Práctica 1.- Introducción al manejo de C y medidas de rendimiento

Entregables

El alumno elaborará un documento que deberá subir antes de la fecha señalada usando el "Aula virtual". Fuera de plazo no se aceptará la entrega de prácticas. La puntuación máxima de esta práctica es de 10 puntos. La puntuación de cada actividad se indica entre paréntesis al final de su enunciado. Todas las actividades y capturas de pantalla que se piden en las actividades deben estar incluidas y claramente identificadas en el documento.

Material necesario

- Acceso a un equipo con sistema Linux con el compilador GCC y el depurador GDB instalados

PARTE 1: Introducción a la programación en C

En esta práctica el alumno se familiarizará con los entornos de desarrollo en C.

Objetivos

- Aprender a compilar, ejecutar y depurar un programa en lenguaje C.

Desarrollo

El compilador de C instalado en la máquina suministrada al alumno, así como de manera general en todas las distribuciones de Linux, es el gcc (GNU C Compiler). La herramienta básica de trabajo para escribir un programa en Linux y demás sistemas UNIX es un editor de texto, ya sea desde consola (*vi*, *vim*, *joe*, *pico*, *nano*) o en modo gráfico (*mousepad*, *gedit*, *kedit*, *leafpad*). Con el editor se escribe el código fuente del programa a desarrollar y posteriormente se compila llamando desde la línea de comandos al compilador "gcc".

Para esta primera parte usaremos el editor de texto suministrado con la máquina y denominado *Mousepad*. Podemos acceder a él desde el menú de aplicaciones en la opción de accesorios. No obstante, también podemos instalar cualquier otro editor de texto con el que nos sintamos más cómodos (recordando que el espacio en disco puede ser limitado). Una vez comprobado que se tiene acceso a un editor de textos procederemos a escribir nuestro primer programa en C. Se puede guardar donde se quiera, sin embargo, se recomienda preparar una carpeta denominada "Practica1" para tener organizado el trabajo de esta sesión. Es necesario escribir el siguiente código y guardarlo con el nombre de archivo "hola.c":

```c
#include<stdio.h>
#include<stdlib.h>
int main() {
      int i=0;
      printf("Hola.\n");
      printf("El valor de i es = %d \n", i);
}
```

Una vez guardado el código en texto será necesario compilarlo. Para realizar este proceso utilizaremos una terminal. Accederemos hasta el directorio donde se encuentra ubicado nuestro programa con el comando "cd" y, una vez colocado en él, lanzaremos el compilador para que realice la compilación de la siguiente manera: "gcc hola.c". Este proceso nos creará un fichero binario ejecutable por el sistema denominado "a.out" para ejecutar el fichero binario, desde la terminal escribiremos "./a.out" y obtendremos una salida por pantalla.

- **Actividad 1: Escribir en un archivo la salida que obtenemos por pantalla para posteriormente enviarlo. (0.5 p.)**
- **Actividad 2: Buscar en las opciones del compilador "gcc", cual es la que nos permite establecer un nombre al fichero binario ejecutable de salida. ¿Qué parámetro es? (0.5 p.)**
- **Actividad 3: El proceso de generación de un fichero binario ejecutable se compone de dos fases: Una primera de compilación y otra segunda de enlazado (link) de las librerías. ¿Qué comandos tendríamos que ejecutar para realizar esta compilación en dos fases de nuestro programa hola.c? (0.5 p.)**

En muchas ocasiones, el compilador de c no informa sobre posibles errores en operaciones de acceso a memoria o cuando se mezclan distintos tipos de datos. Esto puede producir errores en los resultados esperados por nuestro programa. Otras veces el programa aborta su ejecución, de forma repentina, por algún problema interno. Para detectar estos posibles errores, existe una herramienta de depuración denominada "gdb", que está instalada en la máquina proporcionada. La llamada se realiza desde la línea de comandos de la siguiente manera "gdb ./a.out", si nuestro programa ejecutable se denomina "a.out". Una vez ejecutado obtendremos una salida por pantalla similar a la siguiente:

```
alumno@debian:~$ gdb ./hola
GNU gdb 6.8-debian
Copyright (C) 2008 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details. This GDB was configured as "i486-linux-gnu"...
(gdb)
```

Para proceder a la ejecución del programa en modo depuración escribimos "run" en la consola del depurador.

- **Actividad 4: Acceda a este enlace: https://www.cs.cmu.edu/~gilpin/tutorial/ Se pide hacer un resumen del documento de no más de media cara de folio A-4. ¿Qué**

parámetro del compilador necesitamos usar para que funcione nuestro depurador? (1 p.)

- **<u>Actividad 5</u>: ¿Qué es Valgrind? Responda brevemente y céntrese en su aplicación relacionada con la memoria. Instale Valgrind en su máquina (*"sudo apt-get install valgrind"*). Haga un ejemplo con una fuga o "leak" de memoria y otro con una lectura inválida de memoria y registre la salida de Valgrind en ambos casos. ¿Cómo valora su utilidad en relación a GDB? (1 p.)**

PARTE 2: Introducción al rendimiento de una aplicación

En esta práctica el alumno conocerá la influencia que tiene sobre la velocidad de los programas el acceso a memoria.

Objetivos

- Aprender a compilar, ejecutar y depurar un programa en lenguaje C.
- Conocer la influencia que produce el recorrido erróneo de un bucle en la velocidad de ejecución de un programa.

Desarrollo

1) Dado el siguiente código C:

```
#include<stdio.h>
#include<stdlib.h>
int main() {
      int i, j, h;
      double column_sum[1000];
      double b[1000][1000];
      for (i = 0; i < 1000; i++) {
            for (j = 0; j < 1000; j++) {
                  b[j][i] = 1.0;
            }
      }
      for (h = 0; h < 50; h++) {
            for (i = 0; i < 1000; i++) {
                  column_sum[i] = 0.0;

                  for (j = 0; j < 1000; j++) {
                        column_sum[i] += b[j][i];
                  }
            }
      }
}
```

- **<u>Actividad 1</u>: ¿Qué realiza? (0.5 p.)**

El comando "*time*" del sistema operativo Unix y por añadido Linux, mide el tiempo empleado en la ejecución de un programa dado como parámetro. Así, por ejemplo, si desde el "prompt"

del sistema ejecutamos "*time ls*", obtendremos además de la salida por pantalla del comando ls, el tiempo empleado en la ejecución del mismo. Obtenemos tres valores de tiempo:

```
real 0m0.007s
user 0m0.000s
sys 0m0.007s
```

El campo real especifica el tiempo real de ejecución, el tiempo user es el tiempo de usuario y el sys es el tiempo empleado por el sistema operativo. Normalmente: real = user + sys, aunque esto no es del todo ciento como veremos más adelante cuando trabajemos con threads.

Nosotros ahora mismo nos vamos a centrar en el campo real.

<u>Actividad 2</u>: Compilar el programa anterior y decir cuál es el tiempo real empleado para distintos valores de h (50,100,500,1000,5000,10000). (0.5 p.)

2) Dado el siguiente código en C:

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

int main() {
      int i, j, h;
      double column_sum[1000];
      double b[1000][1000];

      for (i = 0; i < 1000; i++) {
            for (j = 0; j < 1000; j++) {
                  b[j][i] = 1.0;
            }
      }

      for (h = 0; h < 50; h++) {
            for (i = 0; i < 1000; i++) {
                  column_sum[i] = 0.0;
            }

            for (j = 0; j < 1000; j++) {
                  for (i = 0; i < 1000; i++) {
                        column_sum[i] += b[j][i];
                  }
            }
      }
}
```

<u>Actividad 3</u>: ¿Qué realiza? (0.5 p.)

<u>Actividad 4</u>: Compilar el programa anterior y decir cuál es el tiempo real empleado para distintos valores de h (50,100,500,1000,5000,10000). (0.5 p.)

<u>Actividad 5</u>: Calcular la aceleración del programa 2 respecto del programa 1. Realizar una gráfica de la aceleración para los distintos valores de h. Dar una breve explicación de las diferencias entre los dos programas. ¿Por qué crees que sucede esto? (1 p.)

<u>Actividad 6</u>: Realizar un programa que multiplique un vector por una matriz de manera eficiente y otro programa que no. El número de elementos del vector y de la matriz es un parámetro que podemos indicar desde la línea de comandos cuando ejecutemos el programa. Vector y matriz se reservarán dinámicamente en memoria "heap", teniendo el vector una dimensión y la matriz dos. Hacer una gráfica de la eficiencia del programa eficiente respecto del que no lo es para distintos valores de la matriz y el vector. (2 p.)

<u>Actividad 7</u>: Modificar uno de los programas de la actividad 6 para que la matriz se reserve dinámicamente en un único bloque de memoria "heap" (una dimensión) y explicar los cambios en el código. (1.5 p.)