Project1 & Kernel Programming

March 21, 2017

SPL, SNU

Project 1

General explanation about proj1

- Writing a system call
  - `int ptree(struct prinfo *buf, int *nr);`
  - It should be assigned number **380**

- Test your system call
  - Print the entire process tree in pre-order

Example program output

- Swapper(pid 0)
  - the process is used to represent the state of 'not working'
- Systemd(pid 1)
  - Manage all the process, init system.
- kthreadd(pid 2)
  - kernel thread daemon.
  - kthread_create()

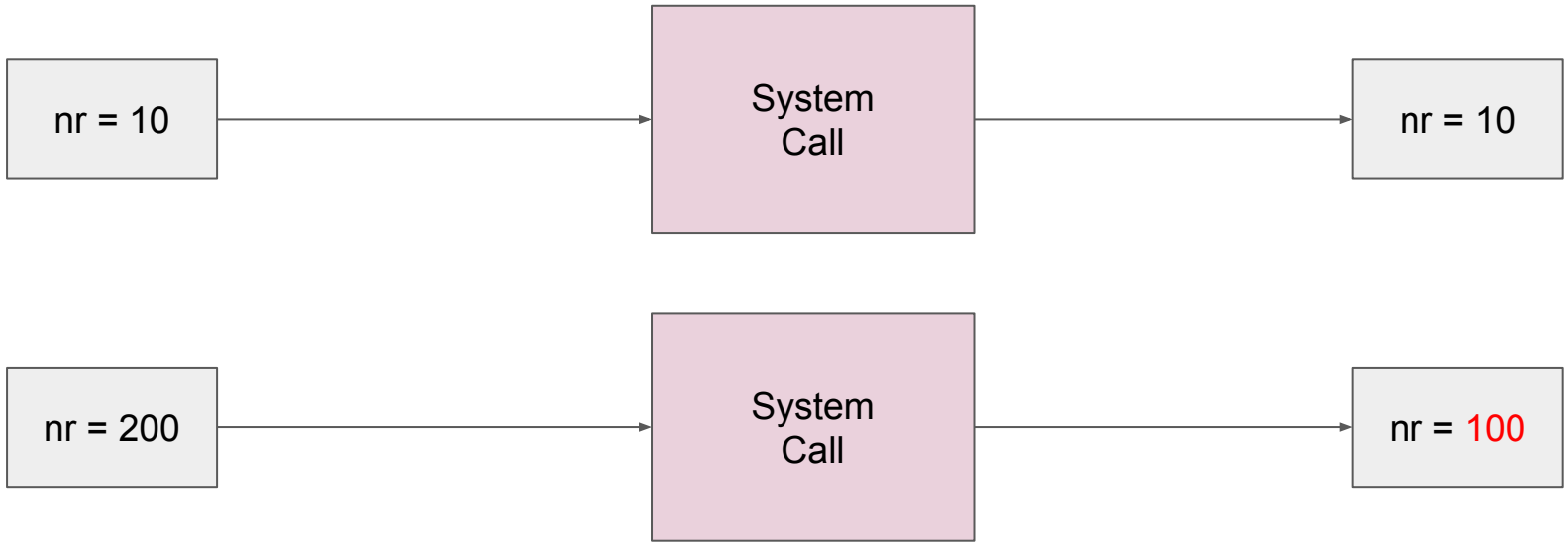
```
systemd,1,1,0,156,2,0
  systemd-journal,156,1,1,0,185,0
  systemd-udevd,185,1,1,0,484,0
  syslogd,484,1,1,0,495,0
    ...
  deviced,802,1,1,1612,857,0
    systemctl,1612,64,802,0,1613,0
    systemctl,1613,64,802,0,1614,0
    systemctl,1614,64,802,0,31175,0
    ...
kthreadd,2,1,0,3,0,0
  ksoftirqd/0,3,1,2,0,5,0
  kworker/0:0H,5,1,2,0,6,0
  kworker/u8:0,6,1,2,0,7,0
```

Return Value

- `int ptree(struct prinfo *buf, int *nr);`
- 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
  - defined in include/uapi/asm-generic/errno-base.h

nr can be changed

- # of total processes: 100



Error Handling

- -EINVAL
  - if buf or nr are null, or the number of entries is less than 1
- -EFAULT
  - if buf or nr are outside the accessible address space

Error Handling

- How to print error message?
  - int result = your_system_call(380, ….);
  - printf ("%d", result);
  - ?????

- You cannot get -EINVAL or EFAULT on the return value
  - Use **errno** to print it out.
  - Use **perror()** function

Check before submission!

- Unsafe access to user-space memory
- Return value
  - Incorrect return value
  - Not modifying *nr when needed
- etc...

Remind

Concise README file

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

Project 1 – what to submit

- Concise 4-minute presentation slides (including your video demo) to ~~os-staff~~ os-tas
  - **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: *proj1*
- Don't be late!
  - TA will not grade the commits after the deadline.
- Slides and Demo
  - Send it to the TA's email (os-tas@spl.snu.ac.kr) before the deadline.
  - os-tas@spl.snu.ac.kr
  - Title: [OS-ProjX] TeamX slides&demo submission
  - File name: TeamX-slides.ppt(.pdf), TeamX-demo.mp4(.avi….)

Kernel Programming

Directories you may need to take a look...

- ./arch
  - Architecture-dependent (i.e. x86, arm, mips, …) parts of linux
  - Tizen is arm-based system :)
- ./kernel
  - Common kernel codes
- ./net
  - Common network-related codes
- ./drivers
  - Common driver codes for linux
- ./fs
  - Common file system codes for linux
- ./include
  - Common header files

Some things you should keep in mind...

- No memory protection
  - Corrupting kernel memory space will make the whole machine crash!
- No floating point or MMX operation
  - Real number calculation might be challenging and painful!
  - You might have to do it on some projects, though :)
- A rigid stack limit
  - Be cautious about allocating local arrays or having recursive calls
  - Use kmalloc() instead for huge arrays
- Your kernel code will be run in multi-core environment
  - Use proper synchronization mechanism to avoid race conditions
  - Be aware of deadlocks

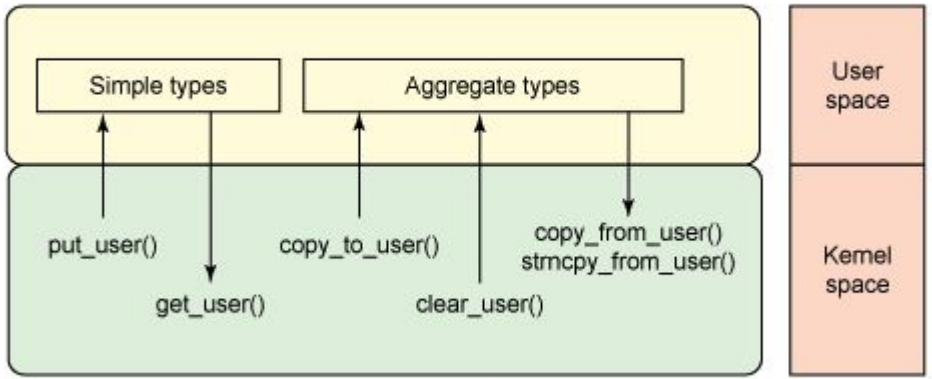
Accessing user-space memory

- In kernel mode, you should avoid accessing user memory space directly
  - Can result in kernel panic
- include/asm/uaccess.h provides user-memory-accessing macros
  - get_user() / put_user(): copies simple variables
  - copy_from_user() / copy_to_user(): copies a block of data
  - More things: http://www.ibm.com/developerworks/library/l-kernel-memory-access/
- Mark system call parameters containing user-space memory pointer with "__user"
  - e.g. In ./include/linux/syscalls.h: asmlinkage long sys_time(time_t __user *tloc);

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

User Space Memory Access API

Figure 4. Data movement using the User Space Memory Access API



kmalloc() / kfree()

- Used for allocating / releasing kernel memory instead of malloc() / free()
  - Defined in linux/slab.h
- kmalloc() usage 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 allocation process cannot sleep. Used inside interrupts or other non-sleep routines.
  - More things: http://www.makelinux.net/books/lkd2/ch11lev1sec4
- kfree() usage is similar to free()

Task_struct

- About 300 lines!
- children, and sibling list → Doubly linked list

```
struct task_struct __rcu *real_parent; /* real parent process */
struct task_struct __rcu *parent; /* recipient of SIGCHLD, wait4() reports */
/*
 * children/sibling forms the list of my natural children
 */
struct list_head children;      /* list of my children */
struct list_head sibling;       /* linkage in my parent's children list */
```

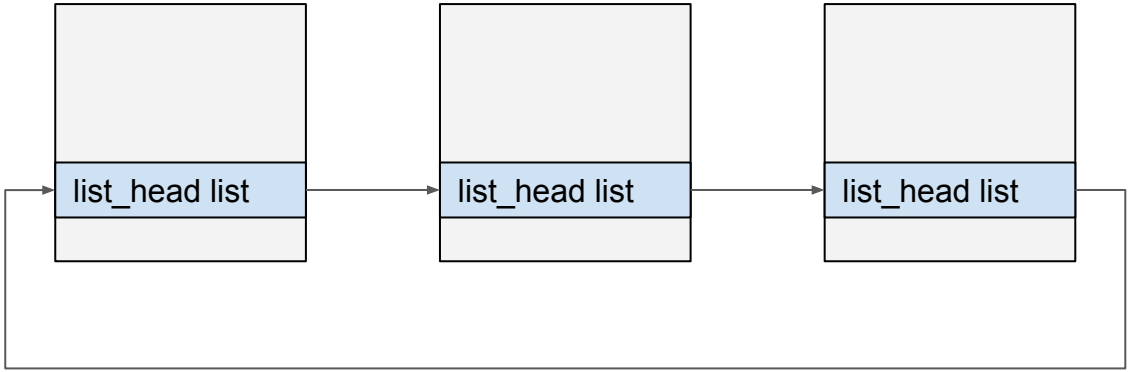
Doubly linked list in linux kernel

- Linux kernel has a doubly linked list implementation for kernel programming
  - Extensively used across all linux kernel codes
- 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;
    struct list_head list;
};
```

Doubly linked list in linux kernel (Contd.)

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



Doubly linked list in linux kernel (Contd.)

- Defining list head pointer
  - LIST_HEAD(student_list)
- Initializing list node
  - INIT_LIST_HEAD(&first_student->list);
- More things on linux kernel list (highly recommended)
  - http://www.makelinux.net/ldd3/chp-11-sect-5
  - https://bbingju.wordpress.com/2013/08/25/linux-kernel-list-h/ (Korean document)

Doubly linked list in linux kernel (Contd.)

- Commonly used macros/functions
  - list_add() / list_add_tail(): adding a node to a list
  - list_del() / list_del_init(): deleting a node from a list
  - list_for_each_entry(): iterating a list
  - list_for_each_entry_safe(): iterating a list **when nodes could be deleted**
  - and many more…(for_each_task():??)

Task_struct - schedule info

- **volatile** long state;*/* -1 unrunnable, 0 runnable, >0 stopped */*
-

```
#define TASK_RUNNING            0
#define TASK_INTERRUPTIBLE      1
#define TASK_UNINTERRUPTIBLE    2
#define __TASK_STOPPED          4
#define __TASK_TRACED           8
/* in tsk->exit_state */
#define EXIT_ZOMBIE             16
#define EXIT_DEAD               32
/* in tsk->state again */
#define TASK_DEAD               64
#define TASK_WAKEKILL           128
#define TASK_WAKING             256
#define TASK_PARKED             512
#define TASK_STATE_MAX          1024
```

Task_struct - schedule info

- int prio, static_prio, **normal_prio**;
  unsigned int rt_priority;
  const struct **sched_class** ***sched_class**;
  struct **sched_entity** **se**;
  struct **sched_rt_entity** **rt**;

- Scheduling policy and priority

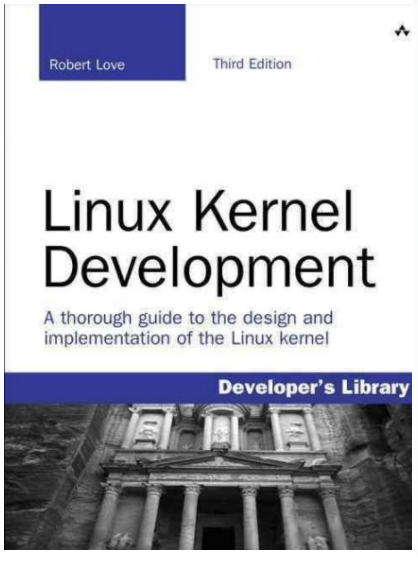
Task_struct - cpu mask

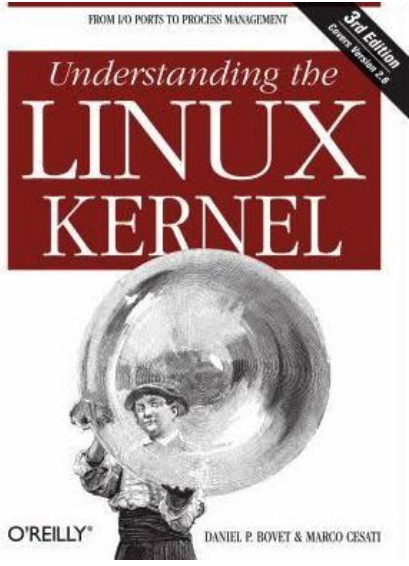
- int nr_cpus_allowed;
  cpumask_t cpus_allowed;

- The task can be running in the cpu in cpus_allowed

Some useful references

- Linux cross reference
  - http://lxr.free-electrons.com/source/?v=3.10
- Unreliable Guide To Hacking The Linux Kernel by Rusty Russel
  - http://kernelbook.sourceforge.net/kernel-hacking.pdf





Q & A

Back-up Slides