Introduction to Makefiles
(Trivial version)

Systems Programming

JOHANNES KEPLER
UNIVERSITÄT LINZ

Michael Sonntag
Institute of Networks and Security

INSTITUTE
OF NETWORKS
AND SECURITY

Makefiles
Automating the build process

- Makefiles allow you to **automate the build process**
  - Not issuing the same commands every time
  - Automatically recompile all changed files
- A kind of “intelligent” macro
  - Or: a separate programming language!
- Drawbacks:
  - Can be complex to write
  - Can be hard to understand
- Note: We will cover **only** a few extremely **basic aspects** here!
  - Today the actual makefiles (still used widely on Linux!) are typically automatically generated through build systems or from configuration files, and not manually anymore

Anatomy of a makefile

- A makefile can have any name
  - But if you call it `Makefile`, it will be found automatically!
    - Merely calling "`make`" is sufficient then
  - Running `make` will execute the first rule; otherwise specify target to build
- **Dependencies** are calculated **automatically** based on prerequisites
  - How? **Modification time** of any file in the dependency list is later than the modification time of the target file, or the target file does not exist

- A makefile consists of several **rules**:

```
target: prerequisites
	recipe
```

Single tab →

  - Attention: **All "recipe" lines** must start with a "**tabulator**"!
  - **Target**: What this rule produces
  - **Prerequisites**: If any of these has changed, this rule must be executed
    - Empty → Will always be executed
  - **Recipe**: How to produce the target

A simple makefile

```
all: test

clean:
   rm -rf *.o
   rm -f test

test: test.o lib.o
   gcc test.o lib.o -o test

test.o: test.c lib.h
   gcc -c test.c -o test.o

lib.o: lib.c lib.h
   gcc -c lib.c -o lib.o
```

- One common header file and two source files, all must be linked together
- `all`: Produce everything (first target runs if no parameter given: **`make`**)
- `clean`: Remove all intermediate products (to run: **`make clean`**)

A more complex makefile

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

SOURCES=$(wildcard *.c)
EXECUTABLES=$(SOURCES:%.c=%)

all: $(EXECUTABLES)

clean:
	rm -rf $(EXECUTABLES)
```

This will create an executable for every "`.c`" file in this directory

- ☐ Compile it with gcc, show all warnings, and insert debug information
- ☐ **Each .c file produces a separate executable, not a single one from all files!**

JOHANNES KEPLER
UNIVERSITÄT LINZ

INSTITUTE
OF NETWORKS
AND SECURITY

https://www.ins.jku.at

THANK YOU FOR YOUR ATTENTION!

Michael Sonntag
michael.sonntag@ins.jku.at
+43 (732) 2468 - 4137
S3 235 (Science park 3, 2nd floor)