Lecture Note 5.2: Implementing User Programs in xv6-riscv

0. Overview

By the end of this lecture, we should be able to:

- ➔ Explain how user programs fit into the xv6-riscv system.
- ➔ Write a simple xv6 user program without command-line parameters that performs basic mathematical operations.
- ➔ Write a simple xv6 user program with command-line parameters that behaves as a small calculator.
- ➔ Integrate new user programs into the xv6 build system using the UPROGS list in the Makefile.
- ➔ Compile, run, and test these programs inside the xv6 shell.

We will use two example programs in this lecture: one without any arguments or parameters and another with arguments or parameters.

1. How user programs fit into xv6

Before writing any code, it is important to understand where user programs live and how they are built in xv6-riscv.

1.1 Where user programs live

In xv6-riscv:

- ➔ All user programs are placed in the **user/** directory.
- ➔ The xv6 file system image is populated with binaries built from these source files.

1.2 How user programs are built

The top-level Makefile contains a variable UPROGS that lists all user programs that should be compiled and included in the file system image. A typical fragment looks like:

```
UPROGS=\

        $U/_cat\
        $U/_echo\
        $U/_forktest\
        $U/_grep\
        $U/_init\
        $U/_kill\
        $U/_ln\
        $U/_ls\
        $U/_mkdir\
        $U/_rm\
        $U/_sh\
```

For each entry **$U/_progname**, xv6 expects a corresponding C source file user/progname.c. The build system compiles this file into a binary named _progname and places it into the file system image. Inside the xv6 shell, the user runs the program using the name without the underscore, for example, cat, echo, or sh.

1.3 Standard includes for user programs

Most user programs in xv6 start with the following two includes:

```
#include "kernel/types.h"
#include "user/user.h"
```

kernel/types.h provides type definitions, while user/user.h exposes the interfaces for basic library functions and system calls such as printf, fprintf, exit, atoi, and strcmp.

2. Program 1: addDemo.c – user program without parameters

The first example is a simple program called **addDemo.c**. It calculates the sum of two integers, 2 and 3, stores the result in the variable add, and then prints the result of the addition.

Step 1: Create user/addDemo.c

```
// user/addDemo.c

#include "kernel/types.h"
#include "user/user.h"

int main(){
    int add = 2+3;
    printf("add: %d\n",add);

    return 0;
}
```

Step 2: Add _addDemo to UPROGS

To tell xv6 to build and include the new program, add an entry for **_addDemo** to the UPROGS list in the Makefile:

```
UPROGS=\

      $U/_cat\
      $U/_echo\
      $U/_forktest\
      $U/_grep\
      $U/_init\
      $U/_kill\
      $U/_ln\
      $U/_ls\
	$U/_mkdir\
	$U/_rm\
	$U/_sh\
	$U/_stressfs\
	$U/_usertests\
	$U/_grind\
	$U/_wc\
	$U/_zombie\
	$U/_logstress\
	$U/_forphan\
	$U/_dorphan\
	$U/_addDemo\
```

The pattern rule in the Makefile will now compile **user/addDemo.c** into a binary named **_addDemo**. Inside the xv6 shell, the user will invoke this program simply as **addDemo**.

Step 3: Build and test addDemo

From the root of the xv6 source tree, rebuild and run the system:

```
$ make qemu
```

After xv6 boots and the shell prompt appears, run:

```
$ addDemo
```

Inside the xv6 shell, test **addDemo** which will always provide the following output:

```
$ addDemo
add: 5
```

3. Program 2: add_with_arg.c – user program with parameters

The second example, **add_with_arg.c**, is a program that accepts command-line arguments. It performs addition of two integers which are taken as command-line inputs. It expects exactly two input integer values; if fewer than two or more than two arguments are provided, it prints a message indicating failure of the addition due to missing or extra arguments. When exactly two command-line inputs are given as integers, the program adds them, and prints the result of the addition.

Step 1: Create user/add_with_arg.c

```c
// user/add_with_arg.c

#include "kernel/types.h"
#include "user/user.h"

int main(int argc, char *argv[]){
    if(argc < 3){
        printf("Addition failed_____No arguments given_____\n");
    }
    else if(argc == 3){
        int a = atoi(argv[1]);
        int b = atoi(argv[2]);
        int add = a + b;
        printf("addition of 2 inputs %d and %d = %d\n",a,b,add);
    }
    else{
        printf("Addition failed_____Too many arguments given_____\n");
    }

    return 0;
}
```

Step 2: Add _add_with_arg to UPROGS

To tell xv6 to build and include the new program, add an entry for **_add_with_arg** to the UPROGS list in the Makefile:

```
UPROGS=\

	$U/_cat\
	$U/_echo\
	$U/_forktest\
	$U/_grep\
	$U/_init\
	$U/_kill\
	$U/_ln\
	$U/_ls\
	$U/_mkdir\
	$U/_rm\
	$U/_sh\
	$U/_stressfs\
	$U/_usertests\
	$U/_grind\
	$U/_wc\
	$U/_zombie\
	$U/_logstress\
	$U/_forphan\
	$U/_dorphan\
	$U/_addDemo\
	 $U/_add_with_arg\
```

The pattern rule in the Makefile will now compile **user/.c** into a binary named **_add_with_arg**. Inside the xv6 shell, the user will invoke this program simply as **add_with_arg**.

Step 3: Build and test add_with_arg

From the root of the xv6 source tree, rebuild and run the system:

```
$ make qemu
```

After xv6 boots and the shell prompt appears, run:

```
$ add_with_arg 4 6
```

Inside the xv6 shell, test **add_with_arg** which will always provide the following outputs:

```
$ add_with_arg 4 6
addition of 2 inputs 4 and 6 = 10
```

Now test **add_with_arg** with various inputs inside xv6 shell:

```
$ add_with_arg
Addition failed_____No arguments given_____
$ add_with_arg 3
Addition failed_____No arguments given_____
$ add_with_arg 9 8
addition of 2 inputs 9 and 8 = 17
$ add_with_arg 7 8 10
Addition failed_____Too many arguments given_____
```

4. Summary

- ➔ The lecture explains how xv6 builds and runs user programs from the user/ directory and the UPROGS list in the Makefile.
- ➔ Implemented addDemo, a program without parameters.
- ➔ Implemented add_with_arg, a program that takes two command-line integer inputs and performs addition.
- ➔ Practised compiling, running, and testing programs inside the xv6 shell.