CS 251: Real-time Embedded Systems
Homework 1

Team 09: *Manish Chugani, Yeahn Kim, Jesus Martinez Vega*

1. How does a system call execute? Explain briefly the steps from making the call in the user space process to returning from the call with a result.

A: User-space Call:

- The user-space program (test_count_tasks.c) makes a system call using the syscall function.
- The system call number *451* corresponds to the custom system call *count_rt_tasks*.

User-space to Kernel Transition:

- The *syscall* function triggers a switch from user space to kernel space.
- The kernel identifies the system call based on the system call number.

Kernel-side Argument Handling:

- The kernel executes the *count_rt_tasks* system call.
- The *__sys_count_rt_tasks* function checks if the *result* pointer is valid (non-null) and returns an error if not.
- It then iterates through the task list, counting real-time tasks.

Kernel-side Data Copying:

- The *copy_to_user* function is used to copy the result (counter) from the kernel space to the user space.
- If the copying fails, an error is returned.

Return to User Space:

- The kernel returns control to the user space after completing the system call.
- The return value of the system call (0 for success, -1 for failure) is returned to the user space.

User-space Handling:

- The user-space program checks the return value of the system call.
- If the return value is -1, an error occurred, and the program handles the error accordingly.
- If the return value is 0, the program can use the result variable (task_count) to determine the count of real-time tasks.

Logging:

- The kernel prints a log message using *printk* with information about the count of real-time tasks.

Execution of the User-space Program:

- The user-space program continues its execution after the system call, printing the count of real-time tasks.

2. Take a look at the container_of macro defined in include/linux/container_of.h. What does it do and how is it implemented? Explain with an example.

A. The *container_of* macro in the *container_of.h* file is a commonly used macro in the Linux kernel for obtaining a pointer to the container structure that contains a given member. It is often used in situations where a pointer to a member of a structure is known, and one needs to find the address of the enclosing structure.

```
#define container_of(ptr, type, member) ({                              \
        void *__mptr = (void *)(ptr);                                   \
        static_assert(__same_type(*(ptr), ((type *)0)->member) ||       \
                      __same_type(*(ptr), void),                        \
                      "pointer type mismatch in container_of()");       \
        ((type *)(__mptr - offsetof(type, member))); })
```

- ptr: The pointer to the member whose container structure we want to find.
- type: The type of the container structure that contains the member.
- member: The name of the member within the container structure.

void *__mptr = (void *)(ptr);: This line casts the pointer ptr to a void* and assigns it to the variable __mptr.

static_assert(__same_type(*(ptr), ((type *)0)->member) || __same_type(*(ptr), void), ...);: This line uses a static_assert to check whether the type of the member matches the expected type. It compares the type of the member against the type of the member as defined in the container structure (((type *)0)->member). This helps catch potential type mismatches at compile time.

((type *)(__mptr - offsetof(type, member)));: This line calculates the address of the container structure by subtracting the offset of the member within the container structure (offsetof(type, member)) from the address of the member (__mptr). The result is cast to the type of the container structure.

```c
struct ExampleStruct {
    int a;
    float b;
    char c;
};

int main() {
    struct ExampleStruct myStruct;
    float *ptr = &myStruct.b;

    struct ExampleStruct *container = container_of(ptr, struct ExampleStruct, b);

    printf("Address of container structure: %p\n", (void *)container);

    return 0;
}
```

In this example, *ptr* is a pointer to the *float member b* within struct *ExampleStruct*. The *container_of macro* is then used to find the address of the entire structure that contains this member (*struct ExampleStruct*). The resulting container pointer can be used to access other members of the structure.

3. Contributions of each member:
    - Manish Chugani: Manish was responsible for setting up the Part 1 environment for testing purposes. Manish was also responsible for generating the report and attaching screenshots of examples and output proofs from the Raspberry Pi. Finally, Manish attempted the debugging of the *mod_count_tasks.c* file to insert the LKM that modified the system call.
    - Yeahn Kim: Yeahn was responsible for writing the user-space applications and LKM modules. Yeahn also assisted with the setup of the environment and debugging of the LKM module *mod_count_tasks.c*. Yeahn also contributed to the coordination of the team logistics.
    - Jesus Martinez Vega: Jesus was responsible for designing the system call from scratch. Jesus managed all the files that were required to be changed in the kernel to add the system call successfully. Alongside the other two members, Jesus was also responsible for figuring out the LKM *mod_count_tasks.c* that modifies the system call.

OUTPUT from the Raspberry Pi 4:

4.1, 4.2:

```
root@raspberrypi:~/proj1_output# ls
lkmhello.ko  mod_count_tasks.ko  test_count_tasks  test_count_tasks.c  usrhello
root@raspberrypi:~/proj1_output# ./usrhello
Hello world! team09 in user space
root@raspberrypi:~/proj1_output# insmod lkmhello.ko
root@raspberrypi:~/proj1_output# lsmod | grep lkm
lkmhello               16384  0
root@raspberrypi:~/proj1_output# lsmod | grep hello
lkmhello               16384  0
root@raspberrypi:~/proj1_output# dmesg | tail
[ 1786.833263] Hello world! team09 in kernel space
root@raspberrypi:~/proj1_output# insmod lkmhello.ko
insmod: ERROR: could not insert module lkmhello.ko: File exists
root@raspberrypi:~/proj1_output# rmmod lkmhello
root@raspberrypi:~/proj1_output# dmesg | tail
[ 1786.833263] Hello world! team09 in kernel space
```

```
root@raspberrypi:~/proj1_output# insmod lkmhello.ko
root@raspberrypi:~/proj1_output# dmesg | tail
[ 6194.427176] Hello world! team09 in kernel space
root@raspberrypi:~/proj1_output# rmmod lkmhello
root@raspberrypi:~/proj1_output# dmesg | tail
[ 6194.427176] Hello world! team09 in kernel space
root@raspberrypi:~/proj1_output# insmod lkmhello.ko
root@raspberrypi:~/proj1_output# dmesg | tail
[ 6194.427176] Hello world! team09 in kernel space
[ 6222.331923] Hello world! team09 in kernel space
root@raspberrypi:~/proj1_output# rmmod lkmhello
root@raspberrypi:~/proj1_output#
```

***4.3*:**

```
root@ubuntu-vm:/home/ubuntu# ssh root@192.168.0.112
root@192.168.0.112's password:
Linux raspberrypi 6.1.73-RTES-rt22-v8 #8 SMP PREEMPT_RT Sat Feb  3 20:43:20 PST 2024 aarch64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sat Feb  3 22:13:57 2024 from 192.168.0.107
root@raspberrypi:~# uname -a
Linux raspberrypi 6.1.73-RTES-rt22-v8 #8 SMP PREEMPT_RT Sat Feb  3 20:43:20 PST 2024 aarch64 GNU/Linux
root@raspberrypi:~# cd proj1_output/
root@raspberrypi:~/proj1_output# dmesg --clear
root@raspberrypi:~/proj1_output# ./test_count_tasks
76
root@raspberrypi:~/proj1_output# insmod mod_count_tasks.ko
root@raspberrypi:~/proj1_output# ./test_count_tasks
5
root@raspberrypi:~/proj1_output# rmmod mod_count_tasks
root@raspberrypi:~/proj1_output# ./test_count_tasks
76
root@raspberrypi:~/proj1_output# insmod mod_count_tasks.ko
root@raspberrypi:~/proj1_output# ./test_count_tasks
5
root@raspberrypi:~/proj1_output# rmmod mod_count_tasks
root@raspberrypi:~/proj1_output# ./test_count_tasks
76
root@raspberrypi:~/proj1_output# dmesg | tail
[   51.629506] count_rt_tasks: 76
[   60.536907] mod_count_tasks: loading out-of-tree module taints kernel.
[   62.138022] count_rt_tasks[MOD]: 5
[   68.679807] count_rt_tasks: 76
[   75.788576] count_rt_tasks[MOD]: 5
root@raspberrypi:~/proj1_output#
```

The final *./test_count_tasks* command does not get flushed to the *dmesg* log since the kernel buffers messages to be pushed to the log.