PRÁCTICA: Creación de aplicaciones en C (I)

Objetivos:

-Manejo del Lenguaje C

-Uso de funciones de librería y ficheros cabecera

-Etapas y uso del compilador: gcc

-Manejo de ficheros usando la librería estándar de C

-Crear librerías dinámicas y estáticas

-Gestionar los proyectos con make

Sabe instalar lo necesario para programar en c y sus librerías

Instala o comprueba que están instalados los siguientes paquetes o aplicaciones:

Compilador, librerías y documentación necesaria para programar en C:

sudo apt-get update

sudo apt-get install build-essential // Con este paquete se instalan los paquetes esenciales para programar en C

Además de ayuda y documentación extra:
sudo apt-get install debian-keyring //GnuPG keys of Debian Developers
sudo apt-get install gcc-9.3-doc //Documentation for the GNU compilers (gcc, gobjc, g++)
sudo apt-get install glibc-doc //Embedded GNU C Library: Documentation
sudo apt-get install manpages-dev //Manual pages about using GNU/Linux for development

Instala la ayuda y documentación de la APIs de POSIX:
sudo apt-get install manpages-posix //Manual pages about using POSIX system

POSIX(**POSIX** es el acrónimo de **P**ortable **O**perating **S**ystem **I**nterface, y **X** viene de UNIX como seña de identidad de la API.), norma que define una interfaz estándar del sistema operativo y el entorno, incluyendo un intérprete de comandos (o "shell"), y programas de utilidades comunes para apoyar la portabilidad de las aplicaciones a nivel de código fuente. El nombre POSIX surgió de la recomendación de Richard Stallman, que por aquel entonces en la década de 1980 formaba parte del comité de IEEE.

Sabe usar la ayuda a la programación

Utilización del man:

A partir de ahora vamos a utilizar comandos del "*bash*", funciones de librería de C también conocidas como las "API" de "C", funciones de librería del SO Linux también conocidas como las API-s de Linux. Para obtener la ayuda correcta dentro del man, vamos a analizar las secciones de la ayuda del man:

Nº de Sección - Tema

1-.User commands o comandos del bash

2-.System calls o funciones del SO Linux también conocidas como las API-s de Linux

3-.Subroutines o funciones de librería de C también conocidas como las “API” de “C”

4-.Devices

5-.File Formats

6-.Games

7-.Miscellanous

8-.System Administration

9-.New

Uso: man [opciones] [sección] <nombre>

Ejemplos:

$ man 1 write

$ man 2 write
$ man chmod
$ man 5 shadow
$ man -a passwd # Aparece información de todas las secciones

Sabe usar las herramientas de programación

Editor de C:

Gedit

Ir a Ver->Modo resaltado->Fuentes->C

Sabe dónde están los ficheros cabeceras y librerías en linux

1.- ¿Qué contiene el fichero stdio.h?
Contiene las cabeceras (headers) de las funciones de la biblioteca standart imput output.
less /usr/include/stdio.h

2.- Localiza el prototipo/declaración/cabecera de la función printf en /usr/include/stdio.h
cd /usr/include/stdio.h
grep "printf" stdio.h

3.- ¿Dónde están las librerías o bibliotecas libc.so y libc.a estándar de C en linux?
xabierland@ubuntu:/$ sudo find / -name "libc.a"
/usr/lib/x86_64-linux-gnu/libc.a

xabierland@ubuntu:/$ sudo find / -name "libc.so"
/usr/lib/x86_64-linux-gnu/libc.so

TAMBIEN
whereis libc.a

4.- Para más información échale un vistazo a la documentación de la librería estándar de C.
http://www.gnu.org/software/libc/manual/html_mono/libc.html

Sabe manejarse entre los ficheros intermedios de las diferentes etapas de compilación

3.- Crea un programa fuente “pi.c” que contenga el código siguiente:

```
#include <stdio.h>
#define PI 3.1415
int main ()
{
char c;
printf("¿Quieres conocer al número PI? (S/N)");
c=getchar();
if (c=='S' || c=='s') printf("%f\n",PI);
else
printf("Agur\n");
return 0;
}
```

a) Preprocesa el programa pi.c y analiza el código.

gcc -o pi.i -E pi.c

b) Obtén pi.s (ensamblado).

gcc -o pi.s -S pi.i

c) Obtén pi.o (objeto).

gcc -o pi.o -c pi.s

d) Obtén el ejecutable pi a partir del fuente.

gcc -o pi pi.c

Sabe abrir un fichero en modo lectura

4.- Realiza un programa en C que preguntando al usuario el nombre de un fichero calcule su tamaño en bytes.

Ejemplo:

$ **gcc -o longFich longFich.c**

$ **./longFich**

¿De qué fichero deseas conocer el tamaño?longFich.c

El tamaño del fichero es de 483

$ **ls -l longFich.c**

-rw-r--r-- 1 kepa kepa **483** 2013-03-01 12:36 longFich.c

```
#include <stdio.h>

int main()
{
        char dir[255];
        char c;
        printf("Introduce la direccion del fichero\n");
        scanf("%s",dir);
        FILE * fp = fopen(dir,"r");
        int i=0;
        while(1)
        {
                c=fgetc(fp);
                if(feof(fp))
                        break;
                i++;
        }
```

```
	printf("El numero de Bytes es: %i\n", i);

	//END
	return 0;
}
```

Sabe abrir un fichero en modo escritura

5.- Realiza un programa en C que preguntando al usuario por el nombre de un fichero escriba en este el abecedario tanto en minúsculas como en mayúsculas, cada uno en una fila.

Ejemplo:
$ **gcc -o abece abece.c**
$ **./abece**
Introduce el nombre del fichero a crear: prueba.txt
$ **less prueba.txt**
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ

```
#include <stdio.h>

int main()
{
	char dir[255];  //DIRECCION DEL ARCHIVO
	printf("Introduce la direccion del fichero:\t");
	fflush(stdin);
	scanf("%s",dir);

	FILE * fp = fopen(dir,"w");
	char c;
	c='a';
	while(c<='z')
	{
		fputc(c,fp);
		c++;
	}
	fputc('\n',fp);
	c='A';
	while(c<='Z')
	{
		fputc(c,fp);
		c++;
	}
	fputc('\n',fp);
	fclose(fp);
	return 0;
}
```

Sabe reutilizar los ejecutables en otros programas:

Realiza un programa por lotes o script del Bash que ejecute los dos ejecutables realizados en los ejercicios anteriores. Si el primer ejecutable se ejecuta correctamente que se ejecute el segundo. Para ello si el ejecutable “abece” termina correctamente, ejecute “longFich”, y si no muestre un mensaje de que el primer programa ha fallado. El programa se detalla abajo. Comprueba su uso correcto.

gedit bashscript.sh

./**abece**

```
if [ $? -eq 0 ] ; then
   ./longFich
   echo -e "\nwell done"
   exit 0
else
   echo -e "\nfailed"
   exit 1
fi
```

chmod u+x ./bashscript.sh

./bashscript.sh

He entendido el uso del $? y la importancia de hacer el return 0 en los programas de 0 para ver si la ejecución es correcta.

Sabe gestionar ficheros usando librerías propias

1.-El siguiente programa principal “numeroaes.c” está incompleto:

```
#include <stdio.h>
int main()
{
char nombre[80],c;
FILE *fp;
int contador=0;
printf ("¿De qué fichero deseas conocer el número de aes?");
scanf("%s",nombre);
fp=fopen(nombre,"r");
if (fp==NULL)
      {
         printf("No se puede abrir el fichero %s \n",nombre);
```

```
        return -1;
    }
else
    {    contador=ffichnumcarac('a',fp);
         fclose(fp);
         printf("El número de a-s es de %d",contador);
    }
return 0;
}
```

Se pide:

a)Completa la función “ffichnumcarac” dentro de “numeroaes.c”. A esta función se le pasarán como parámetros el descriptor de fichero y el carácter a buscar. Y nos devolverá el número de veces que aparece dicho carácter en el fichero. Ejecuta dicho programa principal *numeroaes*.

numeroaes.c

```
#include <stdio.h>

int ffichnumcarac(char c, FILE *fp)
{
        int i=0;
        while(1)
        {
                if(c==fgetc(fp))
                {
                        i++;
                }
                if(feof(fp))
                {
                        break;
                }
        }
        return i;
}
```

```
int main()
{
        char nombre[80],c;
        FILE *fp;
        int contador=0;
        printf ("¿De qué fichero deseas conocer el número de aes?\n");
        scanf("%s",nombre);
        fp=fopen(nombre,"r");
        if (fp==NULL)
    {
                printf("No se puede abrir el fichero %s \n",nombre);
          return -1;
     }
        else
    {
                contador=ffichnumcarac('a',fp);
          fclose(fp);
          printf("El número de a-s es de %d\n",contador);
    }
        return 0;
}
```

b) Ahora introduce dicha función “ffichnumcarac” en un módulo que se llamará “general.c”. Finalmente introduce dicho módulo en una biblioteca dinámica cuyo nombre será “libgeneral.so”. Y crea el ejecutable *numeroaes* usando dicha librería dinámica. Para ello modifica el programa principal para que la compilación tenga éxito.

```
gcc -c general.c -o general.o -fPIC

ld -o libgeneral.so general.o -shared

gcc -o numeroaes2 numeroaes.c -I. -L. -lgeneral -Bdynamic

LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/xabierland/Documents/IOS/LAB3/numeros/
```

export LD_LIBRARY_PATH

numerosaes.c

```
#include <stdio.h>
#include "general.h"

int main()
{
        char nombre[80],c;
        FILE *fp;
        int contador=0;
        printf ("¿De qué fichero deseas conocer el número de aes?\n");
        scanf("%s",nombre);
        fp=fopen(nombre,"r");
        if (fp==NULL)
    {
                printf("No se puede abrir el fichero %s \n",nombre);
          return -1;
     }
        else
    {
                contador=ffichnumcarac('a',fp);
          fclose(fp);
          printf("El número de a-s es de %d\n",contador);
    }
        return 0;
}
```

general.c

```
#include <stdio.h>

int ffichnumcarac(char c, FILE *fp)
{
        int i=0;
```

```
        while(1)
        {
                if(c==fgetc(fp))
                {
                        i++;
                }
                if(feof(fp))
                {
                        break;
                }
        }
        return i;
}
```

general.h

```
#ifndef _LIBGENERAL_SO
#define _LIBGENERAL_SO
int ffichnumcarac(char c, FILE *fp);
#endif
```

Ejecutar comando de shell desde C

¿Para qué sirve la función system?. Ejecuta el siguiente código:

```
#include<stdio.h>
int                                                    main(){
system("ls                                                -l");
return                                                    0;
}
```

System sirve para ejecutar comandos de Bash desde C como si los estuviéramos escribiendo en la terminal.