Trabajo Práctico 0: Infraestructura básica

Damián Muszalski, *Padrón Nro. 88462*
damianfiuba@gmail.com
Federico García, *Padrón Nro.*
fedeagb@gmail.com
Nicolás Fernandez Lema, *Padrón Nro.*
nicolasfernandezlema@gmail.com

2do. Cuatrimestre de 2013
66.20 Organización de Computadoras
Facultad de Ingeniería, Universidad de Buenos Aires

10/09/2013

1. Objetivo

El objetivo de este trabajo es familiarizarse con las herramientas que utilizaremos para emular un entorno MIPS32.

2. Introducción

En el presente trabajo práctico, se deben implementar en lenguaje C una versión del comando *rev* de UNIX. El mismo concatena y escribe en stdout el contenido de uno o mas archivos, invirtiendo el orden de los caracteres de cada línea. En nuestro caso se asume que cada caracter mide 1 byte.

3. Implementación

El programa debe leer los datos de entrada desde stdin o bien desde uno o más archivos. La salida del programa debe imprimirse por stdout, mientras que los errores deben imprimirse por stderr. La ayuda y versión del programa puede seleccionarse mediante las opciones -h o -V respectivamente. La implementación del comando debe consiste en una función con el siguiente prototipo: int procesarArchivo(FILE* fd);

4. Desarrollo

A continuación se citan los pasos a que se siguieron para hacer el trabajo práctico.

4.1. Paso 1: Configuración de Entorno de Desarrollo

El primer paso fue configurar el entorno de desarrollo, de acuerdo a la guía facilitada por la cátedra. Trabajamos con la distribución Ubuntu (basada en Debian) que se puede bajar libremente de http://www.ubuntu.com/. Se realizó posteriormente la instalación de GxEmul para emular un sistema MIPS; se utilizó el proporcionado por la cátedra, el cual traía una imagen del sistema operativo NetBSD con algunas utilidades (compilador C, ssh, editor vi, etc).

4.2. Paso 2: Implementación del programa

El programa se subdividió en dos funciones:
- main, encargada del parseo de los parámetros de entrada.
- procesarArchivo, encargada de realizar la funcionalidad del comando rev.

El programa debe ejecutarse por línea de comando, donde la salida sera también por consola.

4.2.1. Implementación de las funciones

- main Es la función principal, encargada de tomar los parámetros de entrada. Luego imprimir ayuda o versión si corresponde, o realizar llamadas sucesivas a la función procesarArchivo con el puntero a archivo que corresponda, ya sea el cual cuyo nombre se recibe como parámetro o simplemente *stdin*.

- procesarArchivo Es la función encargada de realizar la inversión de los caracteres de las líneas de los archivos de entrada. Lee secuencialmente las líneas reservando 1 byte de memoria adicional con cada caracter leído.
  Una vez en memoria se arma un segundo arreglo de caracteres donde se copia la línea invirtiendola. Finalmente se imprime por pantalla esta línea invertida.
  Se libera la memoria y se lee la siguiente línea del archivo. El ciclo corta al llegar a una señal de EOF.

4.2.2. Ingreso de parámetros

El formato para invocar al programa es el siguiente:
./tp1 nombreArchivo1 nombreArchivo2 .. nombreArchivoN
Donde nombreArchivoX posee el texto al cual aplicar el comando rev.

5. Código para compilar el programa con gcc

El proyecto se debe compilar tanto en el sistema Host (en nuestro caso Ubuntu) como en el Guest (NetBSD). Se dispone del mismo compilador en ambos sistemas por lo tanto para ambos casos debemos situarnos en el directorio donde se encuentran todos los fuentes y utilizar el siguiente comando:

```
gcc -o tp0 tp0.c
```

Con esto se generará un archivo ejecutable, llamado tp0.

6. Ejemplos de ejecución

A continuación se mostrarán algunos ejemplos de archivos sobre los cuales se utiliza el programa realizado.

7. Conclusión

8. Bibliografía

- Material de la cátedra
  Se puede encontrar en el grupo Yahoo:
  http://groups.yahoo.com/neo/groups/orga-comp/files

9. Código Fuente

9.1. tp0.c

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

#define VERSION 3

// Definicion de funcines

int procesarArchivo(FILE* fd);

// Funcion principal
int main(int argc, char** argv) {
int status = 0;

if(argc == 2) {
if(strcmp(argv[1], "-V") == 0) {
printf("Version: %d\n", VERSION);
// No se si solo eso :P
return 0;
}
if(strcmp(argv[1], "-h") == 0) {
printf("Comandos y argumentos disponibles:\n");
printf("-V Version del programa\n");
printf("[File...] (Archivo/s de entrada)\n");
printf("En caso de no pasar archivos se toma la entrada estandar\n");
printf("Cada linea se cuenta hasta un enter\n");
printf("Para finalizar el programa estando con la entrada estandar");
printf(" pulsar 'ctrl+d' para un correcto cierre del mismo");
// Y cualquier otra cosa que se quiera agregar
return 0;
}
}
if(argc == 1) {
status = procesarArchivo(stdin);
}
else{
int arch = 1;
FILE* entrada;
while(arch < argc) {
entrada = fopen(argv[arch], "r");
if(!entrada) {
fprintf(stderr, "No se pudo abrir el archivo: %s\n", argv[arch]);
return 1;
}
status = procesarArchivo(entrada);
if(status)
```

```
return status;
fclose(entrada);
entrada = NULL;
arch++;
}
}
return status;

}

// Procesa el archivo de entrada (puede ser stdin)
// Invierte las lineas del archivo y las imprime por stdout
// En caso satisfactorio devuelve 0, distinto de 0 en otros casos
int procesarArchivo(FILE* fd){
int nextChar = fgetc(fd);
size_t sizeOfChar = sizeof(unsigned char);
unsigned char* line = malloc(sizeOfChar);
if(!line) {
fprintf(stderr, "Problema al querer alocar memoria\n");
return -1;
}
int lineWidth = 0;
int result = 0;
int aux_special;
while(nextChar != EOF) {
if(nextChar > 127) { // Para solucionar tema de caracteres especiales
lineWidth += 2;
line = (unsigned char*) realloc(line, lineWidth*sizeOfChar);
if(!line) {
fprintf(stderr, "Problema al querer alocar memoria\n");
return -1;
}
aux_special = nextChar;
nextChar = fgetc(fd);
line[lineWidth - 2] = (unsigned char) nextChar;
line[lineWidth - 1] = (unsigned char) aux_special;
nextChar = fgetc(fd);
continue;
}
if(nextChar != '\n') {
lineWidth++;
line = (unsigned char*) realloc(line,lineWidth*sizeOfChar);
if(!line) {
fprintf(stderr, "Problema al querer alocar memoria\n");
return -1;
}
line[lineWidth - 1] = (unsigned char) nextChar;
}
else {
unsigned char* inverseLine = (unsigned char*) malloc((lineWidth + 1)*sizeOfChar);
```

```
for(int pos = 0; pos < lineWidth; pos++){
inverseLine[pos] = line[(lineWidth - 1) - pos];
}
inverseLine[lineWidth] = '\n';
int outStatus = fwrite ( inverseLine, sizeOfChar, lineWidth + 1, stdout);
if( outStatus != lineWidth + 1) {
fprintf(stderr, "Error al escribir en salida\n");
result = 2;
}
free(inverseLine);
lineWidth = 0;
}
nextChar = fgetc(fd);
}
free(line);
return result;
}
```

10. Código MIPS generado por el compilador

10.1. tp0.s