

Project 1 Help Document

Sep 22, 2020

SNU Operating Systems

Project 1

General Overview of Project 1

- Write a system call

- `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

- 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
```

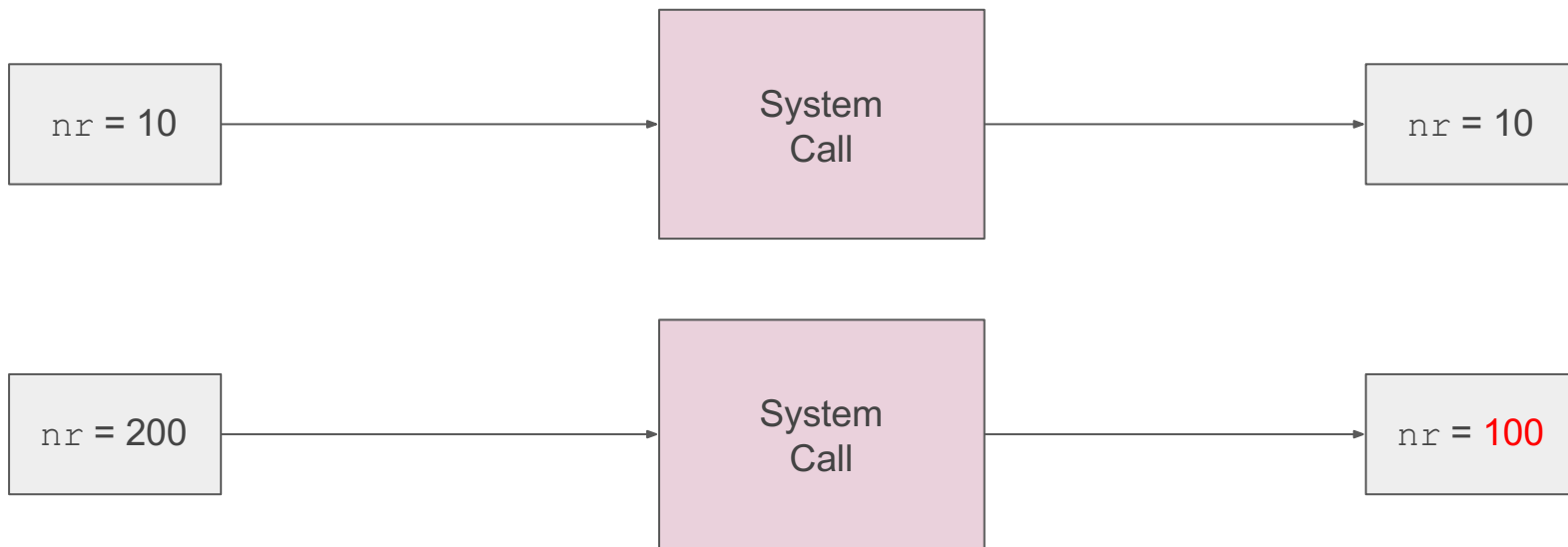
```
      kthreadd,2,1,0,3,0,0
            kworker/0:0,3,1026,2,0,4,0
            kworker/0:0H,4,1026,2,0,5,0
            kworker/u8:0,5,1026,2,0,6,0
            mm_percpu_wq,6,1026,2,0,7,0
            ksoftirqd/0,7,1,2,0,8,0
            rcu_preempt,8,1026,2,0,9,0
            rcu_sched,9,1026,2,0,10,0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

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

Error Handling

- 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!

- **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 your video demo) to os-staff os-tas@dcslab.snu.ac.kr
 - **Limit: 10 slides including the title slide**
 - **I won't look at slides after the 10th.**

- Your presentation includes
 - A. High-level design and implementation
 - B. A video clip** that shows that your system works
 - C. The investigation of the process trees
 - D. Lessons learned

About Submission (IMPORTANT!)

- Make sure your branch name is *proj1*
- Don't be late!
 - TA will not grade the commits after the **deadline**.
- Slides and Demo
 - Send them to the TAs before the **deadline**.
 - os-tas@dcslab.snu.ac.kr
 - Title: [OS-ProjX] TeamX slides&demo submission
 - File name: TeamX-slides.ppt(.pdf), TeamX-demo.mp4(.avi....)
 - No confirmation email this time

Announcement

- Deadline
 - Due: 2020-10-08 Thursday 23:59.
- Check your source code before submission
 - There were some codes which were not compiled....

Kernel Programming

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

Our Architecture

- AArch64 kernel
- ARMv7 userland
- This means there are compatibility issues!
 - Example: `sizeof(long)` is different

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);
```

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

kmalloc and kfree

- Used for allocating / releasing kernel memory instead of `malloc` / `free`
 - Defined in `linux/slab.h`
- `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;
```

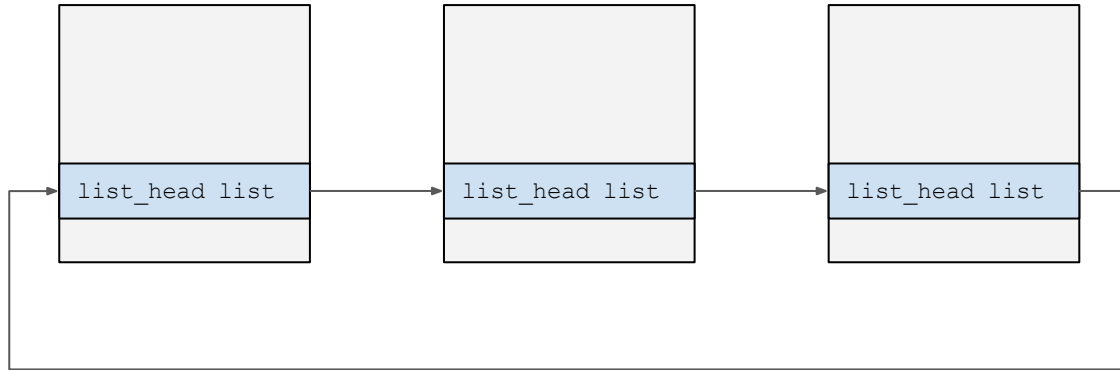
Doubly Linked List in Linux Kernel

- 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)

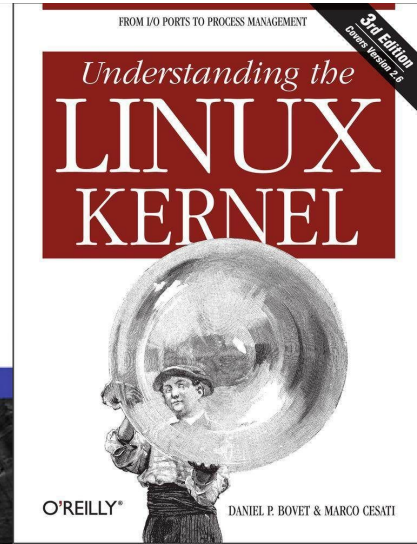
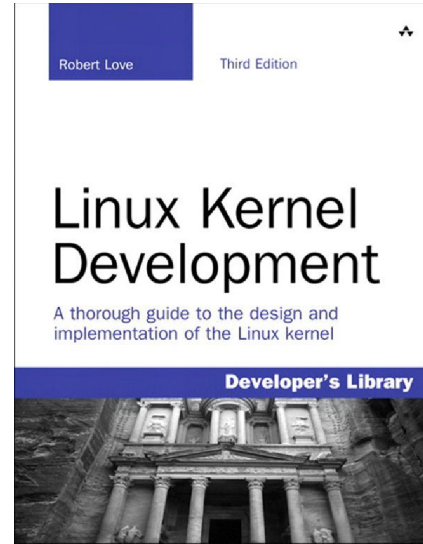
- Initializing a list node (must be declared beforehand)
 - `INIT_LIST_HEAD(&first_student->list)`
- Defining and initializing a list head pointer (declaration + `INIT_LIST_HEAD`)
 - `LIST_HEAD(student_list)`
- More about Linux kernel list (highly recommended)
 - <http://www.makelinux.net/ldd3/chp-11-sect-5.shtml>

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

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>



Q & A

Back-up Slides