Operating System-Assignment 1

314551128 張哲榕

Requirement 1

Step	Command	Description
Step1	ARCH=riscv CROSS_COMPILE=riscv64-linux-gnu-	Set the environment variable to riscv environment
Step2	make defconfig	Generate .config
Step3	make menuconfig	Set the Local version to “-os-314551128”
Step4	make -j$(nproc)	Parallel compilation

```
~ # uname -a
Linux (none) 6.1.0-os-314551128 #1 SMP Tue Sep 30 06:26:08 UTC 2025 riscv64 GNU/Linux
~ # cat /proc/version
Linux version 6.1.0-os-314551128 (root@88debd3be653) (riscv64-linux-gnu-gcc (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0,
GNU ld (GNU Binutils for Ubuntu) 2.42) #1 SMP Tue Sep 30 06:26:08 UTC 2025
```

- Q&As

  1. What are the main differences between the RISC-V and x86 architectures?
  A: RISC-V uses Reduced Instruction Set, and x86 uses Complex Instruction Set.

  2. Why do the architecture differences matter when building the kernel? What happens if you build the kernel without the correct RISC-V cross-compilation flag?
  A: Different architecture has their own ISA, if the flag is set to wrong architecture, the machine cannot execute kernel, so we may encounter “failure to boot” or something like that.

  3. Why is Docker used in this assignment, and what advantages does it provide? Please list at least two of them.
  A: Docker container has great isolation and it encapsulates the software and its dependencies, so a container can run on any computer that has Docker.

Requirement 2

- sys_revstr

```
/bin # ./test_revstr
Ori: hello
Rev: olleh
Ori: Operating System
Rev: metsyS gnitarepO
```

```
[    7.569691] The origin string: hello
[    7.569858] The reversed string: olleh
[    7.570391] The origin string: Operating System
[    7.570404] The reversed string: metsyS gnitarepO
```

- sys_tempbuf

```
/bin # ./test_tempbuf
Hello Operating Systems
Operating Systems
```

```
[  196.462176] [tempbuf] Added: Hello
[  196.462337] [tempbuf] Added: Operating Systems
[  196.462491] [tempbuf] Hello Operating Systems
[  196.464157] [tempbuf] Removed: Hello
[  196.464246] [tempbuf] Operating Systems
```

- The process of adding system calls

Step	Code or Operation	File or Directory

Step1	asmlinkage long sys_revstr(char __user *str, size_t n);
	
	asmlinkage long sys_tempbuf(int m, char __user *str, size_t n);
		linux/include/linux/ syscalls.h

Step2	#define __NR_revstr 451 __SYSCALL(__NR_revstr, sys_revstr) #define __NR_tempbuf 452 __SYSCALL(__NR_tempbuf, sys_tempbuf) #undef __NR_syscalls #define __NR_syscalls 453	linux/include/uapi/ asm-generic/unistd.h
Step3	Finish revstr.c & tempbuf.c	linux/kernel
Step4	make ARCH=riscv CROSS_COMPILE=riscv64-linux-gnu- -j$(nproc)	linux

- The implementation details of sys_revstr & sys_tempbuf
  1. sys_revstr : The most import detail is that char* in the argument list should be declare as char __user *, and get the string by using copy_from_user(). Also, I implement error detection to deal with errors which may caused by copy_from_user().
  2. sys_tempbuf : In this system call, I use a built-in linked list structure to store

every string passed by "ADD" operation. For "REMOVE" operation, I use built-in iteration function to go through the list to find the string required to delete. And for "PRINT" operation, I use the same iteration function mentioned above, but use a string to concatenate every string in the list. Moreover, there are some error checking during each operation, for example, -EFAULT is used to detect invalid address and -ENOENT is used to imform there's no entry for the structure. Lastly, the list I used in this system call is binded to user process, then I can delete the node that does not belong to caller before every operations, since the list would still exist if we don't unload the module maually, and the rest nodes of last caller would be printed by "PRINT" operation which is called by current caller.

- Q&As

  1. What does the include/linux/syscalls.h do?
  A: It declares system call function's prototype, and provides the "SYSCALL_DEFINE" macro.

  2. Explain the difference between a system call and a glibc library call, then give one example that maps a glibc function to the specific system call it ultimately invokes.
  A: A system call only runs in kernel space, and it's a low-level interface, it is related to accessing system stuff. While a glibc library call is a high-level interface, and it provides convenient, abstracted service for programmer. And the example that maps a glibc function to the specific system call is fopen(), it's a glibc function call but eventually invokes sys_open() to handle file access.

  3. Explain the difference between static linking and dynamic linking.
  A: Static linking is occurred during compilation, and dynamic linking happens during execution.

  4. In this assignment environment, why do we have to compile the test programs with the -static flag? What would happen if we omitted it?
  A: The -static flag tells the compiler to use static linking to link the function used in the program. If we omitted it, the executable may not be able to execute because of missing shared object.