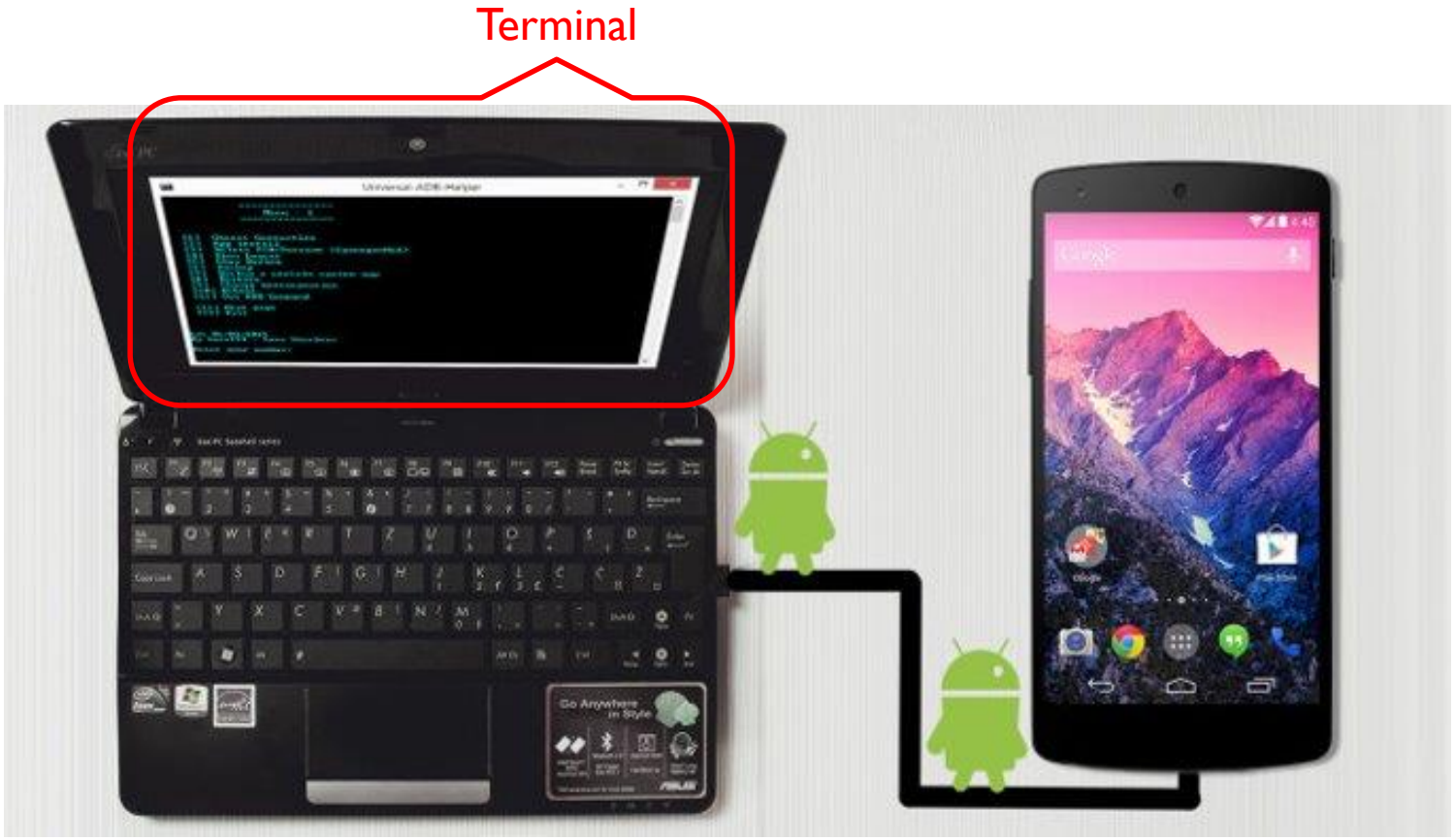
SSE3052: Embedded Systems Practice

Jinkyu Jeong
jinkyu@skku.edu
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu



https://www.makeuseof.com/tag/new-adb-make-process-simple-easy/

adb (Android Debug Bridge)

- Command line tool that lets you communicate wit h an emulator instance or connected device.
  - Copying files to/from device.
  - Installing and debugging apps.
  - Running shell commands.

adb (Android Debug Bridge)

- Located in `Android/Sdk/platform-tools`.
  - Ex) `~/Android/Sdk/platform-tools`

- Set PATH variable.
  - Ex) `PATH=~/Android/Sdk/platform-tools:$PATH`
  - (Append it to `~/.bashrc` file for permanent change)

adb (Android Debug Bridge)

- Query for list of emulator/device instances.
  - `adb devices`

- Start a remote shell in the target instance.
  - `adb shell`
  - `adb -e shell`
  - `adb -d shell`
  - `adb -s <serialNumber> shell`

adb (Android Debug Bridge)

- Copy file to emulator/connected instance.
  - `adb push <local> <remote>`
  - Ex) `adb push foo.txt /data/local/tmp`

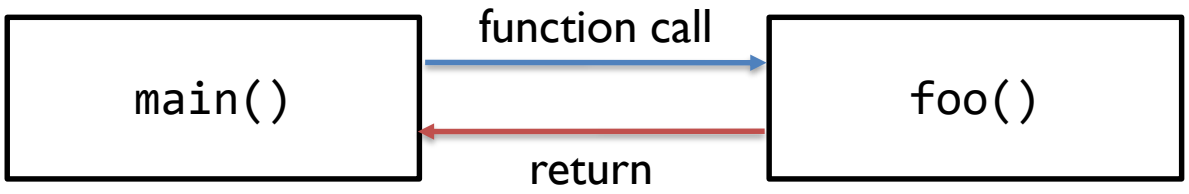
- Copy file from emulator/connected instance.
  - `adb pull <remote> <local>`

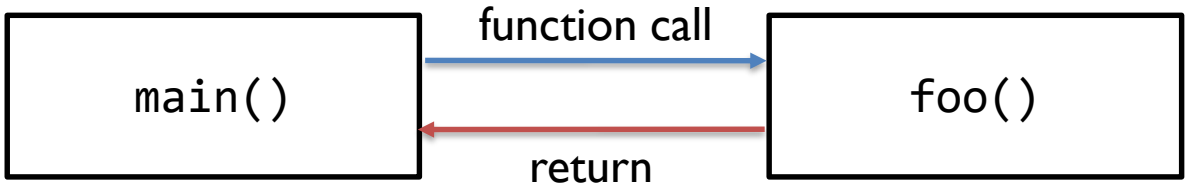
https://developer.android.com/studio/command-line/adb.html

Agenda

1. Add a **system call** to Linux kernel.
2. Invoke added **system call** from user-level program.

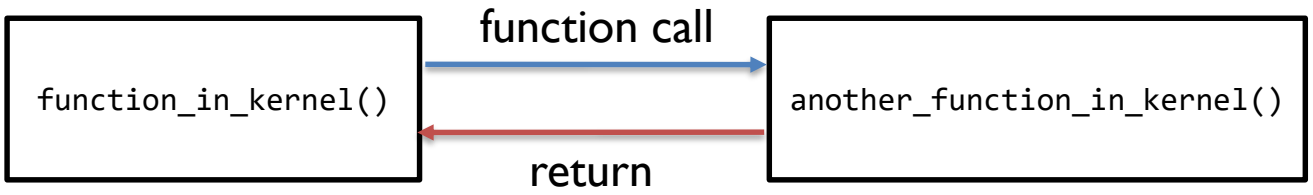
System Call?

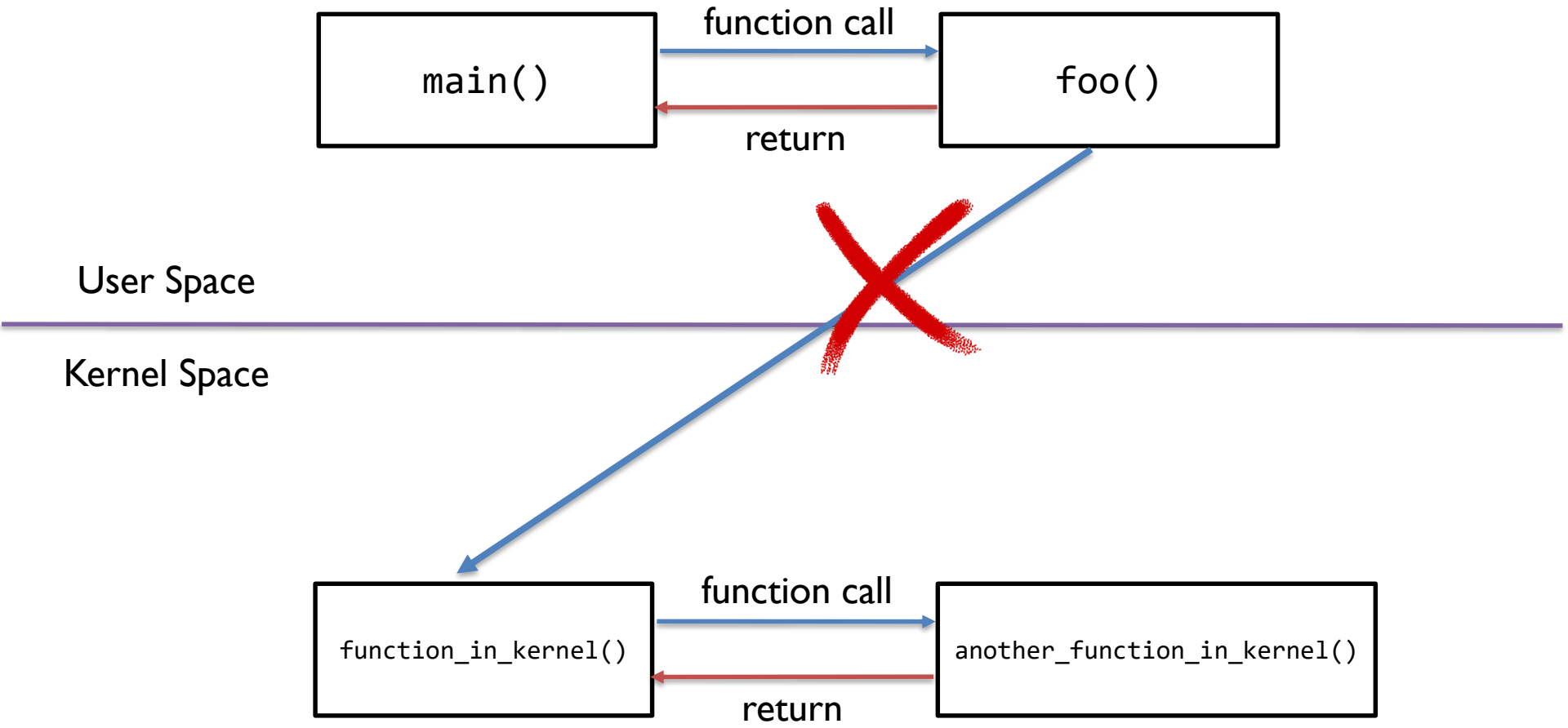

main()
function call
foo()
return

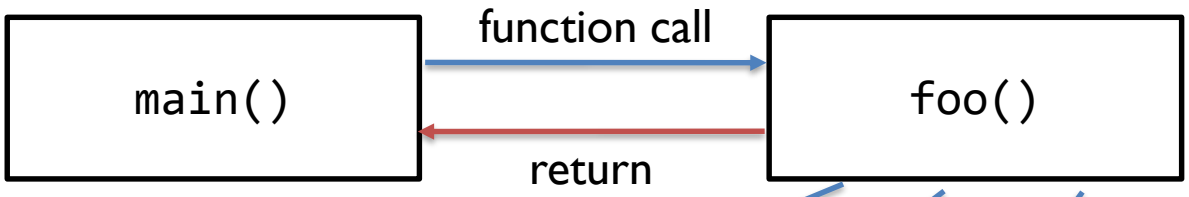


User Space

Kernel Space

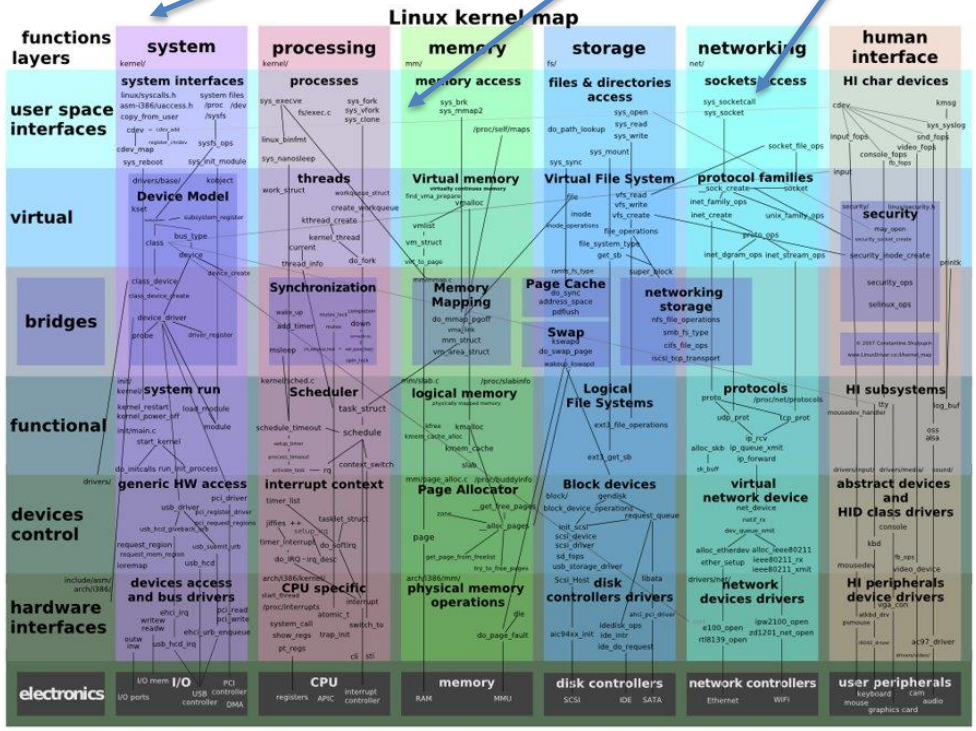



main()
function call
foo()
return
User Space
Kernel Space
function_in_kernel()
function call
another_function_in_kernel()
return

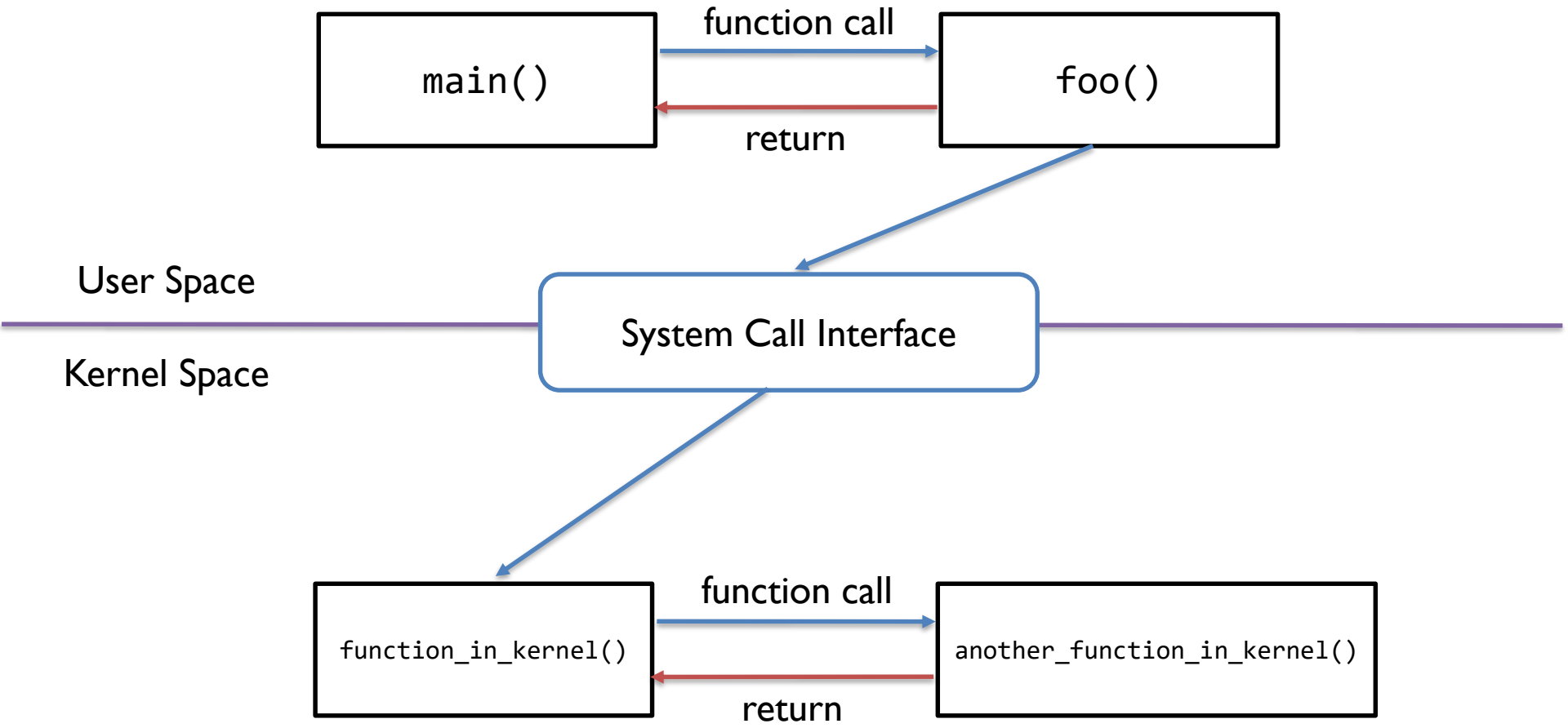


User Space

Kernel Space



http://www.makelinux.net/kernel_map/


main()
function call
return
foo()
User Space
Kernel Space
System Call Interface
function_in_kernel()
function call
return
another_function_in_kernel()

Adding System Call

1. Add a new system call to the system call table.
2. Define (implement) the new system call.
3. Modify Makefile.
4. Build.

Warning: varies by kernel version & architecture.

(We use version: 3.10 & arch: x86_64)

Adding System Call

- In `goldfish/arch/x86/syscalls/syscall_64.tbl`
- Add a system call at the end of the entries.

```
 1 #
 2 # 64-bit system call numbers and entry vectors
 3 #
 4 # The format is:
 5 # <number> <abi> <name> <entry point>
 6 #
 7 # The abi is "common", "64" or "x32" for this file.
 8 #
 9 0       common  read                    sys_read
10 1       common  write                   sys_write
11 2       common  open                    sys_open
12 3       common  close                   sys_close
13 4       common  stat                    sys_newstat
14 5       common  fstat                   sys_newfstat
15 6       common  lstat                   sys_newlstat
16 7       common  poll                    sys_poll
17 8       common  lseek                   sys_lseek
18 9       common  mmap                    sys_mmap
19 10      common  mprotect                sys_mprotect
20 11      common  munmap                  sys_munmap
21 12      common  brk                     sys_brk
22 13      64      rt_sigaction            sys_rt_sigaction
23 14      common  rt_sigprocmask          sys_rt_sigprocmask
24 15      64      rt_sigreturn            stub_rt_sigreturn
25 16      64      ioctl                   sys_ioctl
26 17      common  pread64                 sys_pread64
27 18      common  pwrite64                sys_pwrite64
28 19      64      readv                   sys_readv
29 20      64      writev                  sys_writev
30 21      common  access                  sys_access
31 22      common  pipe                    sys_pipe
32 23      common  select                  sys_select
                                                        1,1           Top
```

Adding System Call

```
# The format is:
# <number> <abi> <name> <entry point>
#
# The abi is "common", "64" or "x32" for this file.
```

Ex) `322  common  minwoo_world  sys_minwoo_world`

Adding System Call

- Create a new c file for implementing system call.
- Ex) `goldfish/arch/x86/mm/syscall_test.c`

```
#include <linux/kernel.h>

asmlinkage long sys_minwoo_world(void)
{
    printk("Hello, My name is Minwoo~~~\n");
    return 0;
}
```

Adding System Call

- Modify `Makefile (arch/x86/mm/Makefile)`.
- Ex) Add syscall_test.o to `obj-y`.

```
obj-y   :=  init.o init_$(BITS).o fault.o ioremap.o extable.o pageattr.o mmap.o \
            pat.o pgtable.o physaddr.o gup.o setup_nx.o syscall_test.o

# Make sure __phys_addr has no stackprotector
nostackp := $(call cc-option, -fno-stack-protector)
CFLAGS_physaddr.o               := $(nostackp)
CFLAGS_setup_nx.o               := $(nostackp)

obj-$(CONFIG_X86_PAT)           += pat_rbtree.o
obj-$(CONFIG_SMP)               += tlb.o

obj-$(CONFIG_X86_32)            += pgtable_32.o iomap_32.o

obj-$(CONFIG_HUGETLB_PAGE)      += hugetlbpage.o
obj-$(CONFIG_X86_PTDUMP)        += dump_pagetables.o

obj-$(CONFIG_HIGHMEM)           += highmem_32.o

obj-$(CONFIG_KMEMCHECK)         += kmemcheck/

obj-$(CONFIG_MMIOTRACE)         += mmiotrace.o
mmiotrace-y                     := kmmio.o pf_in.o mmio-mod.o
obj-$(CONFIG_MMIOTRACE_TEST)    += testmmiotrace.o

obj-$(CONFIG_NUMA)              += numa.o numa_$(BITS).o
obj-$(CONFIG_AMD_NUMA)          += amdtopology.o
obj-$(CONFIG_ACPI_NUMA)         += srat.o
obj-$(CONFIG_NUMA_EMU)          += numa_emulation.o

obj-$(CONFIG_MEMTEST)           += memtest.o
```

Adding System Call

- Build the kernel.
- Check out the system call number.
  - Ex) `grep -nR minwoo_world *`

```
mw@mw:~/Desktop/sse3052/goldfish$ grep -nR minwoo_world *
arch/x86/include/generated/asm/syscalls_64.h:301:__SYSCALL_COMMON(322, sys_minwoo_world, sys_minwoo_world)
arch/x86/include/generated/uapi/asm/unistd_64.h:319:#define __NR_minwoo_world 322
arch/x86/include/generated/uapi/asm/unistd_x32.h:277:#define __NR_minwoo_world (__X32_SYSCALL_BIT + 322)
arch/x86/mm/syscall_test.c:3:asmlinkage long sys_minwoo_world(void)
```

- Copy the kernel image to the appropriate directory.

Agenda

1. Add a **system call** to Linux kernel.
2. Invoke added **system call** from user-level program.

Invoking System Call

1. Set up toolchains. (cross compiler)
2. Write a user-level program that invokes newly ad ded system call.
3. Compile.
4. Copy the executable to the device.
5. Execute.
6. Check out the message.

Invoking System Call

- Go to https://developer.android.com/ndk/downloads/index.html.
- Download NDK.
- Unzip.
- Execute the following:
  - `[path_to_NDK]/build/tools/make_standalone_toolchain.py --arch x86_64 --api 24 --install-dir ~/my-android-toolchain`

Invoking System Call

- Write a user-level program.
- Ex) userspace.c

```
#include <unistd.h>
#define __NR_minwoo_world 322

int main()
{
    syscall(__NR_minwoo_world);
    return 0;
}
```

Invoking System Call

- Compile with `-pie` option.
  - Ex) `~/my-android-toolchain/bin/x86_64-linux-android-gcc -pie userspace.c`
- Copy to `/data/local/tmp (target)`.
  - Ex) `adb push a.out /data/local/tmp`
- Execute & check out the message.
  - `su`
  - `./data/local/tmp/a.out`
  - `dmesg | grep “Minwoo”`

```
generic_x86_64:/data/local/tmp # dmesg | grep "Minwoo"
[  332.662784] Hello, My name is Minwoo~~~
```

Lab

1. Add a system call that print own Student ID and NAME.
2. Invoke added system call from user-level program.
3. Submit report. (include Concept, Implementation, Result)

- Format: yourstudentID_lab1.pdf

  Deadline: 3/22 (Sun.) 23:59