Práctica 2 del Laboratorio de Sistemas Operativos

Departamento de Automática
Universidad de Alcalá

/gso>

Fases de desarrollo

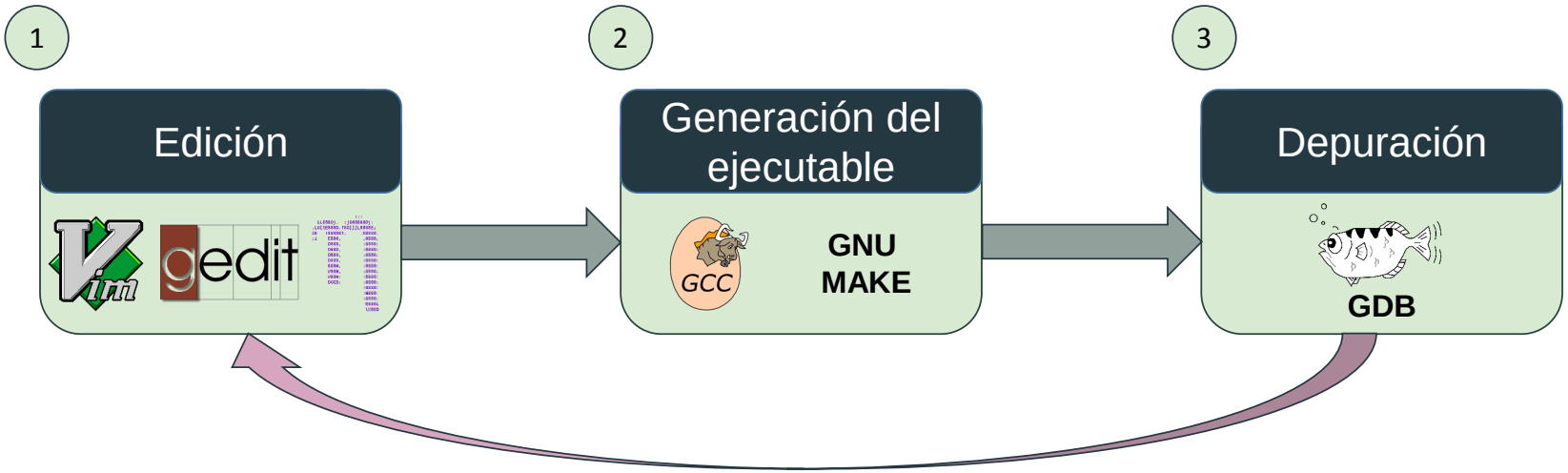
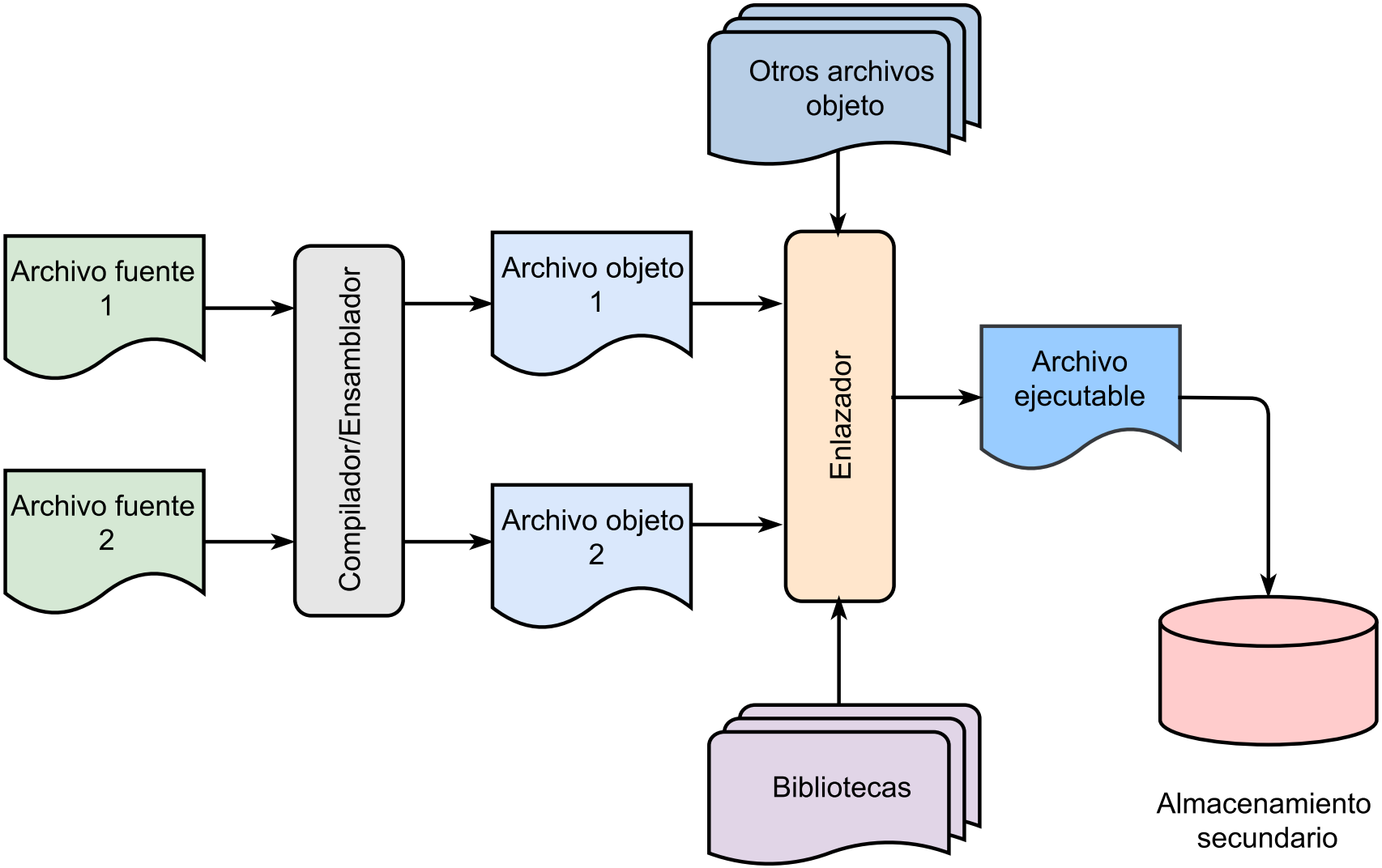


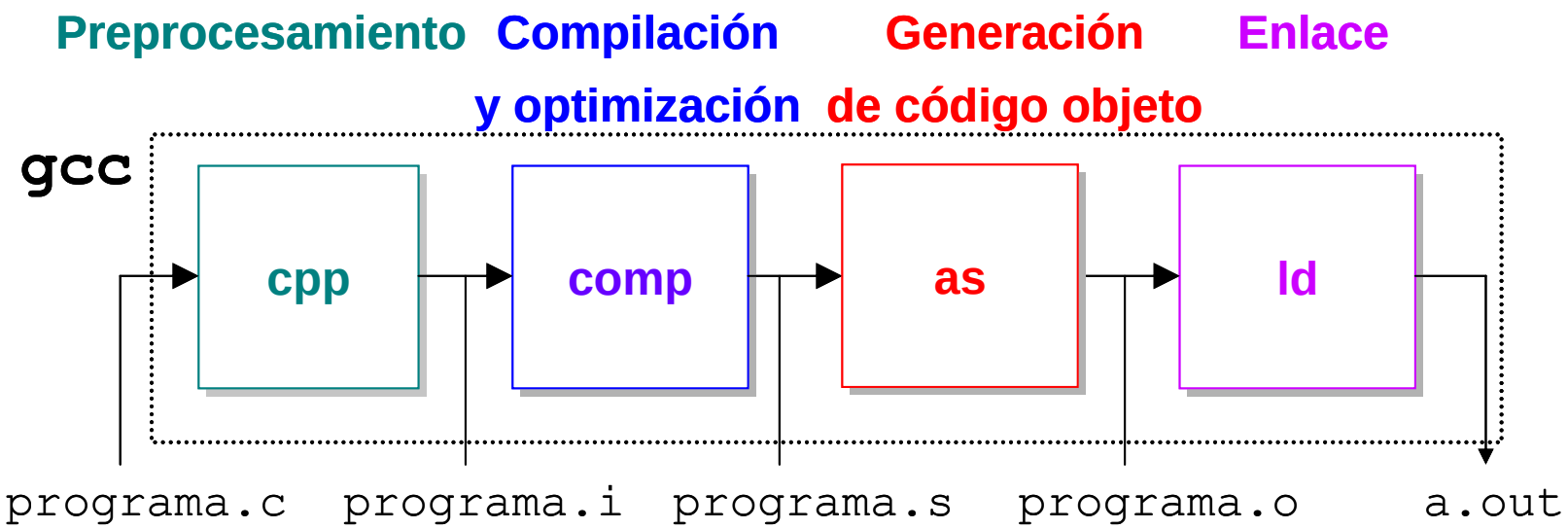
Figura: Fases de desarrollo del ciclo de creación de un programa.

Generación del ejecutable



Generación del ejecutable

Conjunto de herramientas GCC



- La orden `gcc` tiene opciones para generar los archivos intermedios (`-E`, `-S`, `-c`).
- Otras opciones de gcc: `-o`, `-Wall`, `-g`

Generación del ejecutable

Ejemplo de programa

principal.c

```
#include <stdio.h>

int sumar(int, int);

int main(){
   int a = 5, b = 10, c;
   c = sumar(a, b);
   printf("Resultado suma: % d\n", c);
   return 0;
}

int sumar(int sumando1, int sumando2){
   return sumando1 + sumando2;
}
```

sumar.h

```
int sumar(int, int);
```

sumar.c

```
int sumar(int sumando1, int
      sumando2){
   return sumando1 + sumando2;
}
```

- Proceso de generación del ejecutable programa:
  1. `gcc -g -Wall -c principal.c`
  2. `gcc -g -Wall -c sumar.c`
  3. `gcc -g -Wall principal.o sumar.o -o programa`

Generación del ejecutable

Posibles avisos y errores al compilar con `gcc -Wall`

unused variable ‘`variable`’
: Indica que `variable` se reserva y no se utiliza en todo el programa.

implicit declaration of function ‘`funcion`’
: Indica que no existe declaración previa a la llamada de `funcion`. Puede ser por varias causas, normalmente suele ser por la falta del archivo de cabecera correspondiente.

return type defaults to ‘int’
: Indica que la función definida en esa línea no tiene explícitamente marcado el tipo de los datos que devuelve y se le asigna, por defecto, tipo entero.

suggest parentheses around assignment used as truth value
: Indica que se está realizando una asignación y una operación de comparación en la misma expresión o que se está realizando una asignación por error en lugar de una comparación, sugiriendo la utilización de paréntesis para asegurar que el orden de las dos operaciones es correcto.

Generación del ejecutable

Posibles avisos y errores al compilar con `gcc -Wall`

return type of 'main' is not 'int'
: Indica que el tipo devuelto por la función `main` no es un entero.

'return' with a value, in function returning void
: indica que se está devolviendo un valor desde una función declarada como `void`.

control reaches end of non-void function
: Indica que se ha alcanzado el final de una función no declarada como `void` sin que se devuelva ningún valor.

assignment makes integer from pointer without a cast
: Indica que se está realizando una asignación entre valores de diferentes tipos sin utilizar conversiones cast.

Make: Definición y características

Make

Herramienta que permite automatizar la gestión de las órdenes de compilación y facilita la tarea de creación de ejecutables.

- Permite definir **una sola vez** las opciones de compilación de los módulos que forman parte del programa.
- Es capaz de llevar un **control de los cambios** que se han realizado en los archivos fuente y ejecutables.
- Se evita la recompilación de los módulos del programa que no hayan sido modificados y, por lo tanto, se optimiza el proceso de compilación.

Archivo `Makefile` (I): Definición y características

Makefile

Archivo de texto que utiliza *make* para llevar a cabo la gestión de la compilación de programas. Este archivo contiene, en forma de reglas, las dependencias entre los diferentes archivos de un proyecto.

- Por defecto, la herramienta *make* busca un archivo `makefile` o `Makefile` en el directorio actual.
- Si se desea modificar este nombre, *make* deberá invocarse con el parámetro -f seguido del nombre de archivo.
- Las reglas de un archivo `makefile` se ejecutan mediante una especie de *backtracking*. La primera vez recorre las reglas hacia abajo y a continuación las resuelve de abajo hacia arriba.

Archivo `Makefile` (II): Sintaxis y semántica

Fragmento de código 1: Sintaxis de un `makefile`.

```
objetivo: dependencia_1 ... dependencia_n
<TAB> orden_1
<TAB> :::
<TAB> orden_n
```

- **Objetivo**: Normalmente, es un archivo (o varios) que se quiere(n) crear o actualizar.
- **Dependencias**: Son nombres de archivos u otros objetivos necesarios para actualizar el objetivo de la regla.
- **orden_i**: Orden necesaria para reconstruir el objetivo de la regla. Puede ser cualquier orden de la *shell* o intérprete de órdenes. Debe comenzar necesariamente por el carácter <TAB> (tabulador).

Archivo `Makefile` (III): Reglas ficticias

Reglas ficticias

Reglas que no generan un archivo específico sino que se utilizan para realizar una determinada tarea dentro del desarrollo de la aplicación.

Para evitar confusiones entre el objetivo ficticio clean y un posible archivo que pueda existir con el mismo nombre, se suele utilizar el objetivo especial de tipo `PHONY`.

Fragmento de código 2: Sintaxis de un `makefile`.

```
.PHONY: clean
clean:
<TAB> rm -f archej *.o
```

Archivo `Makefile` (IV): Variables

- Las variables facilitan la portabilidad a diferentes plataformas y entornos así como la modificación del archivo *makefile*.
- Las variables se suelen escribir en su totalidad en mayúsculas.
- Para obtener el valor de una variable, se pone el signo '$' y el nombre de la variable entre paréntesis o llaves.

Fragmento de código 3: Sintaxis de un `makefile`.

```
CC = gcc
CFLAGS = -g -Wall

arch2.o: arch2.c arch1.h
	$(CC) $(CFLAGS) -c arch2.c
```

Archivo `Makefile` (V): consejos para hacer un correcto `makefile`

- ¿Expresiones repetidas frecuentemente? ¡**Usa variables**!
- Todo `makefile` comienza siempre por el objetivo principal.
- ¿Tiene el objetivo todas sus dependencias? ¡**Revisa las relaciones entre los archivos**!

Archivo `Makefile` (VI): consejos para hacer un correcto `makefile`

- Recuerda incluir los archivos de cabecera .h de los archivos creados cuando se vaya a crear la regla relacionada con el archivo objeto (.o).

- No olvides el *clean*. Es tan importante como el resto de reglas.

Archivo `Makefile` (VII): cómo **SI** hacer un `makefile`

Fragmento de código 4: Ejemplo de un `makefile`.

```
CC = gcc
CFLAGS = -g -Wall

saludos: saludos.o
	$(CC) $(CFLAGS) saludos.o -o saludos

saludos.o: saludos.c saludos.h
	$(CC) $(CFLAGS) -c saludos.c

.PHONY: clean
clean:
	rm -f saludos *.o
```

Archivo `Makefile` (VIII): cómo **NO** hacer un `makefile`

Errores típicos

- Escribir una única regla.
- No emplear las dependencias en las órdenes.

Fragmento de código 5: Ejemplo de un `makefile` INCORRECTO.

```
CC = gcc
CFLAGS = -g -Wall

saludos: saludos.o bienvenida.o despedida.o
	$(CC) $(CFLAGS) saludos.c bienvenida.c despedida.c -o saludos
```

GDB: Definición y características

GDB

Herramienta estándar de GNU que permite depurar un programa para mejorar su calidad.

- Es posible alterar y monitorizar los valores de las variables internas del programa.
- Es posible establecer puntos de ruptura para verificar el estado de ejecución del programa en un instante determinado.
- Permite encontrar fallos durante la ejecución del programa.

GDB: Definición y características

Ejecución

`run`	comenzar la ejecución del programa
`set args <argumentos>`	establecer los argumentos del programa
`quit`	salir del depurador

Puntos de ruptura

`break <position>`	establecer un punto de ruptura en una posición del código
`delete <# breakpoint>`	borrar el punto de ruptura #
`clear`	ir a la siguiente ocurrencia

Ejecución paso a paso

`step`	ejecutar la siguiente línea de código entrando en las funciones
`next`	ejecutar la siguiente línea de código sin entrar en las funciones
`finish`	continuar hasta la salir de la función actual
`continue`	continuar la ejecución del programa

Variables y memoria

`print/`*`format`* `<item>`	imprimir un valor con formato
`x/`*`nfu`* `<address>`	volcado de *n* posiciones de memoria con formato

`<item>`	expresión (e.g. nombre de variable) dirección de memoria `$registro` (e.g. `$ax`)

`format`	**Formato**
a	puntero
d	decimal con signo
u	decimal sin signo
f	coma flotante
x	hexadecimal

u	**Unidades**
b	byte
h	media palabra (dos bytes)
w	palabra (cuatro bytes)
g	palabra gigante (ocho bytes)

Depuración con GDB

ejemplo_gdb.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  int imprimir = 1;
6  char *nomprograma;
7
8  int main(int argc, char *argv[])
9  {
10     int i;
11     char *ptr;
12     nomprograma = argv[0];
13
14     printf("Numero de argumentos = %d\n", argc);
15     printf("Nombre del programa: %s\n", nomprograma);
```

Depuración con GDB

`ejemplo_gdb.c` *(cont.)*

```
16
17      for (i = 1; i <= argc; i++)
18      {
19          ptr = malloc(strlen(argv[i])+1);
20          strcpy(ptr, argv[i]);
21
22          if (imprimir)
23              printf(" %s\n", ptr);
24
25          free(ptr);
26      }
27
28       return 0;
29  } /* fin main */
```

Compilar, ejecutar y depurar `ejemplo_gdb.c` (I)

1. Compile el programa `ejemplo_gdb.c`.
2. Ejecute el programa `ejemplo_gdb.c` pasándole como parámetro `-l`. ¿Qué sucede?
3. Compile el programa `ejemplo_gdb.c` añadiendo el parámetro necesario para poder depurarlo.
4. Inicie el depurador con el programa `ejemplo_gdb`.
5. Obtenga ayuda sobre las órdenes del depurador.
6. Obtenga las funciones utilizadas por `ejemplo_gdb`.
7. Pase el parámetro `-l` al programa `ejemplo_gdb`.
8. Ejecute el programa `ejemplo_gdb` ¿Qué sucede?
9. Establezca tres puntos de ruptura: *líneas 8* (`main`), *17* y *22*, respectivamente.

NOTA: Establecer adecuadamente los puntos de ruptura es clave para acotar los errores de ejecución que se hayan producido y eliminarlos.

Compilar, ejecutar y depurar `ejemplo_gdb.c` (II)

10. Ejecute `ejemplo_gdb` hasta el primer punto de ruptura.
11. Ejecutar la siguiente línea del programa (*línea 12*).
12. Ejecute las dos siguientes funciones (`printf`) **sin** entrar en su código.
13. Ejecute la siguiente línea del programa (`for`).
14. Muestre el valor de la variable `i`.
15. Ejecute hasta el siguiente punto de ruptura (*línea 22*).
16. Ejecute hasta llegar a la segunda iteración del `for`. Si la línea es una función, ejecútela **sin** entrar en su código.
17. Ejecute la línea del `for` (*línea 17*).
18. Imprima el valor de la variable `i` y el de `argv[2]`. Observe sus valores...
19. Ejecute la *línea 19* sin entrar en el código de sus funciones. ¿Ha detectado cuál es el error?