2022 FALL OS
Project 1 Help Document

JINYONG HA

Distributed Computing Systems Laboratory
Department of Computer Science and Engineering
Seoul National University, Korea
2022-09-23

NOTE

- 과제 **#0** Set up development environment

  기한: 09/17 midnight
  방법: etl 제출

- 과제 **#1**

  기한: 10/08 midnight
  방법: 팀별 repository에 proj1 branch 생성 / 제출물은 대표 한 명이 etl에

- 과제 **#2**

  기한: 11/05 midnight
  방법: 팀별 repository에 proj2 branch 생성 / 제출물은 대표 한 명이 etl에

Github repository

[main] https://github.com/jyha200/osfall2022

[per-team] https://github.com/[github_id]/osfall2022-team[#]

reminder:

아래 링크에 **Github ID**를 입력해주세요

https://docs.google.com/spreadsheets/d/1VBGmW-qkLfKm7FeIp5JhEDqcJ0CYFrx3/edit?usp=sharing&ouid=106917183581249491181&rtpof=true&sd=true

What you did in project 0

1. Build your kernel
2. Make image files
3. emulate Tizen with QEMU

What is QEMU

QEMU is a generic and open source machine emulator and virtualizer.

1. Install QEMU
   sudo apt-get install qemu (or sudo apt-get install qemu-system-aarch64)
2. Update /arch/arm64/configs/tizen_bcmrpi3_defconfig file
3. Run Tizen on RPI3 emulation with QEMU

Move files into Tizen when using QEMU

1. Mount `rootfs.img` on ${mnt_dir}
2. Move files under ${mnt_dir}/root/ (You may need sudo)
3. Unmount ${mnt_dir}

- Project 1

General Overview of Project 1

- Write a system call **as a kernel module**
  - `int ptree ( struct prinfo *buf, int *nr)`
  - System call number **398**
  - You can name your function sys_ptree; doesn't matter as long as it works
  - **Tizen repository for project 1 updated**
    - Pull & checkout tizen git repository
      - https://github.com/jyha200/tizen-5.0-rpi3/tree/**project1_base**
    - Contains
      - Simple kernel module example (kernel/hello_mod.c)
      - Some changes to ease project
- Test your system call
  - Print the entire process tree in pre-order

Example Program Output

- `swapper/0` (pid 0)
  - The first ever process created
  - Used to represent the state of ‘not working’
- `systemd` (pid 1)
  - Manages all the processes
- `kthreadd` (pid 2)
  - Kernel thread daemon
  - `kthread_create`

```
sh-3.2# ./proj1
swapper/0,0,0,0,1,0,0
        systemd,1,1,0,167,2,0
                systemd-journal,167,1,1,0,185,0
                systemd-udevd,185,1,1,0,241,0
                dbus-daemon,241,1,1,0,297,81
                amd,297,1,1,0,298,301
                dlog_logger,298,1,1,0,307,1901
                buxton2d,307,1,1,0,313,375
                key-manager,313,1,1,0,325,444
```

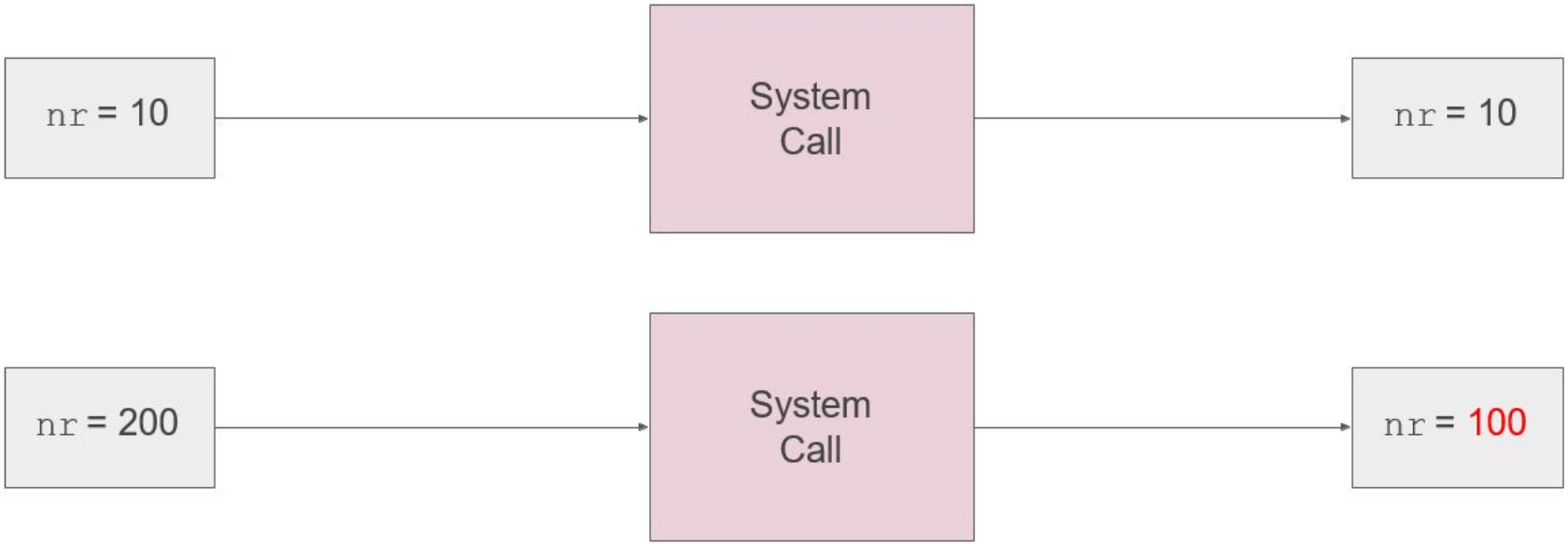
```
        kthreadd,2,1,0,3,0,0
                kworker/0:0,3,1026,2,0,4,0
                kworker/0:0H,4,1026,2,0,5,0
                kworker/u8:0,5,1026,2,0,6,0
                mm_percpu_wq,6,1026,2,0,7,0
                ksoftirqd/0,7,1,2,0,8,0
                rcu_preempt,8,1026,2,0,9,0
                rcu_sched,9,1026,2,0,10,0
```

Return Value

- Success
  - Your system call should return the total number of entries (this may be bigger than the actual number of entries copied).
  - nr can be changed
- Error
  - Error handling: -EINVAL or -EFAULT
  - You may handle other errors but we will not grade them
  - Defined in include/uapi/asm-generic/errno-base.h

nr EXAMPLE

- Assuming the number of total processes is 100:



Error Handling

-EINVAL

- If `buf` and/or `nr` are `NULL`, or if `nr` is less than 1

-EFAULT

- If buf and/or nr are outside the accessible address space

- How to print error messages?
  - `int result = syscall(398, ...);`
  - `printf("%d", result);`
  - `???????`
- You cannot get -EINVAL or -EFAULT as the return value
  - Use `errno` and `perror()`

Check Before Submission!!!!!

- **Implement your own code**
  - You might refer to others code but implementation should be made by yourself with your own style.
  - **Copy 발견 시 불이익 있을 수 있음**
- **All your changes should be included in ‘kernel/ptree_mod.c’**

- Unsafe access to user space memory
- Return value
  - Incorrect return value
  - Not modifying `*nr` when needed
- Whether you follow the project specifications (final check!) …
- … and whether you have delineated all unspecified/different implementation
  - details in README
- **White-box test for this project!**

Reminder

concise README file

- Describe how to build your kernel
- Describe the high-level design and implementation
- Investigation of the process tree
- Any lessons learned

concise 4-minute presentation slides(including video demo submitted on etl)

- Limit: 10 slides including title slide
- We will not consult slides after 10

Presentation should include

A. High level design and implementation
B. video clip that shows that your system works
C. investigation of the process trees
D. lesson learned

About Submission (IMPORTANT!)

- Make sure your branch name is ***proj1***
- **Cheating might be penalized**
- Don't be late! marks will be deducted

We allow 3 days late submission! TA will not grade any submissions after that!

- Slides and Demo

Submit to etl before deadline.

we allow 3 day late submission, after that submission on etl will not be possible

- zip dir title: [OS-ProjX] TeamX submission

inside the directory there should be 4 files:

- The slides to your presentation: TeamX-slides.ppt(.pdf)
- Your Demo Video: TeamX-demo.mp4(.avi….)
- Recorded presentation(sound only): TeamX-presentation.mp3(.wav….)
- ptree_mod.c (for verification)
  - **should be buildable when just place submitted ptree_mod.c to 'kernel/' of project 1 base branch (https://github.com/jyha200/tizen-5.0-rpi3/tree/project1_base)**

Kernel Programming Guideline

Important Directories

`arch`

Architecture dependent (i.e. x86, arm, mips, …) parts of Linux

`kernel`

Common kernel code

`net`

Common network related code

`drivers`

Common driver code for Linux

`fs`

Common file system code for Linux

`include`

Common header files

Things to Keep in Mind...

- No memory protection
  - Corruption in kernel memory space can make the whole machine crash!
- No floating point or MMX operation
  - Dealing with real numbers can be challenging and painful!
  - You unfortunately have to do it for some projects :(
- Rigid stack limit
  - Use extra caution when allocating local arrays or having recursive calls
  - `kmalloc` instead for huge arrays
- Your kernel code will run in a multi-core environment
  - Use proper synchronization mechanisms to avoid race conditions
  - Beware of deadlocks

Accessing User Memory

- In kernel mode, you should avoid directly accessing user memory space
  - Can result in kernel panic

- `include/asm/uaccess.h` provides macros for this
  - `get_user` / `put_user`: copies simple variables
  - `copy_from_user` / `copy_to_user`: copies a block of data
  - More on http://www.ibm.com/developerworks/library/l-kernel-memory-access/

- Mark system call parameters that access user space memory with `__user`
  - e.g. In `include/linux/syscalls.h`:
  - `asmlinkage long sys_time(time_t __user *tloc);`

The User Space Memory Access API

Function	Description
access_ok	Checks the validity of the user space memory pointer
get_user	Gets a simple variable from user space
put_user	Puts a simple variable to user space
clear_user	Clears, or zeros, a block in user space
copy_to_user	Copies a block of data from the kernel to user space
copy_from_user	Copies a block of data from user space to the kernel
strnlen_user	Gets the size of a string buffer in user space
strncpy_from_user	Copies a string from user space into the kernel

kmalloc and kfree

- Used for allocating / releasing kernel memory instead of `malloc` / `free`
- `kmalloc` is similar to `malloc`, but has an additional flag parameter

```
void *kmalloc(size_t size, int flags)
```

Frequently used flags

`GFP_KERNEL` Allocate kernel space memory

`GFP_USER` Allocate user space memory

`GFP_ATOMIC` similar to `GFP_KERNEL` , but cannot sleep; used inside interrupts or other non-sleep routines

More on http://www.makelinux.net/ldd3/chp-8-sect-1.shtml

- `kfree` is similar to `free`

task_struct

- 598 lines!

  You don't need to read everything

- `children` and `sibling`: doubly linked lists

```
/* Real parent process: */
struct task_struct __rcu        *real_parent;

/* Recipient of SIGCHLD, wait4() reports: */
struct task_struct __rcu        *parent;

/*
 * Children/sibling form the list of natural children:
 */
struct list_head                children;
struct list_head                sibling;
struct task_struct              *group_leader;
```

Doubly Linked List in Linux Kernel

- Linux kernel has a doubly linked list implementation for kernel programming

  Extensively used across all Linux kernel code

- Defined in `include/linux/list.h`

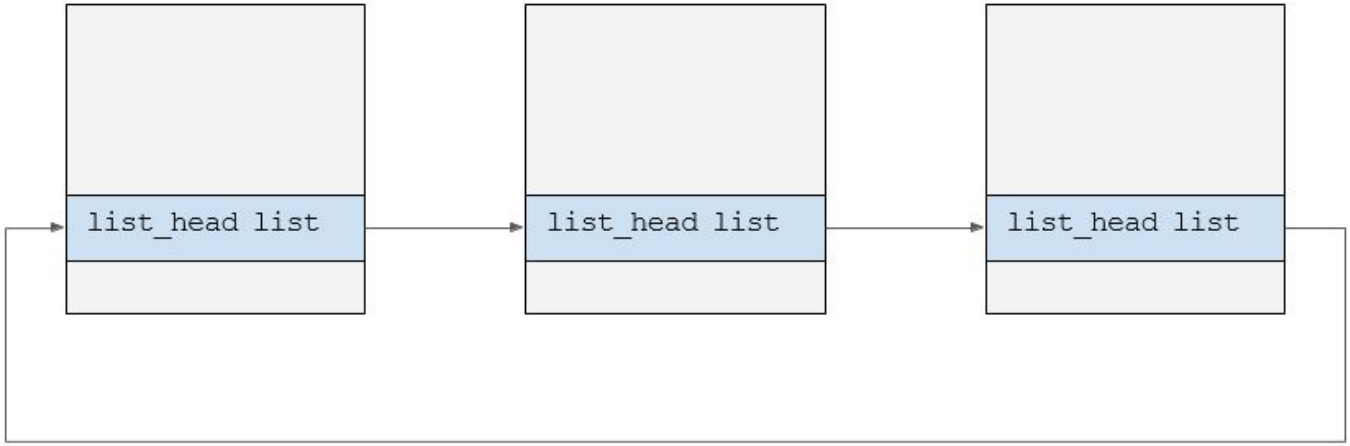
  Can only be used in kernel space!

- Unlike other commonly used linked lists, kernel list nodes are stored ***inside*** data

```
struct student {
    char* name;
    char* student_id;   str
    uct list_head list;
};
```

Doubly Linked List in Linux Kernel (cont'd)

```
struct list_head {
    struct list_head *next, *prev;
}
```



Doubly Linked List in Linux Kernel (cont'd)

- Initializing a list node (must be declared beforehand)
  - `INIT_LIST_HEAD(&first_student->list)`
- Defining and initializing a list head pointer (declaration + `INIT_LIST_HEAD`)
  - `LIST_HEAD(student_list)`
- More about Linux kernel list (highly recommended)

http://www.makelinux.net/ldd3/chp-11-sect-5.shtml

Doubly Linked List in Linux Kernel (cont'd)

- Commonly used macros/functions

`list_add` / `list_add_tail`: adds a node to a list

`list_del` / `list_del_init`: deletes a node from a list

`list_for_each_entry`: iterates over a list

`list_for_each_entry_safe`: iterates over a list **when nodes can be deleted**

`list_entry` / `container_of`: returns the item given a list node

Kernel Module

- What is kernel module?
  - Pluggable binary for extra functionalities of kernel
  - Named as **mod_name.ko**

- How to use
  - Insert
    - # insmod XX.ko
  - Remove
    - # rmmod XX.ko

Replace Syscall Table

- Syscall Table
  - Array of syscall function pointers
  - Indexed by syscall ID
  - See arch/arm64/kernel/sys32.c

- **Change only 398th entry**
  - Otherwise, it might bother testing
  - May be reflected to your score

Implement Kernel Module (See kernel/hello_mod.c)

- module_init(f1)
  - Call module initialization function **f1** when module is inserted
- module_exit(f2)
  - Call module cleanup function **f2** when module is removed
- MODULE_LICENSE()
  - Indicate the license of module
  - Use “GPL v2”

Useful References

- Linux cross reference (LXR)

  https://elixir.bootlin.com/linux/v4.14.67/source

- Unreliable Guide To Hacking The Linux Kernel by Rusty Russel

  http://kernelbook.sourceforge.net/kernel-hacking.pdf