Sistemas Operativos II

Makefiles

Universidad Arturo Jauretche

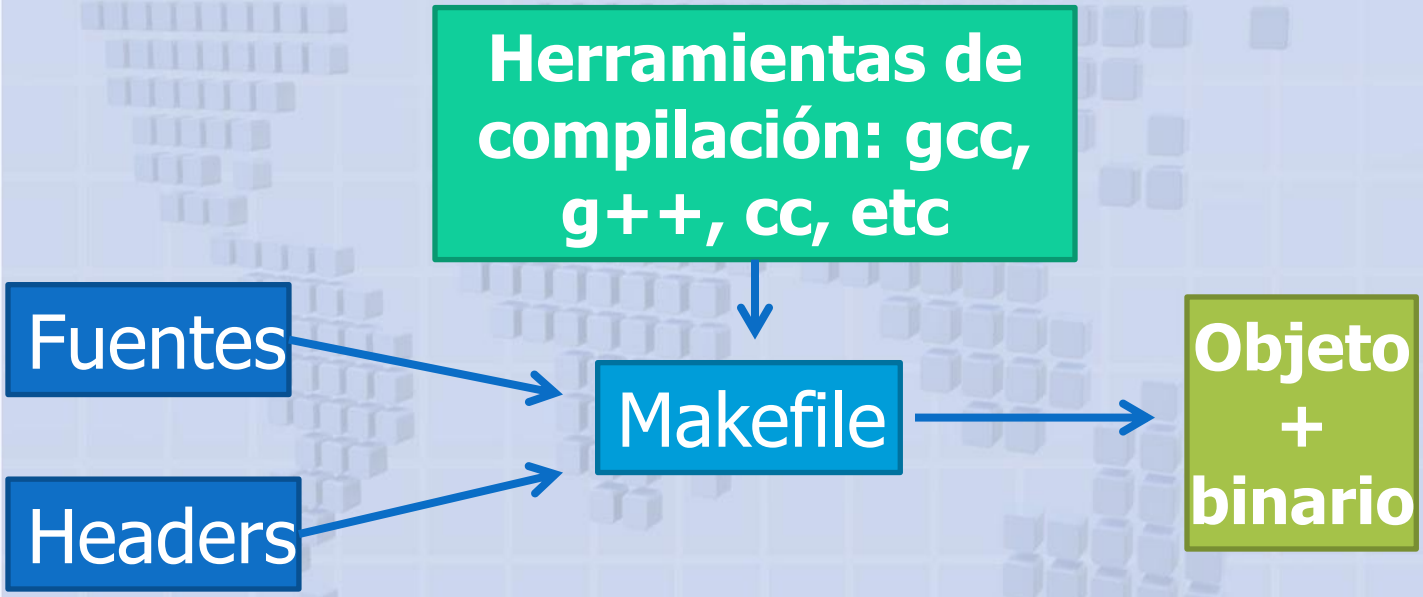
Ingeniería Informática

Docentes:

Ing. Jorge Osio

Ing. Eduardo Kunysz

Makefile


Herramientas de compilación: gcc, g++, cc, etc
Fuentes
Headers
Makefile
Objeto + binario

Makefile

- Sabe que cosas hay que compilar
- Solo se compilan aquellos archivos que hemos modificado
- Útil para proyectos grandes
- Guarda parámetros, librerías, headers (.h)
-Evitamos escribir líneas largas de compilación

Ejemplo de compilación

Crear el archivo HolaMundo.c y escribir el siguiente código:

```
#include <stdio.h>
main()
{
    printf ("Hola mundo\n");
}
```

Líneas de compilación:

```
$ cc HolaMundo.c -o HolaMundo
```

Se generó binario «HolaMundo»

Ejemplo de compilación con make

Si lo eliminamos el archivo HolaMundo y escribimos:

```
$make HolaMundo
```

Vuelve a compilarlo, si volvemos a poner

```
$make HolaMundo
```

Dirá que «no hay nada para hacer»

¡Porque sabe que está compilado!

Variables de entorno

- Make tiene ciertas reglas implícitas, llamadas **variables de entorno** definidas en make.rules

Ej: **CFLGAGS**

Si hacemos:

```
$CFLAGS=-g; export CFLAGS
$make HolaMundo
```

Se le estará pasando al compilador la opción –g (para generar códigos de debug)

Variables de entorno

La opción –I nos permite pasar path para buscar los ficheros:

```
$ CFLAGS=-I../directorio; export CFLAGS
```

Compilará los archivos dentro del directorio «directorio».

Archivo Makefile

Para darle opciones personalizadas al comando make se debe crear un archivo llamado **Makefile**

Formato:

```
objetivo: dependencia1 dependencia2 …
        <tab>comando1
        <tab>comando2
        <tab> …
```

Importante: El Makefile es muy sensible a espacios y tabulaciones.

Archivo Makefile

Objetivo: es lo que queremos construir.

Por ejemplo el nombre del archivo binario.

Dependencia<i>: nombre de otro objetivo que debe hacerse antes del binario, o ficheros de dependencia.

Por ejemplo el archivo fuente

Comando<i>: es lo que se tiene que ejecutar para construir el objetivo. Se ejecutan en secuencia. (cualquier comando de shell)

Archivo Makefile

Ejemplo:

```
HolaMundo: HolaMundo.c
    cc –I../HOLA HolaMundo.c – o
HolaMundo
```

Esta no es la forma optima de hacerlo, pero nos da una idea de como funciona

Con este makefile si tocamos cualquier fichero recompila todo

Makefile mejorado

Para conseguir un funcionamiento correcto hay que tener los archivos objeto ".o"

```
HolaMundo: HolaMundo.o
	cc HolaMundo.o -o HolaMundo

HolaMundo.o: HolaMundo.c
	cc -c -I../directorio/HolaMundo.c -o HolaMundo.o
```

Makefile mejorado

Para conseguir un funcionamiento correcto hay que tener los archivos objeto ".o"

```
HolaMundo: HolaMundo.o
	cc HolaMundo.o -o HolaMundo

HolaMundo.o: HolaMundo.c
	cc -c -I../directorio/HolaMundo.c -o HolaMundo.o
```

1ero

Makefile mejorado

Para conseguir un funcionamiento correcto hay que tener los archivos objeto ".o"

```
HolaMundo: HolaMundo.o
	cc HolaMundo.o -o HolaMundo

HolaMundo.o: HolaMundo.c
	cc -c -I../directorio/HolaMundo.c -o HolaMundo.o
```

2do

Makefile con variables de entorno

```
OBJETOS= HolaMundo.o
FUENTE= HolaMundo.c
CFLAGS= -I../directorio

HolaMundo: $(OBJETOS)
	cc $(OBJETOS) -o HolaMundo

HolaMundo.o: $(FUENTE)
	cc -c $(FUENTE) -o $(OBJETOS)
```

Makefile típico

```
OBJETOS=HolaMundo.o ... Lista de objetos
FUENTE= HolaMundo.c
CFLAGS=-I../tmp
//en este ejemplo los archivos se encuentran en tmp
all: HolaMundo

HolaMundo: $(OBJETOS)
	cc $(OBJETOS) -o HolaMundo

HolaMundo.o: $(FUENTE)
	cc -c $(FUENTE) -o $(OBJETOS)
clean:
	rm *.o HolaMundo
```

Comandos: make : hace todo

make clean: limpia todo

Referencias

www.gnu.org/software/make/manual/make.html