

# File Structures

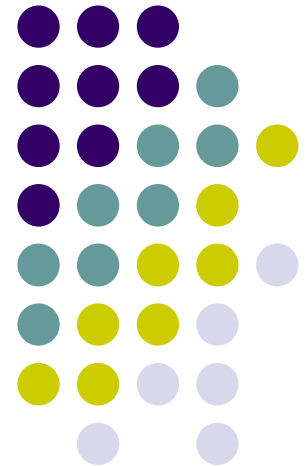
## 05. Appendix. Shared library and Recursive compile

2020. Spring

Instructor: Joonho Kwon

[jhkwon@pusan.ac.kr](mailto:jhkwon@pusan.ac.kr)

Data Science Lab @ PNU



# Outline



- Shared library
- Recursive compile for source codes

# Libraries



- Libraries that are bound at compile-time are called:
  - **Static Libraries** (both Linux and Windows)
- Libraries that are loaded into an executable at run-time are called:
  - **Shared Libraries (Unix and Linux)**
  - **Dynamic Link Libraries** (Windows)

# Static library(1/2)




- Static library
  - functions are copied into a.out during gcc time
  - binding: compile time
  - suffix: .a

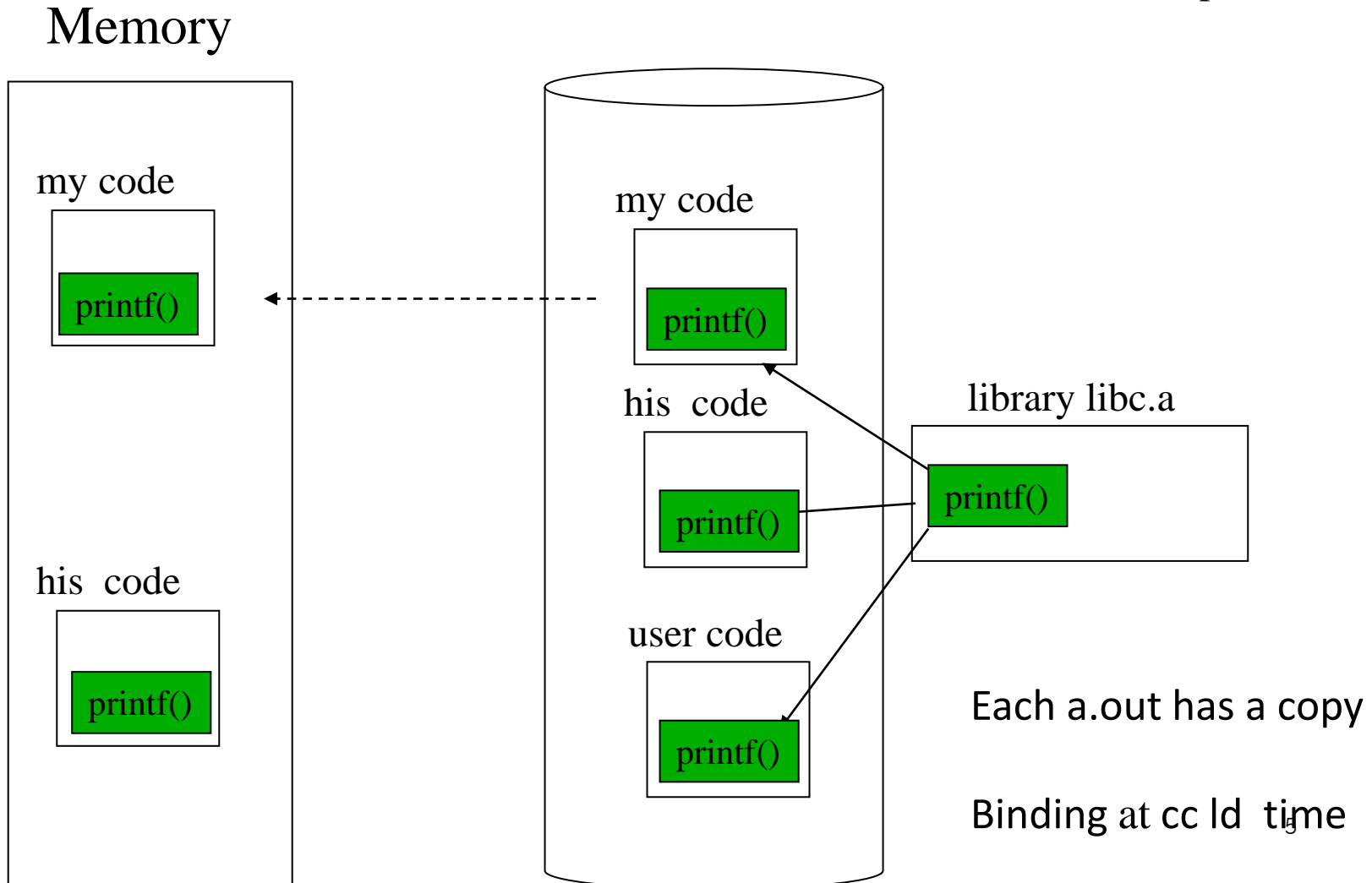
# Static library(2/2)



If `printf()` is \*.a library

 : a.out

 : \*.O (`printf()`)



# Shared library(1/2)

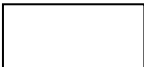



- Shared library (DLL: Dynamically Linking Library)
  - functions are not copied during gcc time. Just map.
  - During the run, function call access the page table
  - load on demand from library (pages are shared)
  - binding: run time
  - suffix **.so**
  - default linking (eg `/lib/libc.a`, `/lib/libc.so`)

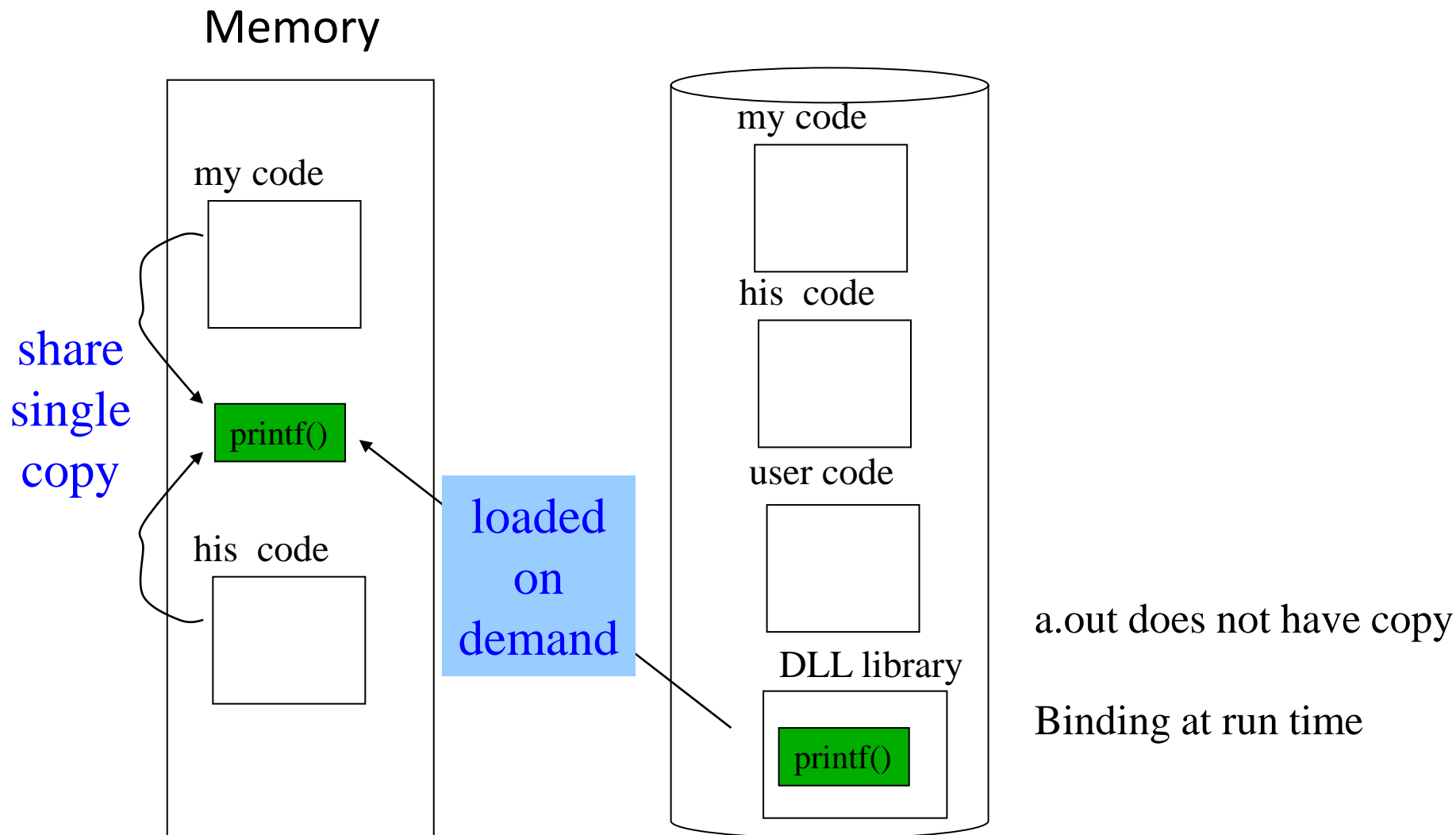
# shared library(2/2)

## DLL library



 : a.out

 : \*.O (printf())



# Comparison(1/2)



- Advantage of shared library
  - Many processes share one lib function in memory
    - eg many a.out's calls printf() --- one copy shared
  - less memory space, fewer page faults
  - library function update is automated

# Comparison(2/2)



- Disadvantage of shared library
  - Cannot give a.out to others who do not have lib!
    - Do they have shared library files? Same version?
  - Slower -- page faults during run time (for lib calls)
  - Not for small, short-lived programs
    - heavy overhead in initial bookkeeping during gcc

# Static libraries: How to use



- Create example:

```
$ gcc -c file1.c file2.c file3.c  
$ ar r libdemo.a file1.o file2.o file3.o  
$ rm file1.o file2.o file3.o
```

- Using library

```
$ gcc -c main.c  
$ gcc -o main main.o libdemo.a  
$ gcc -o main -static -L. main.o -ldemo
```

- Alternate use

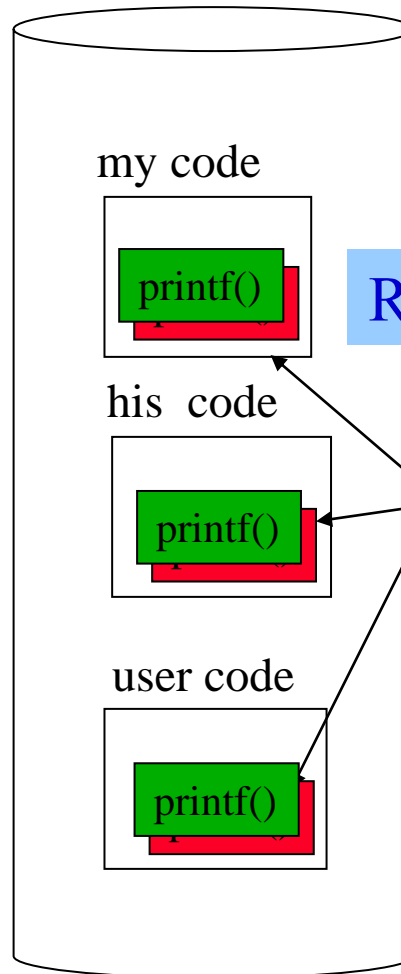
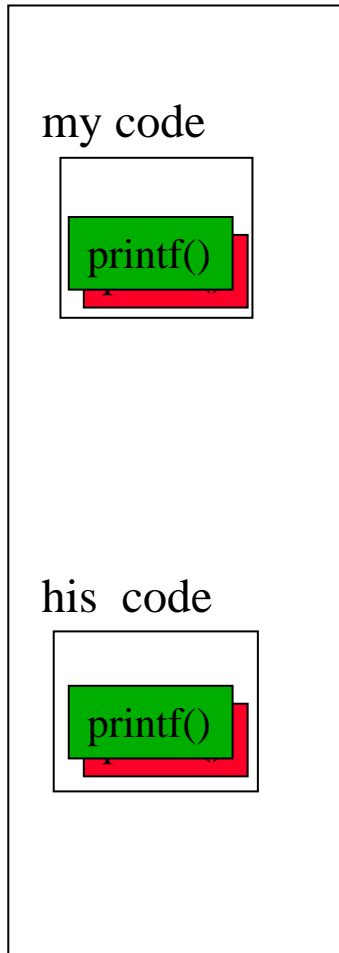
- when library is in standard place like /usr/lib


```
$ gcc -o main main.o -ldemo
```


# Update in \*.a library



Memory

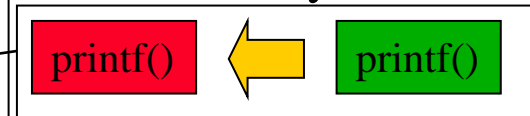


 : a.out

 : \*.o (printf())

Relink & rebuild all a.out's

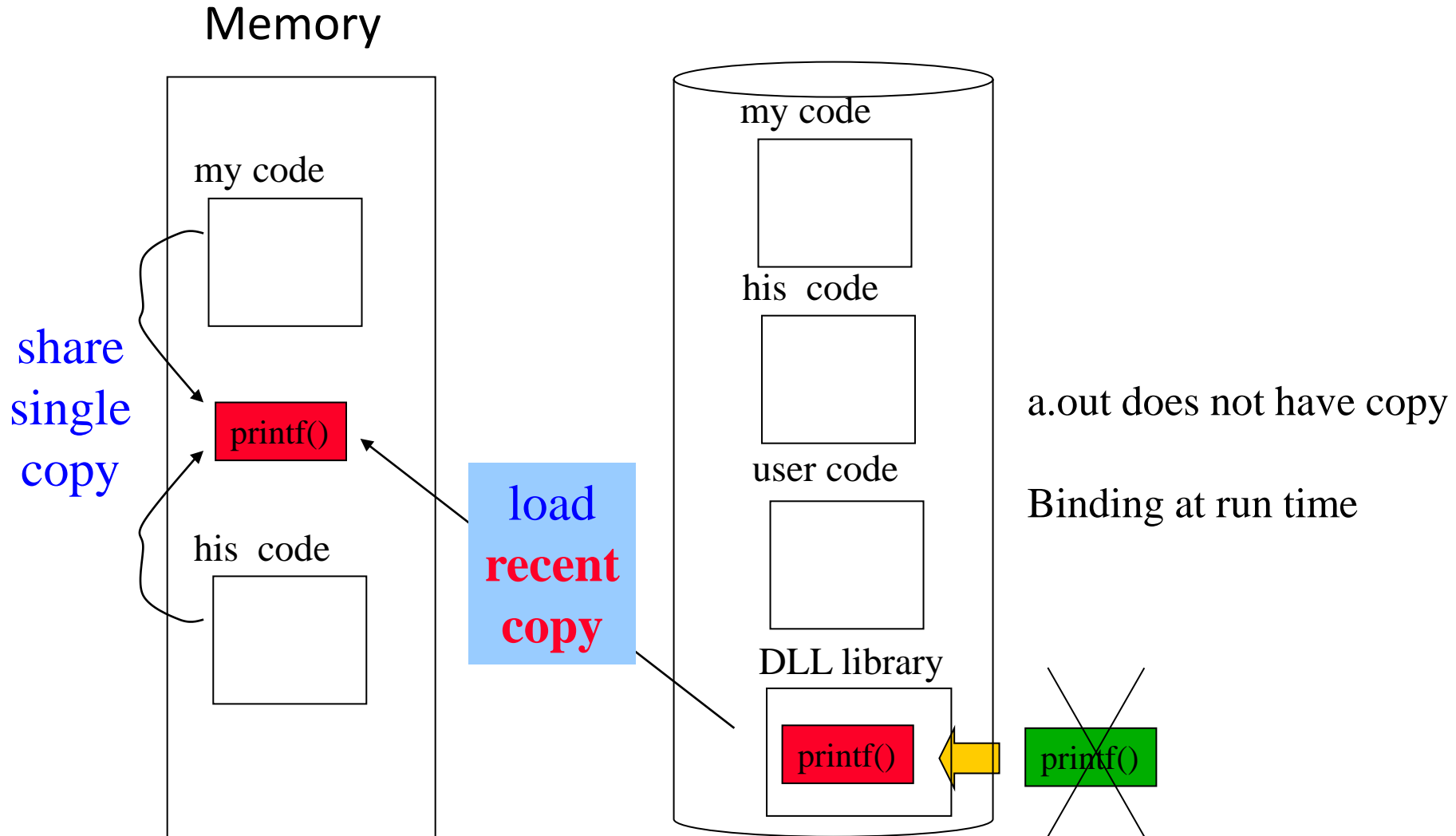
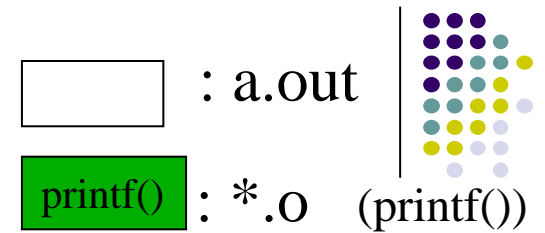
static library



Updated?

If Million a.out  
have printf()?

# Update in DLL library



# Static library demo



```
# Create static library's object file, libhello-static.o.  
$gcc -Wall -g -c -o libhello-static.o libhello.c  
  
# Create static library.  
$ar rcs libhello-static.a libhello-static.o  
  
# Compile demo_use program file.  
$ gcc -Wall -g -c demo_use.c  
  
# Create demo_use program;  
$ gcc -g -o demo_use_static demo_use.o -L. -lhello-static  
  
# Execute the program.  
$ ./demo_use_static
```

# Shared libraries: how to use



- Creating shared library:

```
$ gcc -c -fPIC -Wall file1.c file2.c file3.c  
$ gcc -shared -o libdemshare.so file1.o file2.o file3.o
```

- Using a shared library:

```
$ gcc -Wall -o main main.c libdemshare.so  
$ export $LD_LIBRARY_PATH=$LD_LIBRARY_PATH:..  
$ ./main
```

# Name of Shared library (1/2)



- Suffix `.so`
  - so: “shared object” but plus some more
- “soname”:
  - eg `libc.so.5`
  - part 1 name of library: `libc.so`
  - part 2 version number: 5  
(update if interface is changed)
  - Even longer names:
    - `libc.so.5.m.r` -- minor version, release number

# Name of Shared library (2/2)



- **name.so**
  - linker(ld) uses this name at compile time
- **soname**
  - dynamic linker(dl) uses this name to identify a shared library at run time

<b>name.so</b>	<b>soname</b>	<b>libname.so.ver.min</b>
libc.so	<b>libc.so.5</b>	libc.so.5.4.22

# Shared library demo (1/3)



- 1 Create shared library

```
# Create shared library's object file, libhello.o  
$gcc -fPIC -Wall -g -c libhello.c
```

```
# Create shared library.
```

```
$ gcc -shared -Wl,-soname,libhello.so.0 -o libhello.so.0.0 libhello.o -lc
```

W(Lower L),  
no space

- 2. symbolic links

```
$ ln -sf libhello.so.0.0 libhello.so
```

```
$ ln -sf libhello.so.0.0 libhello.so.0
```

for gcc

For  
dynamic  
linking

# Shared library demo (2/3)



## ● 3. Compile executable file and check

```
# Compile demo_use program file.
```

```
$ gcc -Wall -g -c demo_use.c
```

```
# Create demo_use program;
```

```
$ gcc -g -o demo_use_shared demo_use.o -L. -lhello
```

```
$ ldd demo_use_shared
```

```
linux-vdso.so.1 (0x00007ffcb3101000)
```

```
libhello.so.0 => not found
```

```
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fb4aed49000)
```

```
/lib64/ld-linux-x86-64.so.2 (0x00007fb4af33c000)
```

```
# Configure the library path
```

```
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:.
```

# Shared library demo (3/3)



- 4. check and execute

```
$ ldd demo_use_shared
linux-vdso.so.1 (0x00007ffc0e5b9000)
libhello.so.0 (0x00007faec9d1d000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007faec992c000)
/lib64/ld-linux-x86-64.so.2 (0x00007faeca121000)

# Execute the program.
$ ./demo_use
```

# Compile environments



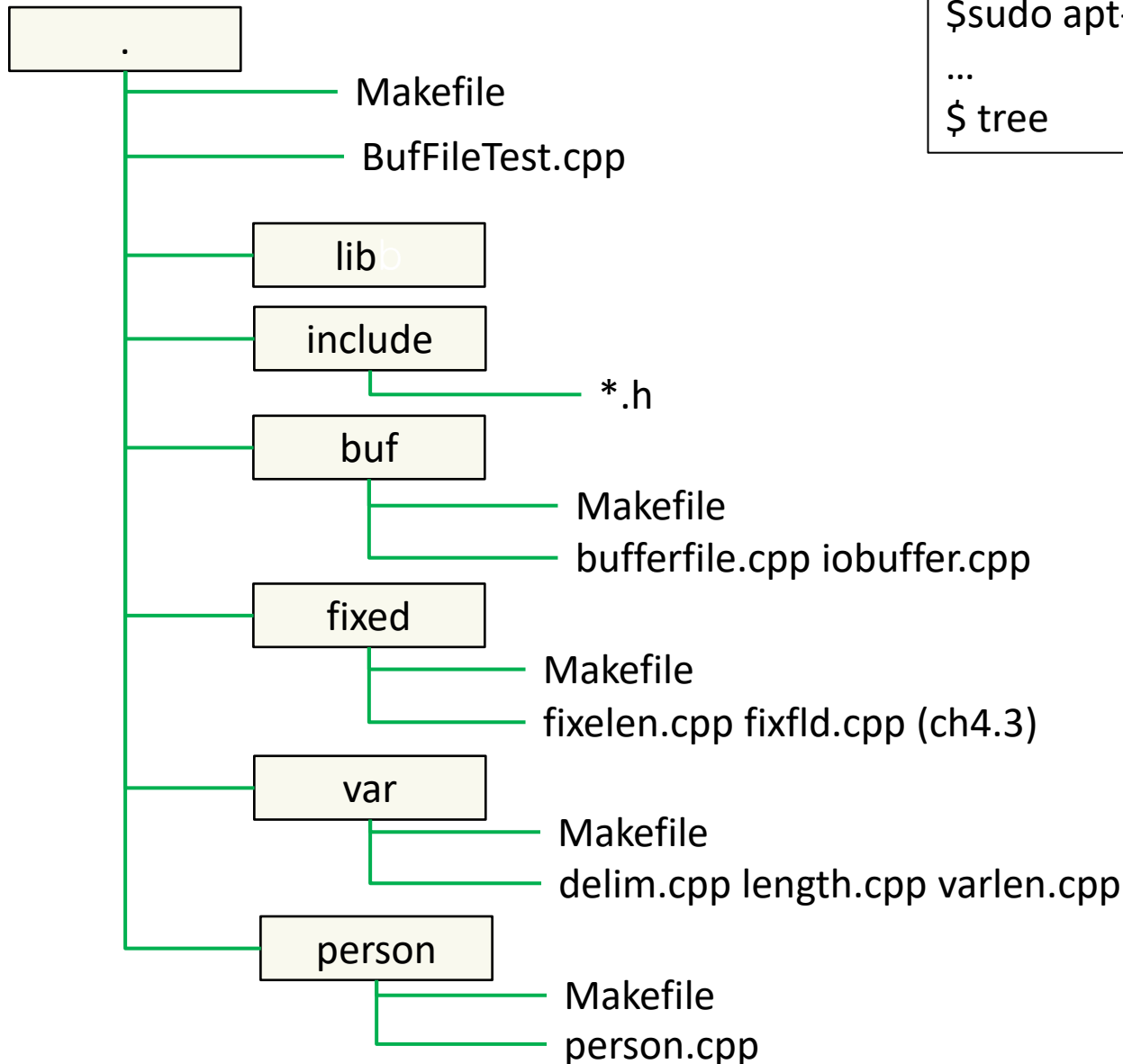
- Compiler: g++
  - Compile \*.cpp and linking as a shared library
    - Two symbolic links are also used
- Make
  - Recursive makefile will be used

# Outline



- Shared library
- Recursive compile for source codes

# Directory structures



```
$sudo apt-get install tree
...
$ tree
```

# 1. For “Buffer” directory (1/2)



- Put Header files at the include directory
  - iobuffer.h buffile.h
- Put \*.cpp files at buffer directory
  - iobuffer.cpp buffile.cpp
- Compile as \*.o file and link as a shared library
  - -fPIC option
  - -shared, -wl option
- copy to lib and make a symbolic link

# 1. For “Buffer” directory (2/2)



- Makefile in “buffer” directory
  - Check whether this makefile works or not

```
CC=g++
LIBNAME=libmybuffer
LIB=$(LIBNAME).so.1.0
CFLAGS= -Wall
OBJS = buffile.o iobuffer.o
INCDIR=../include
LIBDIR=../lib

$(LIB): $(OBJS)
    $(CC) -fPIC -shared -Wl,-soname=$(LIBNAME).so.1 $(OBJS) -o $@ -lc
    cp $(LIB) $(LIBDIR)
    ln -s $(LIBDIR)/$(LIBNAME).so.1.0 $(LIBDIR)/$(LIBNAME).so
    ln -s $(LIBDIR)/$(LIBNAME).so.1.0 $(LIBDIR)/$(LIBNAME).so.1

%.o: %.cpp
    $(CC) -fPIC -c $(CFLAGS) -I$(INCDIR) -o $@ $<

clean:
    -rm -rf $(OBJS) $(LIB) $(LIBDIR)/$(LIBNAME).so*
```

## 2. For “fixed” directory (1/2)



- Put Header files at the include directory
  - fixlen.h fixfld.h
- Put \*.cpp files at buffer directory
  - fixlen.cpp fixfld.cpp
- Compile as \*.o file and link as a shared library
  - -fPIC option
  - -shared, -wl option
- copy to lib and make a symbolic link

## 2. For “fixed” directory (2/2)



- Makefile in “fixed” directory
  - Check whether this makefile works or not

```
CC=g++
LIBNAME=libmyfixed
LIB=$(LIBNAME).so.1.0
CFLAGS= -Wall
OBJS = fixfld.o fixlen.o
INCDIR=../include
LIBDIR=../lib

$(LIB): $(OBJS)
    $(CC) -fPIC -shared -Wl,-soname=$(LIBNAME).so.1 $(OBJS) -o $@ -lc
    cp $(LIB) $(LIBDIR)
    ln -s $(LIBDIR)/$(LIBNAME).so.1.0 $(LIBDIR)/$(LIBNAME).so
    ln -s $(LIBDIR)/$(LIBNAME).so.1.0 $(LIBDIR)/$(LIBNAME).so.1

%.o: %.cpp
    $(CC) -fPIC -c $(CFLAGS) -I$(INCDIR) -o $@ $<

clean:
    -rm -rf $(OBJS) $(LIB) $(LIBDIR)/$(LIBNAME).so*
```

### 3. For “var” and “person” directory



- Put Header files at the include directory
  - person.h delim.h length.h varlen.h
- Put \*.cpp files
  - delim.cpp, length.cpp, varlen.cpp in the “var” directory
  - person.cpp in the “person” directory
- A similar makefile is used
  - Change LIBNAME and OBJS

## 4. Makefile at the current directory (1/2)



- Recursive compile
  - 1. compile BufFileTest.cpp and
  - 2. For each directory in DIRS list
    - Visit the directory and execute make
    - Compile \*.cpp , create a shared library and link two symbolic links (libmybuffer.so, libmyfixed.so, libmyperson.so, libmyvar.so)
  - 3. link BuffFileTest with BufFileTest.o and other precompiled shared libraries
- Recursive cleanup
  - For loop is used

## 4. Makefile at the current directory (2/2)



```
CC=g++
CFLAGS= -Wall
OBJS = BufFileTest.o
INCDIR=./include
LIBDIR=./lib
DIRS = buf fixed person var

.PHONY: all clean

all: BufFileTest
%.o: %.cpp
    $(CC) -c -I$(INCDIR) $(CFLAGS) -o $@ $<

BufFileTest: $(OBJS)
    @for d in $(DIRS); \
    do \
        $(MAKE) -C $$d; \
    done
    $(CC) -o BufFileTest $(OBJS) -L$(LIBDIR) -lmybuffer -lmyfixed -lmyperson -lmyvar

clean:
    @for d in $(DIRS); \
    do \
        $(MAKE) -C $$d clean; \
    done
    -rm -rf BufFileTest $(OBJS)
```

# How to compile



- At the current directory, execute make

```
$ make
g++ -c -I./include -Wall -o BufFileTest.o BufFileTest.cpp
...
make[1]: Entering directory '/home/csedell/fs2020/ch05/ch5.2_Buff_Recursive/buf'
g++ -fPIC -c -Wall -I../include -o buffile.o buffile.cpp
g++ -fPIC -c -Wall -I../include -o iobuffer.o iobuffer.cpp
g++ -fPIC -shared -Wl,-soname=libmybuffer.so.1 buffile.o iobuffer.o -o
libmybuffer.so.1.0 -lc
cp libmybuffer.so.1.0 ../lib
ln -s ../lib/libmybuffer.so.1.0 ../lib/libmybuffer.so
ln -s ../lib/libmybuffer.so.1.0 ../lib/libmybuffer.so.1
make[1]: Leaving directory '/home/csedell/fs2020/ch05/ch5.2_Buff_Recursive/buf'
make[1]: Entering directory '/home/csedell/fs2020/ch05/ch5.2_Buff_Recursive/fixed'
```

# Execute the program (1/2)



- Execute and see the error message

```
$/BufFileTest
./BufFileTest: error while loading shared libraries: libmybuffer.so.1: cannot open
shared object file: No such file or directory
```

- Check libraries
  - Use ldd program

```
$ ldd ./BufFileTest
    linux-vdso.so.1 (0x00007fff0a588000)
    libmybuffer.so.1 => not found
    libmyfixed.so.1 => not found
    libmyperson.so.1 => not found
    libmyvar.so.1 => not found
    libstdc++.so.6 => /usr/lib/x86_64-linux-gnu/libstdc++.so.6 (0x00007f2e56c91000)
    libgcc_s.so.1 => /lib/x86_64-linux-gnu/libgcc_s.so.1 (0x00007f2e56a79000)
    libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f2e56688000)
    libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007f2e562ea000)
    /lib64/ld-linux-x86-64.so.2 (0x00007f2e5721e000)
```

# Execute the program (2/2)



- Set the shared library and re-execute
  - export LD\_LIBRARY\_PATH

```
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:./lib
$ ./BufFileTest
Initializing 3 Persons
Person:
    Last Name 'Ames'
    First Name 'Mary'
    Address '123 Maple'
    City 'Stillwater'
    State 'OK'
    Zip Code '74075'
...
```



# Q&A

