
56
57     if [[ "$EXPECTED_OUTPUT_FILE" == "$ACTUAL_OUTPUT" ]]; then
58       msg_true "$EXPECTED_OUTPUT_FILE" "$ACTUAL_OUTPUT"
59     else
60       msg_false "$EXPECTED_OUTPUT_FILE" "$ACTUAL_OUTPUT"
61     fi
62
63     done
64 }
65
66 test_parameter_file
```

A.0.4. main.s

```
1     .file   1 "main.c"
2     .section .mdebug.abi32
3     .previous
4     .abicalls
5     .rdata
6     .align  2
7 cmdOptions:
8     .word   $LC0
9     .word   1
10    .word   0
11    .word   114
12    .word   $LC1
13    .word   1
14    .word   0
15    .word   99
16    .word   $LC2
17    .word   1
18    .word   0
```

```
19      .word   119
20      .word   $LC3
21      .word   1
22      .word   0
23      .word   72
24      .word   $LC4
25      .word   1
26      .word   0
27      .word   115
28      .word   $LC5
29      .word   1
30      .word   0
31      .word   111
32      .word   0
33      .word   0
34      .word   0
35      .word   0
36      .rdata
37      .align  2
38  $L48:
39      li  $v0,1           # 0x1
40      sw  $v0,40($fp)
41  $L41:
42      lw  $v1,40($fp)
43      li  $v0,1           # 0x1
44      bne $v1,$v0,$L38
45      li  $v0,1           # 0x1
46      sw  $v0,44($fp)
47      b   $L37
48  $L39:
49      sw  $zero,44($fp)
50  $L37:
51      lw  $v0,44($fp)
52      move    $sp,$fp
53      lw  $ra,64($sp)
54      lw  $fp,60($sp)
55      addu    $sp,$sp,72
56      j   $ra
57      .end    parseCmdline
58      .size   parseCmdline, .-parseCmdline
59      .rdata
60      .align  2
61  $LC19:
62      .word   1073741824
63      .align  3
64  $L96:
65      lw  $a0,56($fp)
```

```
66      la  $t9,fclose
67      jal $ra,$t9
68      lw  $a0,68($fp)
69      la  $t9,free
70      jal $ra,$t9
71      move    $v0,$zero
72      move    $sp,$fp
73      lw  $ra,96($sp)
74      lw  $fp,92($sp)
75      addu    $sp,$sp,104
76      j   $ra
77      .end    main
78      .size   main, .-main
79      .ident  "GCC: (GNU) 3.3.3 (NetBSD nb3 20040520)"
```