Sesión 02: Ejecutando C

Programación para Sistemas

Ángel Herranz

2022-2023

Universidad Politécnica de Madrid

Recordatorio

- ¿Cómo van esas instalaciones de Ubuntu?
- ¿Cómo va Bash? (**`pwd`**, `ls`, **`cd`**, `mkdir`, `...`)
- ¿Y C? (`gcc`, `./a.out`)
- ¿Cómo va el repaso de las transparencias?
- ¿Cómo van las hojas de ejercicios?
- ¿Cómo van esos accesos a `triqui`? (`ssh`)

En clase

- *Learning by doing*.

En clase

- *Learning by doing*.
- *But, not yet*.
- Por ahora... *learning by copying and listening*.

⚠ leed con cuidado cada una de las líneas que aparecen en la pantalla,

cualquier cosa que no se entienda, duda, sugerencia, prueba que querais hacer, ...

En clase

- *Learning by doing*.
- *But, not yet*.
- Por ahora... *learning by copying and listening*.

⚠ leed con cuidado cada una de las líneas que aparecen en la pantalla,

cualquier cosa que no se entienda, duda,

sugerencia, prueba que querais hacer, ...

¡Paradme!

Hola aherranz

- Comunicándonos con nuestro programa: variables de entorno

[1] Dónde aherranz es el nombre de usuario en la variable de entorno USER.

Hola aherranz

- Comunicándonos con nuestro programa: variables de entorno

Escribir un programa en C (`hola.c`) que saque por la salida estándar `"Hola NOMBRE"`, donde `NOMBRE` va a ser un argumento de la línea de comandos. que hacer esto:

```
$ ./hola ángel
Hola ángel
```

Si el programa se invoca sin argumentos:[1]

```
$ ./hola
Hola aherranz
```

[1] Dónde `aherranz` es el nombre de usuario en la variable de entorno `USER`.

Hola aherranz

- Comunicándonos con nuestro programa: variables de entorno

Escribir un programa en C (`hola.c`) que saque por la salida estándar `"Hola NOMBRE"`, donde `NOMBRE` va a ser un argumento de la línea de comandos. que hacer esto:

```
$ ./hola ángel
Hola ángel
```

Si el programa se invoca sin argumentos:[1]

```
$ ./hola
Hola aherranz
```

[1] Dónde `aherranz` es el nombre de usuario en la variable de entorno `USER`.

Hola aherranz (solución): variables de entorno

- Ya hemos visto una forma de comunicarnos con nuestros programas: argumentos en línea de comandos
- Una segunda forma es utilizando *variables de entorno*
- Son *variables* que pueden ser establecidas como entorno de ejecución del programa antes de ejecutar el programa y que el programa puede accederlas
- Algunas variables interesantes ya están definidas, para consultarlas en Bash: `printenv`
- Ej. `printenv USER`

Hola aherranz (solución): `man getenv`

```
GETENV(3)          Linux Programmer's Manual          GETENV(3)

NAME
  getenv, secure_getenv - get an environment variable

SYNOPSIS
  #include <stdlib.h>

  char *getenv(const char *name);

DESCRIPTION
  The getenv() function searches the environment list to
  find the environment  variable name, and returns a
  pointer to the corresponding value string.
```

Hola aherranz (solución)

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
  char *quien;
  if (argc == 1) {
    quien = "mundo"; /* TODO: usar la variable USER */
  }
  else {
    quien = argv[1];
  }
  printf("¡Hola %s\n", quien);
  return 0;
}
```

Factorial i

- Hemos aprendido una de las formas más básicas para comunicarnos con nuestro programa:

  ¡pasarle datos al ponerlo en marcha!

- Escribir un programa que imprima en la salida estándar el factorial de un número que se le pasa desde la linea de comandos. Ejemplos de uso:

```
$ ./fact 1
1
$ ./fact 2
2
$ ./fact 3
6
```

```
$ ./fact 4
24
$ ./fact 5
120
$ ./fact 6
720
```

Factorial ii

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

int factorial(int n) {
  int f = 1;
  int i;
  for(i = 1; i <= n; i++) {
    f *= n;
  }
  return f;
}

int main(int argc, char *argv[]) {
  int n;
  if (argc == 1)
    n = 0;
  else
    n = atoi(argv[1]);
```

(sigue)

```c
  if (n = 0) {
    fprintf(
      stderr,
      "Uso: fact N (N > 0)\n"
    );
    return 1;
  }
  else {
    printf("%i\n",
           factorial(n));
    return 0;
  }
}
```

Factorial ii

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

int factorial(int n) {
  int f = 1;
  int i;
  for(i = 1; i <= n; i++) {
    f *= n;
  }
  return f;
}

int main(int argc, char *argv[]) {
  int n;
  if (argc == 1)
    n = 0;
  else
    n = atoi(argv[1]);
```

(sigue)

```c
  if (n = 0) {
    fprintf(
      stderr,
      "Uso: fact N (N > 0)\n"
    );
    return 1;
  }
  else {
    printf("%i\n",
           factorial(n));
    return 0;
  }
}
```

- Compilar y ejecutar los ejemplos de uso

  5’

¿¡*Bugs*!?

¡*Bugs*! (diario con polilla muerta en el Harvard Mark II)

0800 Arctan started
1000 " stopped - arctan ✓ { 1.2700 9.037 847 025
9.037 846 795 correct
13"u/c (032) MP - MC ~~1.9821447000~~ ~~2.1304764915~~ 4.615925059(-2)
(033) PRO 2 2.130476415
correct 2.130676415
Relays 6-2 in 033 failed special speed test
In relay " 10.00 test.
Relays changed
1100 Started Cosine Tape (Sine check)
1525 Started Mult + Adder Test.

1545 Relay #70 Panel F
(moth) in relay.
First actual case of bug being found.
1700 closed down.

Documentado por Grace Murray Hopper
9 de septiembre de 1947

Depurador GDB

- *GNU Debugger*
- Un depurador sirve para depurar ;)
- Pero para nosotros, en esta asignatura

  sirve para ver cómo se ejecuta mi programa.

Modelo Computacional

- Ejecución paso a paso
- Paquete `gdb` en Ubuntu:

```
$ sudo apt-get install gdb
```

GDB *crash course*

Instalación de GDB

- El depurador `gdb` suele *venir* con `gcc`
- En Ubuntu está en el paquete `gdb`[2]:

```
$ sudo apt-get install gdb
```

[2] Casualidad que se llamen igual.

Compilación y arranque de GDB

- Para poder luego usar GDB, compilar con argumento `-g`:

```
$ gcc -g -o fact fact.c
$ ./fact 5
1
$ gdb fact
GNU gdb (Ubuntu 8.1-0ubuntu3) 8.1.0.20180409-git
...
Reading symbols from fact...done.
(gdb)
```

- Depurador en marcha, esperando órdenes.
- El *prompt* ha cambiado a (gdb).

Ejecución *bajo* GDB i

- Para ejecutar el programa cargado (`./fact`) usamos la orden run:

```
(gdb) run
Starting program: /home/angel/...
1
[Inferior 1 (process 21938) exited normally]
(gdb) |
```

- Vemos que el resultado es sacar 1 en la salida estándar.

Ejecución desde GDB ii

- Para ejecutar el programa cargado pasándole `5` como parámetro:

```
(gdb) run 5
Starting program: /home/angel/...
1
[Inferior 1 (process 21938) exited normally]
(gdb) |
```

- Vemos que el resultado es `1` en vez de `120`: algo va mal.

Breakpoints

- Podemos pedir a GDB que ponga un *breakpoint* para que se pare cuando la ejecución llegue a ese punto.
- Usamos la orden break para que GDB se pare cuando comienze a ejecutar la función `main`:

```
(gdb) break main
Breakpoint 1 at 0x5555555475b: file fact.c, line 16.
(gdb) run 5
Starting program: /home/angel/Asignaturas/pps/02/fact 5

Breakpoint 1, main (argc=2, argv=0x7fffffffe088) at ...
16        if (argc == 1) {
(gdb) |
```

- GDB informa de la siguiente linea que va a ejecutar.

Ejecución desde GDB paso a paso i

- En cada paso se puede explorar el valor de las variables y argumentos con la orden print:

```
(gdb) print argc
$1 = 2
(gdb) print argv[0]
$2 = 0x7fffffffe402 "/home/angel/Asignaturas/pps/02/fact"
(gdb) print argv[1]
$3 = 0x7fffffffe433 "5"
(gdb) print n
$4 = 0
```

Ejecución desde GDB paso a paso ii

- A partir del *breakpoint* podemos pedir a GDB que ejecute el programa paso a paso con la orden step:

```
(gdb) step
20          n = atoi(argv[1]);
(gdb) step
atoi (nptr=0x7fffffffe462 "5") at atoi.c:26
26      atoi.c: No such file or directory.
(gdb) |
```

- GDB ha intentado entrar en la implementación de `atoi` pero no tiene disponible el código fuente.
- Aún así, no falla, la ejecución paso a paso continúa.

Ejecución desde GDB paso a paso iii

- Con la orden finish vamos a pedir a GDB que directamente termine de ejecutar la función (atoi):

```
(gdb) finish
Run till exit from #0  atoi (nptr=0x7fffffffe462 ...
0x000055555555477d in main (argc=2, argv=0x7fffff...
20          n = atoi(argv[1]);
Value returned is $5 = 5
(gdb) |
```

Ejecución desde GDB paso a paso iv

```
(gdb) step
22          if (n = 0) {
(gdb) print n
$6 = 5
(gdb) step
27              printf("%i\n", factorial(n));
(gdb) print n
$7 = 0
```

- ¿Despues de ejecutar "`if (n = 0) {`" el valor de `n` es `0`?

No es lo mismo == que =

Ejecución desde GDB paso a paso v

- Trucos: r en vez de run, b en vez de break, etc.
- Truco: Enter para ejecutar la última orden de nuevo.

Reproducir la ejecución bajo el depurador y continuar hasta encontrar el otro bug.

15'

Algunas referencias a GDB

- Visitar https://www.gnu.org/software/gdb/
- *Debugging with GDB. The GNU Source-Level Debugger* (Richard Stallman, Roland Pesch, Stan Shebs, et al.):
  - `https://docs.freebsd.org/info/gdb/gdb.pdf` (v4)
  - `http://tierra.aslab.upm.es/~sanz/old/cursos/C1/gdb.pdf` (v9)
- Introduction to GDB a tutorial - Harvard CS50