Programming in Linux

by Mohamad H. Danesh

Getting Started

- Developing applications for Linux
  - Using graphical programming environments
  - Using command-line programming environments

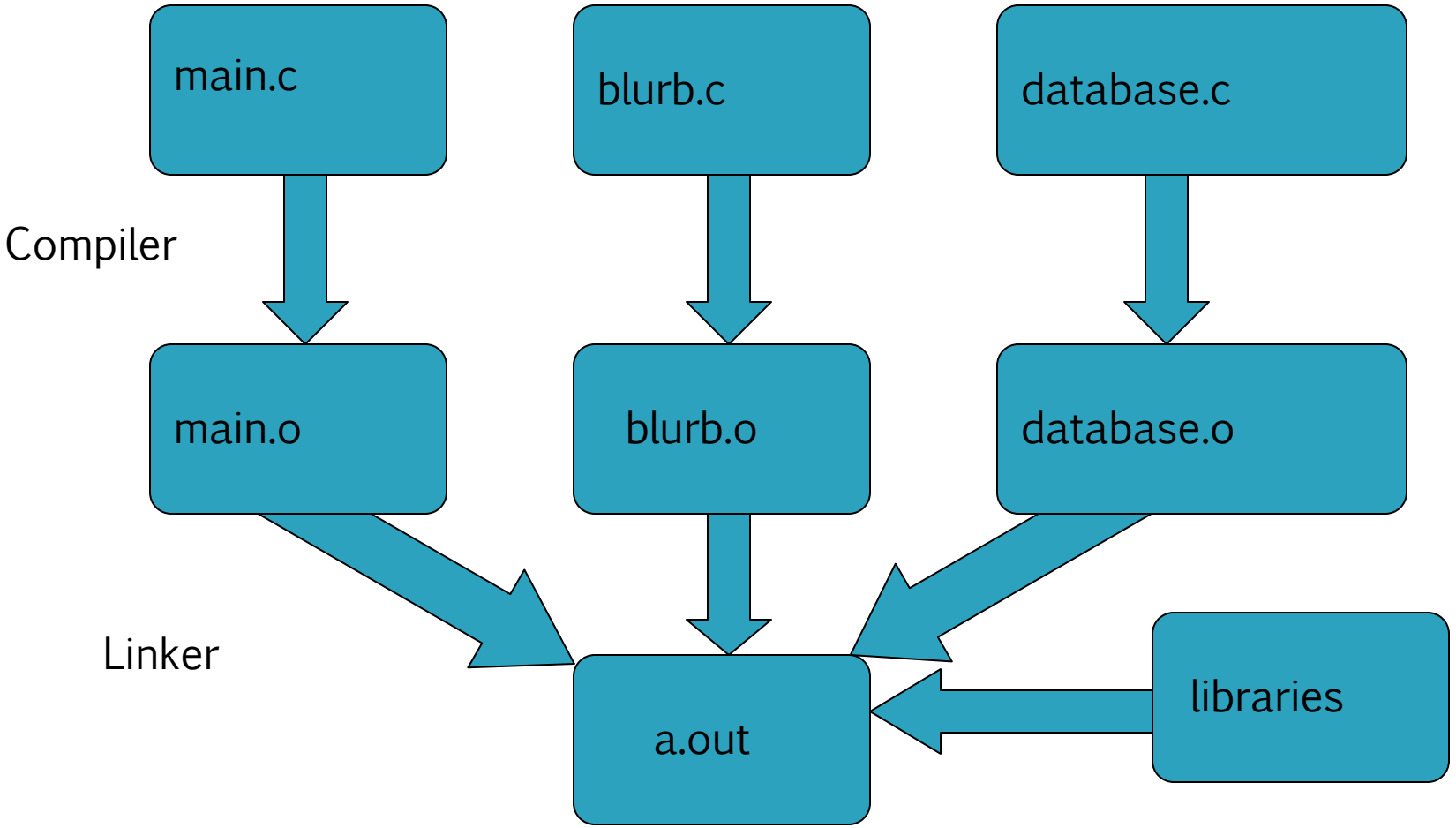
GCC - GNU Compiler Collection

- Frontend for:
  - C: gcc
  - C++: g++
  - More (ada, java, objective-c, fortran, …)
- Backend for:
  - x86, ia-64, ppc, m68k, alpha, hppa, mips, sparc, mmix, pdp-11, vax, …

GCC Compiler

- What happens when you call gcc to build your program?
- Phase 1, Compilation: .c files are compiled into .o object modules
- Phase 2, Linking: .o modules are linked together to create a single executable.

Process


main.c
blurb.c
database.c
Compiler
main.o
blurb.o
database.o
Linker
a.out
libraries

Gcc Compilation Options

- **"-c":** Compiles .c file arguments to .o but does not link (we'll need this for "make" later).
- **"-g":** Generates debugging information that is used by gdb-based debuggers
- **"-I<dir>":** Adds the directory <dir> to the list of directories searched for include files.
- **"-w":** Inhibit all warning messages.

Gcc Linking Options

- **"-o":** Specifies the name of the program to be linked together. Name textr "textr!"
- **"-L<dir>":** Adds the directory <dir> to the list of directories searched for library files

Example

```
/*
* main.c driver program
*/
#include <stdio.h>
#include "msg.h"

int main(int argc, char *argv[])
{
  char msg_hi[] = { "Hi there, programmer!" };
  char msg_bye[] = { "Goodbye, programmer!" };
  printf("%s\n", msg_hi);
  prmsg(msg_bye);
  return 0;
}
```

Example - cont.

```
/* msg.h - header for
   msg.c */
#ifndef MSG_H_
#define MSG_H_
void prmsg(char *msg);
#endif /* MSG_H_ */
```

```
/* msg.c - function
   declared in msg.h */
#include <stdio.h>
#include "msg.h"
void prmsg(char *msg)
{
  printf("%s\n", msg);
}
```

Example - cont.

- First Method:
  - gcc msg.c main.c -o newhello
- Second Method:
  - gcc -c msg.c
  - gcc -c main.c
  - gcc msg.o main.o -o newhello

Automatic Building with Make

- Automatic Building with Make
  - A GNU utility that determines which pieces of a large program need to be compiled or recompiled, and issues a commands to compile and link them in an automated fashion.
    - Faster compile time
  - Saves you from tedious, huge gcc commands!
    - Simpler for users

Sample Makefile

- Makefiles main element is called a rule:

```
target : dependencies
TAB    commands                          #shell commands
```

Example:

```
my_prog : eval.o main.o
    gcc -o my_prog eval.o main.o

eval.o : eval.c eval.h
    gcc -c eval.c
main.o : main.c eval.h
    gcc -c main.c
```

```
    # -o to specify executable file name
    # -c to compile only (no linking)
```

“clean” Target

- An important target that represents an action rather than a gcc operation.
- Has no dependencies, runs a command to remove all the compilation products from the directory, “cleaning” things up.
- Call by typing “make clean” into prompt.

Example:

```
clean:
        rm –f *.o
```

Make Environment

- Environment variables (PATH, HOME, USER, etc.) are available as $(PATH), etc.
- Also passed to commands invoked
- Can create new variables

Variables

- The old way(no variables)

```
my_prog : eval.o main.o
        gcc -o my_prog eval.o main.o
eval.o : eval.c eval.h
        gcc -c -g eval.c
main.o : main.c eval.h
        gcc -c -g main.c
```

Variables - cont.

- A new way(using variables)

```
C = gcc
OBJS = eval.o main.o
HDRS = eval.h

my_prog : eval.o main.o
        $(C) -o my_prog $(OBJS)
eval.o : eval.c
        $(C) -c -g eval.c
main.o : main.c
        $(C) -c -g main.c
$(OBJS) : $(HDRS)
```

Defining variables on command line

- Take precedence over variables defined in the makefile.
  - make C=cc

Make Variables

$@	name of current target
$?	list of dependencies newer than target
$<	name of dependency file
$*	base name of current target
$%	for libraries, the name of member

Using Makefiles

Naming:

- *makefile* or *Makefile* are standard
- other name can be also used

Running `make`

`make`

`make -f filename` – if the name of your file is not "makefile" or "Makefile"

`make target_name` – if you want to make a target that is not the first one

To Dos

- Write a C program to convert Fahrenheit to Celsius.
- Write a C program to check whether the temperature is under the comfort zone.
- Compile and run them.
- Write a makefile with variables to do so.

- C = (F – 32)*5/9
- comfort zone in celsius: 23°