CS 251 / EE 255 REAL-TIME EMBEDDED SYSTEMS
HW Project #1: Introduction to Linux Kernel Programming

Group16

Group members:
Hengshuo Zhang hzhan402@ucr.edu
Zhaoze Sun zsun114@ucr.edu

Writeup

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

A system call is a request made by a userspace process to the operating system for a service. When we need to invoke a system call, system need to perform a "mode switching", process implement a software interrupt or issues a trap to the operating system. The steps to execute the system call are as follows:

- The process makes a system call by invoking a specific instruction, usually the "syscall" instruction, with the system call number and its arguments.
- The processor switches from user mode to kernel mode, which is privileged mode that allows the operating system to execute operations that are not accessible from user mode.
- The system call handler, implemented in the Linux kernel, is executed. It performs any necessary parameter validation and checks the caller's privileges.
- The system call handler invokes the appropriate kernel function to perform the requested service.
- The kernel function performs the necessary operations and sets the return value for the system call.
- The system call handler returns control to the user process by switching from kernel mode back to user mode.
- The user process continues execution with the result of the system call, which is stored in a specific register or memory location.

2. What does it mean for a kernel to be preemptive? Is the Linux kernel you are hacking on preemptive?

A preemptive kernel is a type of operating system kernel that allows multiple processes to execute concurrently and supports the interruption of one process by another with higher priority. In a preemptive kernel, the operating system can interrupt a running process and schedule another process to run at any time, regardless of whether the current process has completed its execution or not. For execution scheduling, there are different algorithms to optimize the order of process scheduling. There are static scheduling and dynamic scheduling. The most typical preemptive algorithm for static scheduling is Round Robin; the preemptive algorithms for dynamic scheduling are Earliest Deadline First (EDF) and Deadline Monotonic (DM).

The Linux kernel is a preemptive kernel, meaning it allows multiple processes to run simultaneously and supports process preemption. This means that the Linux kernel can interrupt a running process and schedule another process to run, even if the current process has not completed its execution. This provides good support for multitasking and enables the operating system to respond to high-priority tasks and events quickly. For example, in the real-time priority covered in Project 4.2, real-time priority processes are processes that have a higher priority compared to normal processes in an operating system. Processes are assigned a priority value, with higher numbers(99) indicating higher priority. Real-time priority processes have the highest priority values and are guaranteed to be executed promptly and in a timely manner.

3. When does access to data structures in userspace applications need to be synchronized?

Access to data structures in user-space applications needs to be synchronized when multiple threads or processes access and modify the same data structure simultaneously. Without synchronization, multiple threads can access the same data structure at the same time, leading to inconsistencies and corruption of data. Multiple threads in a single process: When multiple threads within a single process access and modify the same data structure, synchronization is necessary to ensure that data consistency is maintained and that modifications by one thread do not interfere with modifications by another thread.

4. What synchronization mechanisms can be used to access shared kernel data structures safely? List them.

To synchronize access to data structures, user-space applications use synchronization primitives such as semaphores, mutexes, spin lock, read-copy update, or atomic operations. These primitives allow the user-space application to lock and unlock access to the data structure safely, ensuring that only one thread or process can access and modify the data structure at a time.

Semaphores: Semaphores are a type of synchronization mechanism that allow multiple threads to access a shared resource, but limit the number of threads that can access it simultaneously. They work by counting the number of available resources and blocking threads that try to access the resource when it is unavailable.

Mutexes: Mutexes are a type of synchronization mechanism that allow only one thread to access a shared resource at any given time. They work by blocking all other threads that try to access the resource until the first thread has released the lock.

Atomic operations: Atomic operations are a type of synchronization mechanism that allow multiple threads to access a shared resource simultaneously, but ensure that the resource remains consistent even in the presence of concurrent access. They are implemented using low-level instructions that are guaranteed to be executed atomically, meaning that they cannot be interrupted by other threads.

Spin lock: A spin lock is a synchronization primitive used in concurrent programming to control access to a shared resource by multiple threads. It uses a busy waiting mechanism, where a thread repeatedly checks a flag until it becomes available, and then sets the flag to indicate that it has taken the lock. Once the thread is finished with the resource, it clears the flag to allow other threads to take the lock. Only on multiprocessors system.

Read-Copy Update: RCU is a synchronization mechanism used in the Linux kernel to ensure safe access to shared data structures by multiple threads. RCU works by allowing multiple threads to continue reading a shared data structure while an update is in progress. This allows for parallel access to the shared data by multiple threads and eliminates the need for locks, which can cause performance problems in high-concurrency systems.

5. Take a look at the `container_of` macro defined in *include/linux/kernel.h*. In rough terms, what does it do and how is it implemented?

```
#define container_of(ptr, type, member) ({
     const typeof(((type *)0)->member) * __mptr = (ptr);
     (type *)((char *)__mptr - offsetof(type, member));
})
```

The container_of macro The container_of is used to cast a member of a structure out to the containing structure. It accepts a pointer to a member of the structure and returns a pointer to the structure containing the member. The ptr is a pointer to the member and type is the type of the container struct this is embedded in. The last one part is the member which is a name of member within this struct.

```
const typeof(((type *)0)->member) * __mptr = (ptr);
```

This line creates a local constant pointer variable. The type of this variable is determined using the `typeof` operator.

```
(type *)((char *)__mptr - offsetof(type, member));
```

This `offsetof` macro is used to calculate the offset of a member within a structure, it returns the byte offset of the member to the beginning of the structure.

`__mptr` subtracts the address from the offset to get the starting address of the structure.

6. Give a brief summary of the contributions of each member.

Overall:

Hengshuo Zhang	50%
Zhaoze Sun	50%
Tianze Wu	0%

Breakdown to each Assignment

4.1	Hengshuo/Zhaoze
4.2	Hengshuo/Zhaoze
4.3	Hengshuo/Zhaoze
4.4	Hengshuo/Zhaoze

The content of the project is relevant and we have been working together on it since the first part. There is a lot of discussion and mutual assistance when we can't get a solution to our respective problems during the project. We also made use of the resources given by our professors and TAs to help us solve all the problems we encountered during the project.

Most of the content was solved with mutual discussion and help, and we consider the contribution of the project as average.

Intra-group fact sheet:

Until noon on 2/4/2023, member Tianze Wu, did not share anything related to the project (code/Writeup) with the group members and did not provide any help in moving the project forward. We communicated with Tianze several times about the project and urged him to share the available resources with the group members, but we did not get any substantive response.

IIt is unacceptable for anyone to be unable to submit any project-related content, regardless of the circumstances that ultimately prevent them from doing so.

I have obtained permission from another team member, Zhaoze Sun, to bring this to your attention. We also do not wish to share any of the content or results of Project 1 with Tianze Wu and hope to not include Tianze Wu in subsequent projects.