Universidad de Alcalá

DEPARTAMENTO DE AUTOMÁTICA
ARQUITECTURA Y TECNOLOGÍA DE COMPUTADORES

Grado en Ingeniería de Computadores
COMPUTACIÓN DE ALTAS PRESTACIONES

Práctica 0

Introducción a la programación en C

OBJETIVO

Se pretende adquirir una familiaridad con el lenguaje de programación C, dada su práctica ubicuidad y la utilización que de él haremos en los sucesivos laboratorios de la asignatura. Esta práctica no es obligatoria: puede ser obviada si se juzga oportuno.

Como objetivo adicional, aprendemos aquí también a conectarnos al nodo de usuario del cluster HPC-ATC2. Será en este nodo en donde realizaremos todas las prácticas, editando los programas, compilando y ordenando la ejecución.

RESULTADOS

Esta práctica no requiere presentación formal de resultados, pues su objetivo es, como se ha dicho, facilitar el primer contacto con el entorno de programación y las plataformas de computación que usaremos en el laboratorio.

DESARROLLO DE LA PRÁCTICA

Presentarse en la máquina

Abrimos un terminal alfanumérico, buscándolo para ello en los iconos que se encuentran en la izquierda de la pantalla.

Nos conectamos ahora al nodo de presentación del cluster. Este nodo se llama sesamo y su dirección IP es 172.29.23.142. El profesor nos proporcionará una cuenta y una clave que utilizaremos para presentarnos en el cluster. Si esa cuenta es, por ejemplo, cap_prac1, y estamos sentados frente a la máquina LN05P01 del laboratorio, podemos usar el siguiente comando:

```
larq@LN05P01:~$ ssh -X cap_prac1@sesamo
```

El sistema nos pide la clave y, si todo va bien, veremos el *prompt*:

```
cap_prac1@sesamo:~$
```

El cluster está compuesto de un nodo *frontend*, (el nodo sesamo), diez nodos de cómputo, cuatro nodos como servidores de disco y un nodo adicional que sirve disco y controla los accesos.

El estado actual del cluster puede verse accediendo a la siguiente página web:

http://sesamo/ganglia/

Edición de un programa

Las fases del desarrollo comprenden la creación del código fuente, editándolo con un editor de textos; la compilación para crear un ejecutable; y la ejecución para comprobar la funcionalidad. Por razón de orden, puede ser interesante crear una carpeta para cada práctica y almacenar allí los distintos ficheros. Para ello podemos escribir este comando:

```
cap_prac1@sesamo:~$ mkdir P0
```

que creará una carpeta con el nombre `P0`.

Para editar un programa, ejecutamos algún editor de textos. En nuestro caso, disponemos de uno muy sencillo, llamado `nedit`, que podemos invocar directamente desde el terminal:

```
cap_prac1@sesamo:~$ nedit &
```

Este editor es muy sencillo de usar y sirve para cualquier tipo de fichero textual. Si somos amantes del `vi`, disponemos de una versión gráfica que podemos invocar mediante el comando:

```
cap_prac1@sesamo:~$ xvile
```

En el momento de guardar la edición, elegiremos la carpeta antes creada como destino de nuestro fichero fuente.

Compilación y ejecución

La compilación de un programa C, `ejemplo.c`, se realizará en el terminal alfanumérico. En primer lugar, nos situamos en la carpeta en donde se han almacenado los ficheros fuente, con el comando:

```
cap_prac1@sesamo:~$ cd P0
```

Después se puede compilar el programa con el siguiente comando:

```
cap_prac1@sesamo:~/P0$ gcc -o ejemplo ejemplo.c -lm
```

Para ejecutarlo, basta teclear

```
cap_prac1@sesamo:~/P0$ ./ejemplo
```

Téngase en cuenta que el nodo `sesamo` es simplemente un *frontend* que nos da acceso al cluster y no está pensado para ejecutar programas pesados. El *frontend* simplemente nos permite hacer el desarrollo y probar la funcionalidad, pero la ejecución debe hacerse sobre los nodos de cómputo utilizando para ello el gestor de trabajos SLURM, descrito en otro documento.

A modo de ejemplo, si quisiéramos ejecutar el anterior programa sobre el cluster, una forma sencilla de hacerlo sería la siguiente:

```
cap_prac1@sesamo:~/P0$ srun ./ejemplo
```

El gestor SLURM seleccionará un nodo libre de la partición por defecto y ejecutará en él nuestro programa. Si no hay ninguno libre, el comando se quedará bloqueado hasta que se liberen los recursos necesarios.

EJEMPLOS DE PROGRAMAS

Incluimos a continuación algunos ejemplos de programas sencillos, que pueden servir de plantilla para empezar a hacer otros más complicados.

El inevitable “Hola, mundo”:

```
# include <stdio.h>

int main(int argc, char *argv[])
{
    printf("Hola, mundo.\n");
    return 0;
}
```

A continuación, detallo algunos posibles ejemplos. En internet hay hay cientos de tutoriales, con muchos más, que podéis “bajar” y utilizar.

Una rutina de entrada/salida:

```
# include <stdio.h>

void main()
{
int i, nc;

   nc = 0;
   i = getchar();
   while (i != EOF) {
      nc = nc + 1;
      i = getchar();
   }
   printf("Number of characters in file = %d\n", nc);
}
```

Aquí tenéis uno de cómputo:

```
            /************************/
            /*   Table of           */
            /*   Sine Function      */
            /************************/

            /* Michel Vallieres  */
            /* Written: Winter 1995 */
# include <stdio.h>
# include <math.h>

void main()
{
int    angle_degree;
double angle_radian, pi, value;

   /* Print a header */
   printf("\nCompute a table of the sine function\n\n");
```

```
   /* obtain pi once for all */
   /* or just use pi = M_PI, where
      M_PI is defined in math.h  */
   pi = 4.0*atan(1.0);
   printf("Value of PI = %f \n\n", pi);

   printf("angle     Sine \n");

   /* initial angle value      */
   angle_degree = 0;

   /* scan over angle       */
   /* loop until angle_degree > 360 */
   while (angle_degree <= 360)
   {
      angle_radian = pi * angle_degree/180.0 ;
      value = sin(angle_radian);
      printf(" %3d      %f \n", angle_degree, value);

      /* increment the loop index  */
      angle_degree = angle_degree + 10;
   }
}
```

Ahora uno manejando punteros:

```
void main()
{
float x, y;      /* x and y are of float type         */
float *fp, *fp2; /* fp and fp2 are pointers to float  */

   x = 6.5; /* x now contains the value 6.5       */

   /* print contents and address of x   */
   printf("Value of x is %f, address of x %lu\n", x, &x);

   fp = &x; /* fp now points to location of x    */

   /* print the contents of fp        */
   printf("Value in memory location fp is %f\n", *fp);

   /* change content of memory location */
   *fp = 9.2;
   printf("New value of x is %f = %f \n", *fp, x);

   /* perform arithmetic               */
   *fp = *fp + 1.5;
   printf("Final value of x is %f = %f \n", *fp, x);

   /* transfer values                  */
   y = *fp;
   fp2 = fp;
```

```
   printf("Transferred value into y"
          " = %f and fp2 = %f \n", y, *fp2);
}
```

En este ejemplo se trata con punteros y vectores:

```
#define SIZE 3

void main()
{
float x[SIZE];
float *fp;
int   i;
   /* initialize the array x         */
   /* use a "cast" to force i        */
   /* into the equivalent float      */
   for (i = 0; i < SIZE; i++)
      x[i] = 0.5*(float)i;

   /* print x                        */
   for (i = 0; i < SIZE; i++)
      printf("  %d  %f \n", i, x[i]);

   /* make fp point to array x       */
   fp = x;
   /* print via pointer arithmetic   */
   /* members of x are adjacent to   */
   /* each other in memory           */
   /* *(fp+i) refers to content of    */
   /* memory location (fp+i) or x[i] */
   for (i = 0; i < SIZE; i++)
      printf("  %d  %f \n", i, *(fp+i));
}
```

Uno de manejo de cadenas de caracteres:

```
# include <string.h>

void main()
{
char line[100], *sub_text;

   /* initialize string */
   strcpy(line,"hello, I am a string;");
   printf("Line: %s\n", line);

   /* add to end of string */
   strcat(line," what are you?");
   printf("Line: %s\n", line);

   /* find length of string */
   /* strlen brings back    */
```

```
   /* length as type size_t */

   printf("Length of line: %d\n", (int)strlen(line));

   /* find occurrence of substrings */
   if ((sub_text = strchr(line, 'W')) != NULL)
      printf("String starting with \"W\" ->%s\n", sub_text);

   if ((sub_text = strchr(line, 'w')) != NULL)
      printf("String starting with \"w\" ->%s\n", sub_text);

   if ((sub_text = strchr(sub_text, 'u')) != NULL)
      printf("String starting with \"u\" ->%s\n", sub_text);

}
```

Para manejo de funciones, tenemos:

```
# include <stdio.h>


void exchange(int *a, int *b);

void main()
{
int a, b;

   a = 5;
   b = 7;
   printf("From main: a = %d, b = %d\n", a, b);

   exchange(&a, &b);
   printf("Back in main: ");
   printf("a = %d, b = %d\n", a, b);
}

void exchange(int *a, int *b)
{
int temp;

   temp = *a;
   *a = *b;
   *b = temp;
   printf(" From function exchange: ");
   printf("a = %d, b = %d\n", *a, *b);
}
```

En el programa anterior, podéis observar la diferencia si en vez de pasar los punteros pasáis los valores, es decir, si declaráis la función así:

```
void exchange(int a, int b);
```

Y, por fin, uno para escribir un fichero:

```
# include <stdio.h>

void main()
{
FILE *fp;
int i;

   /* open foo.dat for writing */
   fp = fopen("foo.dat", "w");

   /* write some info */
   fprintf(fp, "\nSample Code\n\n");
   for (i = 1; i <= 10 ; i++)
      fprintf(fp, "i = %d\n", i);

   /* close the file  */
   fclose(fp);
}
```