Universidad De Buenos Aires

Facultad De Ingeniería

66.20 Organización De Computadoras

Trabajo Práctico 0

Integrantes:
Gonzalo Beviglia - 93144
Federico Quevedo - 93159
Damián Manoff - 93169

10 de Septiembre de 2013

Índice

1. Diseño e implementación

Uno de los principales limitantes de las soluciones que pudimos implementar era el largo del archivo a invertir, ya que este podia llegar a ocupar mucho lugar en la memoria. Buscamos mejorar esta situación leyendo e invirtiendo de a un archivo por vez, para evitar tener mas de uno en memoria. También evitamos tener que declarar otro espacio de memoria del mismo tamaño del archivo original, donde iria la cadena invertida, haciendo una inversion in situ de la cadena, es decir, sobre la misma memoria reservada a donde se cargo el archivo. Esto ademas de ahorrarnos duplicar la memoria usada nos ahorra el tiempo de reservar la misma. Sobre la cadena original se realizan swaps espejados hasta llegar a la cadena invertida.

2. Performance

La performance se evaluó invirtiendo el libro "El Príncipe" de Nicolás Maquiavelo, y se comparó la performance del comando realizado para este trabajo práctico con la del comando unix *rev*. El tamaño de dicho texto en formato de texto plano es de 305864 bytes (298KB).

Los tiempos se midieron utilizando el comando Unix *time*.

2.1. Tiempo *ownRev*

```
real 0m0.539s

user 0m0.008s

sys 0m0.016s
```

2.2. Tiempo Unix *rev*

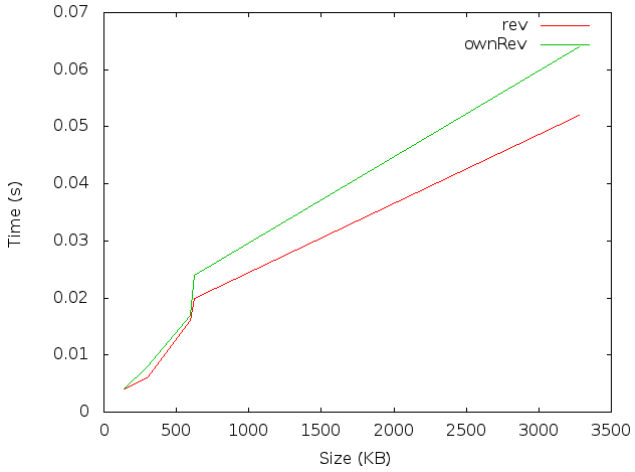
```
real 0m0.540s

user 0m0.012s

sys 0m0.024s
```

2.3. Comparación grafica

Los textos utilizados para esta comparación fueron:

- The Divine Comedy - Dante Alighieri (627KB)
- Adventures of Huckleberry Finn - Mark Twain (596KB)
- Les Miserables - Victor Hugo (3.2MB)
- Metamorphosis - Franz Kafka (139KB)
- The Prince - Nicolo Machiavelli (299KB)

Figura 1: Comparacion entre rev de Unix y nuestro “ownRev”



3. Compilación del programa

Para el compilado del programa hicimos el siguiente makefile:

```
CFLAGS=-Wall -pedantic -std=c99 -g
MEMFLAGS=valgrind --leak-check=full --track-origins=yes -v
CC=gcc
INPUT=ownRev.c
MIDDLEFILE=ownRev.o
EXEC=ownRev
TESTSCRIPT=TestFiles/tests.sh

all: $(EXEC) clean
test: all runTests cleanExec

.SILENT:
$(MIDDLEFILE):
	$(CC) $(CFLAGS) $(INPUT) -c

.SILENT:
$(EXEC): $(MIDDLEFILE)
	$(CC) $(CFLAGS) $(MIDDLEFILE) -o $(EXEC)

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

.SILENT:
runTests: $(EXEC)
	./$(TESTSCRIPT)

.SILENT:
memCheck: $(EXEC)
	$(MEMFLAGS) ./$(EXEC) TestFiles/test TestFiles/test1 TestFiles/test2

.SILENT:
clean: $(MIDDLEFILE)
	rm *.o
```

```
.SILENT:
cleanExec: $(EXEC)
	rm $(EXEC)
```

La ejecución normal de este make file produce el archivo ejecutable y ademas elimina los intermediarios.

Se puede tambien llamar pasando como parametro el nombre del archivo intermediario para generarlo, o el nombre del ejecutable, que realizara lo mismo que la ejecución por defecto pero sin eliminar el intermediario.

Para corroborar que no se estuviera perdiendo memoria tambien incluimos el parametro *memCheck* que corre el programa con valgrind informando si hubo o no alguna perdida.

Desde el mismo makefile tambien incluimos la posibilidad de correr las pruebas, y por ultimo, la de eliminar los archivos generados, tanto intermediarios como programa final.

4. Pruebas

Para las pruebas hicimos el siguiente script de bash:

```
#!/bin/bash
FILES="TestFiles/test
TestFiles/test1
TestFiles/test2"

for f in $FILES
do
	./ownRev $f > auxRev
	rev $f > auxCompRev
	DIFF=$(diff auxRev auxCompRev)
	if [ "$DIFF" != "" ]
	then
		echo "[ERROR] $f was not reversed correctly"
	else
		echo "[OK] $f was reversed correctly"
	fi
	rm auxRev
	rm auxCompRev
done
```

Se toman lineas de un archivo, en este caso llamado test, y se las invierte usando tanto nuestro programa como el comando *rev* de linux, si los resultados son iguales la prueba es valida.

5. Código Fuente

5.1. Código fuente C

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

typedef struct charWrap {
	char* string;
	unsigned length;
} charWrap;

/*
 * Checks the option passed via attribute. Prints a different output
 * depending on the option that was passed. Handles only one option
 * at a time.
 *
```

```
 * attr: The string with the option received.
 */
int checkOption(char* attr) {

        // Match version option.
        if( !strcmp(attr, "-v") || !strcmp(attr, "--version") ) {
                printf("Version 1.0.0\n");
                return 1;
        }
        // Match help option.
        else if ( !strcmp(attr, "-h") || !strcmp(attr, "--help") ) {
                printf("Usage\n");
                return 1;
        }

        return 0;

}
/*
 * Makes a swap in the given string. The positions swapped are pos1
 * and pos2.
 *
 * string: The string whose characters the function will swap.
 * pos1: First position in the string to swap.
 * pos2: Second position in the string to swap.
 */
void swapChars(char* string, int pos1, int pos2) {

        if( pos1 == pos2 ) return;

        char aux = string[pos2];
        string[pos2] = string[pos1];
        string[pos1] = aux;

}

/*
 * Copies the source string in a new space of memory and reverses
 * this copy. Returns the copy's reference.
 *
 * source: The string to reverse.
 */
char* reverseString(charWrap* source) {
        unsigned len = source -> length;

        // One more character for \0 terminator.
        char* reversed = (char*) malloc( (len + 1) * sizeof(char) );
        strncpy(reversed, source -> string, len);

        int posInitial = 0;
        // Start counting from 0, so length - 1. One less for \n.
        int posFinal = len - 2;

        while( posInitial < posFinal )
                swapChars(reversed, posInitial++, posFinal--);
```

```
        reversed[len] = '\0';

        return reversed;
}

/*
 * Reads a line from the given filePtr. If the line is empty,
 * returns NULL. If the line is not empty, returns a reference
 * to the string. reference must be freed after.
 *
 * filePtr: filePtr to be read.
 */
charWrap* readFromFile(FILE* filePtr) {

        // Initial buffer length.
        unsigned bufferLength = 30;
        // Allocate memory for buffer length and \0 terminator.
        char* finalString = (char*) malloc( (bufferLength + 2) * sizeof(char) );
        // Memory alloc error handling.
        if( finalString == NULL ) {
                fprintf(stderr, "Error: unable to allocate %d bytes on line 88\n", (bufferLen
                return NULL;
        }

        // Initialize aux variables.
        char* auxString = NULL;
        int character = 0;
        int length = 0;

        // Parse stdin until we get \n (should be EOF).
        character = fgetc(filePtr);
        while( !feof(filePtr) ) {
                length++;
                // Buffer has been filled. Allocate more memory.
                if( (length + 1) == bufferLength ) {
                        bufferLength = 2 * bufferLength;
                        auxString = (char*) realloc( finalString, bufferLength * sizeof(char)
                        // Memory alloc error handling.
                        if( auxString == NULL ) {
                                fprintf(stderr, "Error: unable to allocate %d bytes on line 1
                                free(finalString);
                                return NULL;
                        }
                        finalString = auxString;
                }
                finalString[length-1] = character;
                if( character == '\n' ) break;
                character = fgetc(filePtr);
        }

        if( length == 0 ) {
                free(finalString);
                return NULL;
        }

        auxString = (char*) realloc( finalString, length * sizeof(char) );
```

```
        // Memory alloc error handling.
        if( auxString == NULL ) {
                fprintf(stderr, "Error: unable to allocate %d bytes on line 126", bufferLengt
                return NULL;
        }
        charWrap* retVal = (charWrap*) malloc( sizeof(charWrap) );
        if( retVal == NULL ) {
                fprintf(stderr, "Error: unable to allocate %d bytes on line 132", sizeof(char
                return NULL;
        }
        retVal -> string = auxString;
        retVal -> length = length;
        return retVal;
}

/*
 * Prints the output of the given file after its lines were
 * reversed. File pointer must be already opened for reading
 * and must be closed after this call.
 *
 * fPtr: file to be reversed.
 */
void reverseFile(FILE* fPtr) {

        char* reversed = NULL;
        charWrap* fileString = readFromFile(fPtr);

        while( fileString != NULL ) {
                reversed = reverseString(fileString);
                printf("%s", reversed);
                free(fileString -> string);
                free(fileString);
                free(reversed);
                fileString = readFromFile(fPtr);
        }

}

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

        FILE* fPtr = NULL;

        // Rev from stdin.
        if( argc == 1 ) {
                reverseFile(stdin);
                return 0;
        }

        // Option may have been passed.
        if( argc == 2 ) {
                // Option was matched.
                if( checkOption(argv[1]) ) return 0;
        }
        unsigned i;
        for( i = 1 ; i < argc ; i++ ) {
                fPtr = fopen(argv[i], "r");
```

```
                // Handling opening file error.
                if( fPtr == NULL ) fprintf(stderr, "Error: unable to open file %s\n", argv[i]
                else {
                        reverseFile(fPtr);
                        fclose(fPtr);
                }
        }
        return 0;
}
```

5.2. Código assembly MIPS

A continuación se detallará el código assembly para la arquitectura MIPS de nuestro programa en C

```
        .file   1 "ownRev.c"
        .section .mdebug.abi32
        .previous
        .abicalls
        .rdata
        .align  2
$LC0:
        .ascii  "-v\000"
        .align  2
$LC1:
        .ascii  "--version\000"
        .align  2
$LC2:
        .ascii  "Version 1.0.0\n\000"
        .align  2
$LC3:
        .ascii  "-h\000"
        .align  2
$LC4:
        .ascii  "--help\000"
        .align  2
$LC5:
        .ascii  "Usage\n\000"
        .text
        .align  2
        .globl  checkOption
        .ent    checkOption
checkOption:
        .frame  $fp,48,$ra              # vars= 8, regs= 3/0, args= 16, extra= 8
        .mask   0xd0000000,-8
        .fmask  0x00000000,0
        .set    noreorder
        .cpload $t9
        .set    reorder
        subu    $sp,$sp,48
        .cprestore 16
        sw      $ra,40($sp)
        sw      $fp,36($sp)
        sw      $gp,32($sp)
        move    $fp,$sp
        sw      $a0,48($fp)
        lw      $a0,48($fp)
        la      $a1,$LC0
        la      $t9,strcmp
```

```
        jal     $ra,$t9
        beq     $v0,$zero,$L19
        lw      $a0,48($fp)
        la      $a1,$LC1
        la      $t9,strcmp
        jal     $ra,$t9
        bne     $v0,$zero,$L18
$L19:
        la      $a0,$LC2
        la      $t9,printf
        jal     $ra,$t9
        li      $v0,1                   # 0x1
        sw      $v0,24($fp)
        b       $L17
$L18:
        lw      $a0,48($fp)
        la      $a1,$LC3
        la      $t9,strcmp
        jal     $ra,$t9
        beq     $v0,$zero,$L22
        lw      $a0,48($fp)
        la      $a1,$LC4
        la      $t9,strcmp
        jal     $ra,$t9
        bne     $v0,$zero,$L20
$L22:
        la      $a0,$LC5
        la      $t9,printf
        jal     $ra,$t9
        li      $v0,1                   # 0x1
        sw      $v0,24($fp)
        b       $L17
$L20:
        sw      $zero,24($fp)
$L17:
        lw      $v0,24($fp)
        move    $sp,$fp
        lw      $ra,40($sp)
        lw      $fp,36($sp)
        addu    $sp,$sp,48
        j       $ra
        .end    checkOption
        .size   checkOption, .-checkOption
        .align  2
        .globl  swapChars
        .ent    swapChars
swapChars:
        .frame  $fp,24,$ra              # vars= 8, regs= 2/0, args= 0, extra= 8
        .mask   0x50000000,-4
        .fmask  0x00000000,0
        .set    noreorder
        .cpload $t9
        .set    reorder
        subu    $sp,$sp,24
        .cprestore 0
        sw      $fp,20($sp)
```

```
        sw      $gp,16($sp)
        move    $fp,$sp
        sw      $a0,24($fp)
        sw      $a1,28($fp)
        sw      $a2,32($fp)
        lw      $v1,28($fp)
        lw      $v0,32($fp)
        bne     $v1,$v0,$L24
        b       $L23
$L24:
        lw      $v1,24($fp)
        lw      $v0,32($fp)
        addu    $v0,$v1,$v0
        lbu     $v0,0($v0)
        sb      $v0,8($fp)
        lw      $v1,24($fp)
        lw      $v0,32($fp)
        addu    $a0,$v1,$v0
        lw      $v1,24($fp)
        lw      $v0,28($fp)
        addu    $v0,$v1,$v0
        lbu     $v0,0($v0)
        sb      $v0,0($a0)
        lw      $v1,24($fp)
        lw      $v0,28($fp)
        addu    $v1,$v1,$v0
        lbu     $v0,8($fp)
        sb      $v0,0($v1)
$L23:
        move    $sp,$fp
        lw      $fp,20($sp)
        addu    $sp,$sp,24
        j       $ra
        .end    swapChars
        .size   swapChars, .-swapChars
        .align  2
        .globl  reverseString
        .ent    reverseString
reverseString:
        .frame  $fp,56,$ra              # vars= 16, regs= 3/0, args= 16, extra= 8
        .mask   0xd0000000,-8
        .fmask  0x00000000,0
        .set    noreorder
        .cpload $t9
        .set    reorder
        subu    $sp,$sp,56
        .cprestore 16
        sw      $ra,48($sp)
        sw      $fp,44($sp)
        sw      $gp,40($sp)
        move    $fp,$sp
        sw      $a0,56($fp)
        lw      $v0,56($fp)
        lw      $v0,4($v0)
        sw      $v0,24($fp)
        lw      $v0,24($fp)
```

```
        addu    $v0,$v0,1
        move    $a0,$v0
        la      $t9,malloc
        jal     $ra,$t9
        sw      $v0,28($fp)
        lw      $v0,56($fp)
        lw      $a0,28($fp)
        lw      $a1,0($v0)
        lw      $a2,24($fp)
        la      $t9,strncpy
        jal     $ra,$t9
        sw      $zero,32($fp)
        lw      $v0,24($fp)
        addu    $v0,$v0,-2
        sw      $v0,36($fp)
$L26:
        lw      $v0,32($fp)
        lw      $v1,36($fp)
        slt     $v0,$v0,$v1
        bne     $v0,$zero,$L28
        b       $L27
$L28:
        addu    $v1,$fp,32
        lw      $v0,0($v1)
        move    $a1,$v0
        addu    $v0,$v0,1
        sw      $v0,0($v1)
        addu    $v1,$fp,36
        lw      $v0,0($v1)
        move    $a2,$v0
        addu    $v0,$v0,-1
        sw      $v0,0($v1)
        lw      $a0,28($fp)
        la      $t9,swapChars
        jal     $ra,$t9
        b       $L26
$L27:
        lw      $v1,28($fp)
        lw      $v0,24($fp)
        addu    $v0,$v1,$v0
        sb      $zero,0($v0)
        lw      $v0,28($fp)
        move    $sp,$fp
        lw      $ra,48($sp)
        lw      $fp,44($sp)
        addu    $sp,$sp,56
        j       $ra
        .end    reverseString
        .size   reverseString, .-reverseString
        .rdata
        .align  2
$LC6:
        .ascii  "Error: unable to allocate %d bytes on line 88\n\000"
        .align  2
$LC7:
        .ascii  "Error: unable to allocate %d bytes on line 107\n\000"
```

```
        .align  2
$LC8:
        .ascii  "Error: unable to allocate %d bytes on line 126\000"
        .align  2
$LC9:
        .ascii  "Error: unable to allocate %d bytes on line 132\000"
        .text
        .align  2
        .globl  readFromFile
        .ent    readFromFile
readFromFile:
        .frame  $fp,72,$ra              # vars= 32, regs= 3/0, args= 16, extra= 8
        .mask   0xd0000000,-8
        .fmask  0x00000000,0
        .set    noreorder
        .cpload $t9
        .set    reorder
        subu    $sp,$sp,72
        .cprestore 16
        sw      $ra,64($sp)
        sw      $fp,60($sp)
        sw      $gp,56($sp)
        move    $fp,$sp
        sw      $a0,72($fp)
        li      $v0,30                  # 0x1e
        sw      $v0,24($fp)
        lw      $v0,24($fp)
        addu    $v0,$v0,2
        move    $a0,$v0
        la      $t9,malloc
        jal     $ra,$t9
        sw      $v0,28($fp)
        lw      $v0,28($fp)
        bne     $v0,$zero,$L30
        lw      $v0,24($fp)
        addu    $v0,$v0,2
        la      $a0,__sF+176
        la      $a1,$LC6
        move    $a2,$v0
        la      $t9,fprintf
        jal     $ra,$t9
        sw      $zero,48($fp)
        b       $L29
$L30:
        sw      $zero,32($fp)
        sw      $zero,36($fp)
        sw      $zero,40($fp)
        lw      $a0,72($fp)
        la      $t9,fgetc
        jal     $ra,$t9
        sw      $v0,36($fp)
$L31:
        lw      $v0,72($fp)
        lhu     $v0,12($v0)
        srl     $v0,$v0,5
        andi    $v0,$v0,0x1
```

```
        beq     $v0,$zero,$L33
        b       $L32
$L33:
        lw      $v0,40($fp)
        addu    $v0,$v0,1
        sw      $v0,40($fp)
        lw      $v0,40($fp)
        addu    $v1,$v0,1
        lw      $v0,24($fp)
        bne     $v1,$v0,$L34
        lw      $v0,24($fp)
        sll     $v0,$v0,1
        sw      $v0,24($fp)
        lw      $a0,28($fp)
        lw      $a1,24($fp)
        la      $t9,realloc
        jal     $ra,$t9
        sw      $v0,32($fp)
        lw      $v0,32($fp)
        bne     $v0,$zero,$L35
        la      $a0,__sF+176
        la      $a1,$LC7
        lw      $a2,24($fp)
        la      $t9,fprintf
        jal     $ra,$t9
        lw      $a0,28($fp)
        la      $t9,free
        jal     $ra,$t9
        sw      $zero,48($fp)
        b       $L29
$L35:
        lw      $v0,32($fp)
        sw      $v0,28($fp)
$L34:
        lw      $v1,28($fp)
        lw      $v0,40($fp)
        addu    $v0,$v1,$v0
        addu    $v1,$v0,-1
        lbu     $v0,36($fp)
        sb      $v0,0($v1)
        lw      $v1,36($fp)
        li      $v0,10                  # 0xa
        bne     $v1,$v0,$L36
        b       $L32
$L36:
        lw      $a0,72($fp)
        la      $t9,fgetc
        jal     $ra,$t9
        sw      $v0,36($fp)
        b       $L31
$L32:
        lw      $v0,40($fp)
        bne     $v0,$zero,$L37
        lw      $a0,28($fp)
        la      $t9,free
        jal     $ra,$t9
```

```
        sw      $zero,48($fp)
        b       $L29
$L37:
        lw      $a0,28($fp)
        lw      $a1,40($fp)
        la      $t9,realloc
        jal     $ra,$t9
        sw      $v0,32($fp)
        lw      $v0,32($fp)
        bne     $v0,$zero,$L38
        la      $a0,__sF+176
        la      $a1,$LC8
        lw      $a2,24($fp)
        la      $t9,fprintf
        jal     $ra,$t9
        sw      $zero,48($fp)
        b       $L29
$L38:
        li      $a0,8                   # 0x8
        la      $t9,malloc
        jal     $ra,$t9
        sw      $v0,44($fp)
        lw      $v0,44($fp)
        bne     $v0,$zero,$L39
        la      $a0,__sF+176
        la      $a1,$LC9
        li      $a2,8                   # 0x8
        la      $t9,fprintf
        jal     $ra,$t9
        sw      $zero,48($fp)
        b       $L29
$L39:
        lw      $v1,44($fp)
        lw      $v0,32($fp)
        sw      $v0,0($v1)
        lw      $v1,44($fp)
        lw      $v0,40($fp)
        sw      $v0,4($v1)
        lw      $v0,44($fp)
        sw      $v0,48($fp)
$L29:
        lw      $v0,48($fp)
        move    $sp,$fp
        lw      $ra,64($sp)
        lw      $fp,60($sp)
        addu    $sp,$sp,72
        j       $ra
        .end    readFromFile
        .size   readFromFile, .-readFromFile
        .rdata
        .align  2
$LC10:
        .ascii  "%s\000"
        .text
        .align  2
        .globl  reverseFile
```

```
        .ent    reverseFile
reverseFile:
        .frame  $fp,48,$ra              # vars= 8, regs= 3/0, args= 16, extra= 8
        .mask   0xd0000000,-8
        .fmask  0x00000000,0
        .set    noreorder
        .cpload $t9
        .set    reorder
        subu    $sp,$sp,48
        .cprestore 16
        sw      $ra,40($sp)
        sw      $fp,36($sp)
        sw      $gp,32($sp)
        move    $fp,$sp
        sw      $a0,48($fp)
        sw      $zero,24($fp)
        lw      $a0,48($fp)
        la      $t9,readFromFile
        jal     $ra,$t9
        sw      $v0,28($fp)
$L41:
        lw      $v0,28($fp)
        bne     $v0,$zero,$L43
        b       $L40
$L43:
        lw      $a0,28($fp)
        la      $t9,reverseString
        jal     $ra,$t9
        sw      $v0,24($fp)
        la      $a0,$LC10
        lw      $a1,24($fp)
        la      $t9,printf
        jal     $ra,$t9
        lw      $v0,28($fp)
        lw      $a0,0($v0)
        la      $t9,free
        jal     $ra,$t9
        lw      $a0,28($fp)
        la      $t9,free
        jal     $ra,$t9
        lw      $a0,24($fp)
        la      $t9,free
        jal     $ra,$t9
        lw      $a0,48($fp)
        la      $t9,readFromFile
        jal     $ra,$t9
        sw      $v0,28($fp)
        b       $L41
$L40:
        move    $sp,$fp
        lw      $ra,40($sp)
        lw      $fp,36($sp)
        addu    $sp,$sp,48
        j       $ra
        .end    reverseFile
        .size   reverseFile, .-reverseFile
```

```
        .rdata
        .align  2
$LC11:
        .ascii  "r\000"
        .align  2
$LC12:
        .ascii  "Error: unable to open file %s\n\000"
        .text
        .align  2
        .globl  main
        .ent    main
main:
        .frame  $fp,56,$ra              # vars= 16, regs= 3/0, args= 16, extra= 8
        .mask   0xd0000000,-8
        .fmask  0x00000000,0
        .set    noreorder
        .cpload $t9
        .set    reorder
        subu    $sp,$sp,56
        .cprestore 16
        sw      $ra,48($sp)
        sw      $fp,44($sp)
        sw      $gp,40($sp)
        move    $fp,$sp
        sw      $a0,56($fp)
        sw      $a1,60($fp)
        sw      $zero,24($fp)
        lw      $v1,56($fp)
        li      $v0,1                   # 0x1
        bne     $v1,$v0,$L45
        la      $a0,__sF
        la      $t9,reverseFile
        jal     $ra,$t9
        sw      $zero,32($fp)
        b       $L44
$L45:
        lw      $v1,56($fp)
        li      $v0,2                   # 0x2
        bne     $v1,$v0,$L46
        lw      $v0,60($fp)
        addu    $v0,$v0,4
        lw      $a0,0($v0)
        la      $t9,checkOption
        jal     $ra,$t9
        beq     $v0,$zero,$L46
        sw      $zero,32($fp)
        b       $L44
$L46:
        li      $v0,1                   # 0x1
        sw      $v0,28($fp)
$L48:
        lw      $v0,28($fp)
        lw      $v1,56($fp)
        sltu    $v0,$v0,$v1
        bne     $v0,$zero,$L51
        b       $L49
```

```
$L51:
        lw      $v0,28($fp)
        sll     $v1,$v0,2
        lw      $v0,60($fp)
        addu    $v0,$v1,$v0
        lw      $a0,0($v0)
        la      $a1,$LC11
        la      $t9,fopen
        jal     $ra,$t9
        sw      $v0,24($fp)
        lw      $v0,24($fp)
        bne     $v0,$zero,$L52
        lw      $v0,28($fp)
        sll     $v1,$v0,2
        lw      $v0,60($fp)
        addu    $v0,$v1,$v0
        la      $a0,__sF+176
        la      $a1,$LC12
        lw      $a2,0($v0)
        la      $t9,fprintf
        jal     $ra,$t9
        b       $L50
$L52:
        lw      $a0,24($fp)
        la      $t9,reverseFile
        jal     $ra,$t9
        lw      $a0,24($fp)
        la      $t9,fclose
        jal     $ra,$t9
$L50:
        lw      $v0,28($fp)
        addu    $v0,$v0,1
        sw      $v0,28($fp)
        b       $L48
$L49:
        sw      $zero,32($fp)
$L44:
        lw      $v0,32($fp)
        move    $sp,$fp
        lw      $ra,48($sp)
        lw      $fp,44($sp)
        addu    $sp,$sp,56
        j       $ra
        .end    main
        .size   main, .-main
        .ident  "GCC: (GNU) 3.3.3 (NetBSD nb3 20040520)"
```

6. Conclusiones

Luego de realizar este trabajo práctico, podemos concluir que el comando “rev” es sencillo de implementar al nivel al que estuvimos interesados a la hora de desarrollarlo. Sería interesante indagar el código fuente del comando original de Linux para ver que tan alejados estamos de una implementación óptima.

Se nos ocurre que, por ejemplo, el comando “rev” debe hacer un checkeo de errores más exhaustivo, que a nosotros, como alumnos, quizas no fuimos capaces de observarlos como posibles.

Es interesante ver como los tiempos de ejecución para ambas implementaciones son similares, siendo estos

tiempos obtenidos mediante la reversión de archivos de texto grandes, sin imprimir el resultado por pantalla.

En nuestra opinion, uno de los elementos más importantes y más útiles a la hora del desarrollo y la corrección del trabajo práctico, fue el uso de las pruebas automatizadas. Fueron la herramienta que nos permitió consolidar una base de código en funcionamiento para luego poder optimizar al programa y corregir errores indicados por el corrector.

Con lo que lleva a la asignatura se nota claramente una diferencia entre en código assembly de MIPs y el código del programa en C. Cuando en C tenemos casi 200 lineas de código en assembly llegamos casi a las 600 es decir el triple de lineas de código por el mismo programa. Claramente notamos lo dificultoso y tedioso que es programar en este lenguaje, pero el poder que tiene es muchisimo mayor, ya que notamos paso a paso , linea a linea como funciona correctamente nuestro trabajo, y en caso de haber *Bugs* que a simple vista en C no podriamos encontrar, aca claramente hallaríamos el problema. Es decir, cuando no nos quede otra alternativa para solucionar los problemas, seguramente sabiendo leer el código de máquina tendremos un panorama mas claro del problema y así talvez llegar a una solución factible.