M8 – Build Automation

CS 136L F22 – LEC 10

Yiqing Irene Huang, Qianqiu Zhang

UNIVERSITY OF WATERLOO | FACULTY OF MATHEMATICS
DAVID R. CHERITON SCHOOL OF COMPUTER SCIENCE

Disclaimer

- The following slides were not presented page by page in class.
- They are my own study notes to share with students.
- In the lab session, we will cover key points, do small demos and give hints on commonly seen errors

Main Points

Separately compile each file and use of make utility

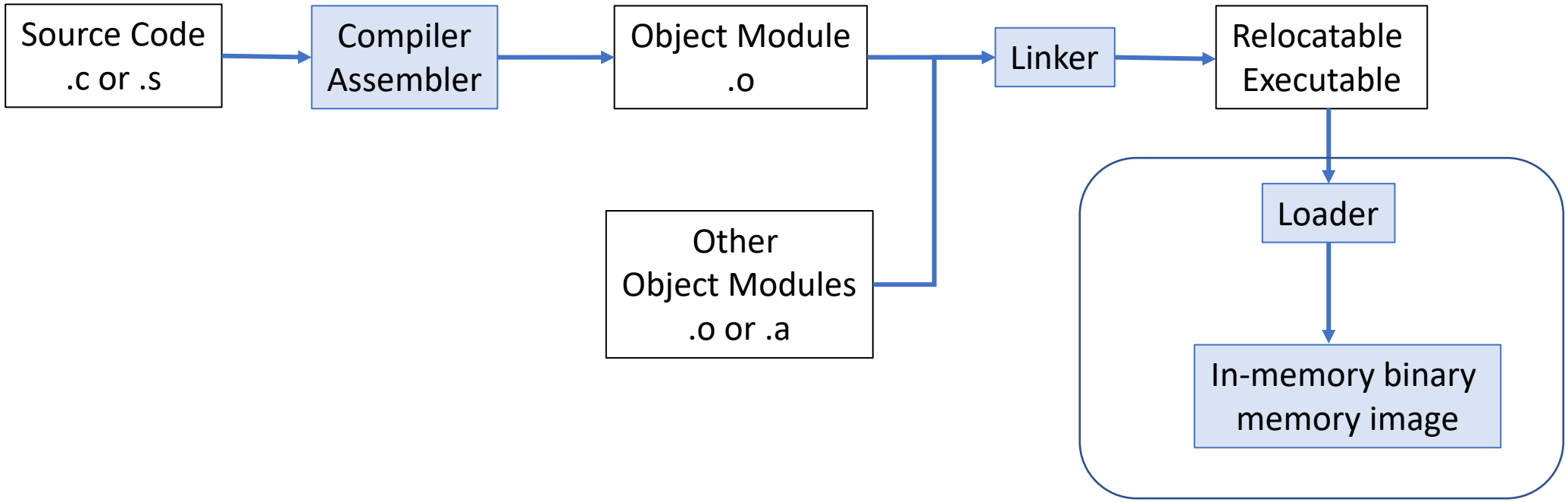
- Separately compile implementation files and then link to build the target
- How to write a Makefile to automate the build process

Lab Thresholds

Question	Description	# of Tests	Pass	Complete
Q1	Makefile for a big project	6	2	6
Q2	Makefile for Document viewer program	5	2	5
Q3	Makefile Command line Argument	3	1	3

- The `cs136.h` and `cs136.o` are in `/u/cs136l/pub/common`.
- The undefined reference to ‘symbol name’ error: you missed certain .o files.
- Start with a sample Makefile in 8.3 and modify it accordingly.

Steps of Compilation



Build Automation

- Automate tasks
- Popular build automation software
  - Make (https://www.gnu.org/software/make/manual/html_node/index.html)
  - Ant
  - Maven
  - Gradle
  - Bazel

Make

- Makefile Basic

```
target: prereq1 prereq2
	commands
```

This is a TAB, not white spaces. **You have been warned!**

- Example:

```
main.out: main.o
	clang -o main.out main.o
main.o: main.c
	clang -c main.c
```

- To build: type "`make main.out`" or just "`make`"

The Make Variables

```
CC=clang
#CC=gcc
CFLAGS=-Wall -g -O0 -std=c99
LD=${CC}
LDFLAGS=
OBJS=main.o

all: main.out
main.out: ${OBJS}
	${LD} ${LDFLAGS} -o main.out ${OBJS}

main.o: main.c
	${CC} ${CFLAGS} -c main.c

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

$(VAR_NAME) is the same as ${VAR_NAME}
White spaces are allowed around "="
Default target is the first target

Implicit recipe, can be omitted

The Implicit Recipe

```
CC=clang
CFLAGS=-Wall -g -O0 -std=c99
LD=${CC}
LDFLAGS=
OBJS=main.o

all: main.out
main.out: ${OBJS}
	${LD} ${LDFLAGS} -o main.out ${OBJS}

main.o: main.c

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

Implicit recipe ${CC} ${CFLAGS} -c will be used

The Header File Dependency

```
CC=clang
CFLAGS=-Wall -g -O0 -std=c99
LD=${CC}
LDFLAGS=
OBJS=main.o

all: main.out
main.out: ${OBJS}
	${LD} ${LDFLAGS} -o main.out ${OBJS}

main.o: main.c main.h

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

More Dependent Files

```
CC=clang
CFLAGS=-Wall -g -O0 -std=c99 -I/u/cs136l/pub/common
LD=${CC}
LDFLAGS=
OBJS=main.o helper.o
LIB_OBJS=/u/cs136l/pub/common/cs136.o

all: main.out
main.out: ${OBJS}
	${LD} ${LDFLAGS} -o main.out ${OBJS} ${LIB_OBJS}

main.o: main.c main.h
helper.o: helper.c /u/cs136/pub/common/cs136l.h

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

More About The Dependency

- Library header files
  - Rarely changes, so do not list in the dependency
- Your own header files
  - You need to recursively specify them
  - `-MMD` flag will automatically generate the .d file
- Use the `include` directive to include .d files
  - The `-` before the `include` means make will throw no error if the files do not exist. We use "`-include`".

The Wildcard function

```
CC=clang
CFLAGS=-Wall -g -O0 -std=c99 -MMD
LD=${CC}
LDFLAGS=
SRC = $(wildcard *.c)
OBJS = $(SRC:.c=.o)
DEPENDS = $(OBJS:.o=.d)

all: main.out
main.out: ${OBJS}
	${LD} ${LDFLAGS} -o main.out ${OBJS}

-include ${DEPENDS}

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

The Linking Error

```
main.c:(.text+0x23c): undefined reference to `read_int'
/usr/bin/ld: main.c:(.text+0x249): undefined reference to `READ_INT_FAIL'
/usr/bin/ld: main.c:(.text+0x28b): undefined reference to `_TRACE_MSG'
/usr/bin/ld: main.c:(.text+0x2ae): undefined reference to `_TRACE_INT'
/usr/bin/ld: main.c:(.text+0x2ef): undefined reference to `READ_INT_FAIL'
/usr/bin/ld: main.c:(.text+0x322): undefined reference to `_TRACE_MSG'
/usr/bin/ld: main.c:(.text+0x361): undefined reference to `_TRACE_MSG'
/usr/bin/ld: main.c:(.text+0x384): undefined reference to `_TRACE_INT'
```

- Use `nm` on the .o file, you will see the symbols
- Use `grep` to find out which header file has the symbol in question
- Link the corresponding `.o` file to resolve the error

Command Line Make Variable Assignment

- You may specify the make variable value at command line
  - `make VAR_NAME="value"`
  - `make VAR_NAME=value`
- Which one(s) use gcc to build the project without modifying the makefile?

  a) `make CC=gcc`

  b) `make CC="gcc"`

  c) `make CC:="gcc"`

  d) `make CC<-"gcc"`

  e) `make CC<="gcc"`

Acknowledgement

- Slides by courtesy of Carmen Bruni and Dave Tompkins
- Demo notes from Nomair Naeem
- Demo lectures by Carmen Bruni, Dave Tompkins, and Nomair Naeem

References

- CS 136L edX notes at https://online.cs.uwaterloo.ca/