

UP24 Lab02

Date: 2024-03-04

- UP24 Lab02
- Random Walk in the Kernel
 - Preparation
 - Specification
 - Lab Hints
 - Grading

Random Walk in the Kernel

This lab aims to practice implementing a character device as a kernel module that handles `read`, `write`, and `ioctl` operations. The character device has to implement several required features to construct and run a maze in the Linux kernel. We have defined the required interfaces offered by the kernel module and prepared a testcase program for evaluating your kernel module.

Preparation

Please read our Lab02 Pre-Lab Announcement (https://md.zoolab.org/s/9TrI_zswV) for the details. You should have at least the `dist-6.6.17.tbz` and the `hellomod-6.6.17.tbz` files. You also need the QEMU emulator to run the files. See the Lab Hints section for more details.

Specification

The specification of the kernel module is summarized as follows.

1. The module has to **automatically** create one device named `maze` in the `/dev` filesystem. Each process can create only one maze at the same time. Once a maze is created, the process can get the size of the maze, the position of the player (current position, start position, end position) in the maze, and the layout of the maze. When the device is opened by a process, the process can interact with the module using `read`, `write`, and `ioctl` operations. Details of each of the commands are summarized as follows.
2. The `ioctl` interface. The `ioctl` command supports the following commands. The

definition of the commands can be obtained from the header file `maze.h` ([view \(https://up.zoolab.org/code.html?file=unixprog/lab02/maze.h\)](https://up.zoolab.org/code.html?file=unixprog/lab02/maze.h) | [download \(https://up.zoolab.org/unixprog/lab02/maze.h\)](https://up.zoolab.org/unixprog/lab02/maze.h)).

- `MAZE_CREATE` : Create a new maze. The size of the maze is passed as a pointer to a `coord_t` structure. Ideally, the layout of the maze should be randomly generated, but it is OK if you use a fixed one for some of the testcases. You have to ensure that the borders of the maze are all walls.

The command may return the following error values:

- `-EINVAL` : returned when any argument has an invalid value.
 - `-EEXIST` : returned when a maze has already been created for the calling process.
 - `-ENOMEM` : returned when there are already `_MAZE_MAXUSER` (3) mazes created in the system.
 - `-EBUSY` : returned when copy data from/to user-space is failed.
- `MAZE_RESET` : Reset the position of the player to the start position. The command may return the following error values:
 - `-ENOENT` : returned when no maze is created for the calling process.
 - `MAZE_DESTROY` : Destroy a maze if it has been created. The command may return the following error values:
 - `-ENOENT` : returned when no maze is created for the calling process.
 - `MAZE_GETSIZE` : Get the dimension of the maze. The result is stored to the pointer of a `coord_t` structure. The command may return the following error values:
 - `-ENOENT` : returned when no maze is created for the calling process.
 - `-EBUSY` : returned when copy data from/to user-space is failed.
 - `MAZE_MOVE` : Move the player position on the maze. The movement is passed as a pointer to a `coord_t` structure. The valid values only include `(-1, 0)`, `(1, 0)`, `(0, -1)`, and `(0, 1)`. You should not move the position of the player if the walk is invalid, e.g., hit a wall.

In case an invalid movement is requested, simply ignore it and return zero (OK) to the user. The command may return the following error values:

- `-ENOENT` : returned when no maze is created for the calling process.
- `-EBUSY` : returned when copy data from/to user-space is failed.

- MAZE_GETPOS : Get the player's position on the maze. The result is stored to the pointer of a `coord_t` structure. The command may return the following error values:
 - -ENOENT : returned when no maze is created for the calling process.
 - -EBUSY : returned when copy data from/to user-space is failed.
- MAZE_GETSTART : Get the start position of the maze. The result is stored to the pointer of a `coord_t` structure. The command may return the following error values:
 - -ENOENT : returned when no maze is created for the calling process.
 - -EBUSY : returned when copy data from/to user-space is failed.
- MAZE_GETEND : Get the end position of the maze. The result is stored to the pointer of a `coord_t` structure. The command may return the following error values:
 - -ENOENT : returned when no maze is created for the calling process.
 - -EBUSY : returned when copy data from/to user-space is failed.

3. The `read` interface. A process can obtain the layout of a maze using the `read` operation. Given a layout of a 5x5 maze as follows:

```
1 1 1 1 1
1 0 0 0 1
1 0 0 0 1
1 0 0 0 1
1 1 1 1 1
```

The `read` operation should return a byte sequence of 25 bytes containing the content [1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1]. Note that the numbers are numeric numbers instead of character symbols.

The function may return the following error numbers:

- -EBADFD : returned when there is not a maze associated with the current process, i.e., the process has not used `MAZE_CREATE` command to create a maze.
- -EBUSY : returned when copy data from/to user-space is failed.

4. The `write` interface. In addition to the `MAZE_MOVE` command in the `ioctl` interface, a process can move the current positoin of the player in the maze in a batch

manner.

To achieve the goal, simply declare an array of `coord_t` and place the movement coordinates iteratively in the array. Once you are done, issue a written request to the kernel module by writing everything in the array to the kernel.

The function may return the following error numbers:

- `-EBADFD` : returned when there is not a maze associated with the current process, i.e., the process has not used `MAZE_CREATE` command to create a maze.
- `-EINVAL` : returned when the length of passed user data is not multiples of `sizeof(coord_t)` .
- `-EBUSY` : returned when copy data from/to user-space is failed.

5. You also have to offer the `/proc/maze` interface. The content of this file is the status of all the mazes created by user space processes. When no maze is created in the system, the content of the `/proc/maze` should look like the following:

```
#00: vacancy
```

```
#01: vacancy
```

```
#02: vacancy
```

We have only three slots, and they have yet to be used. If one is in use, the output should look like the following:

```
#00: pid 66 - [33 x 5]: (31, 1) -> (31, 3) @ (31, 1)
- 000: #####
- 001: #.#.....#.....#.....*#
- 002: #.###.#####.#.###.#####.###.#
- 003: #.....#.....#.....#..E#
- 004: #####
```

The first line of a slot contains the pid of the process created the maze, the size of the maze, the player's start and end position, and the player's current position. The following few lines contain the layout of the maze with the following symbols:

- `#` : wall
- `.` : road
- `s` : start position of the player (may be `*` if it's overlapped with the current

position)

- `E` : end position of the player (may be `*` if it's overlapped with the current position)
- `*` : current position of the player

6. Please release all allocated resources for a process when the device is closed by the process, or the process is terminated.
7. To simplify the implementation, the kernel module can handle at most `_MAZE_MAXUSER` (3) mazes concurrently. Each process can create only one maze at the same time.

Lab Hints

Here are some hints for you.

1. Please install the qemu system emulator in your development platform. For Ubuntu-based dockers, you can install it using the command `apt install qemu-system-x86`. It would work on both Intel and Apple chips. You can even install the native one on Mac by using `brew install qemu`.
2. Once you have the qemu system emulator, you can simply type `sh ./qemu.sh` to boot the Linux kernel in a virtual machine. The current design uses the archive `rootfs.cpio.bz2` as the `initramfs` image. You can add more files in the filesystem by extracting files from the archive, adding files you want, and re-packing the archive.
3. If you plan to have your files in the `initramfs` image, you can extract the files using `bzip2(1)` (<https://linux.die.net/man/1/bzip2>) and `cpio(1)` (<https://linux.die.net/man/1/cpio>) utilities, and re-pack the image using the same tools.

You may need to set the cpio format to `newc` format. Also please ensure that you pack all the required files in the image.

4. A sample `hello, world!` module is available here (`hellomod.tbz`) (<https://up.zoolab.org/unixprog/lab02/hellomod-6.6.17.tbz>). You may implement your module based on the `hello, world!` example. It has sample file operations and `/proc` file system implementations.

In the `qemu` virtual machine runtime, you can use the commands `insmod` and `rmmod` to install and remove modules, respectively. Use `lsmod` to inspect what modules have been loaded.

5. To copy memory content from the user-space process to the kernel, please use the `copy_from_user` (<https://elixir.bootlin.com/linux/v6.6.17/source/include/linux/uaccess.h#L180>) function. To copy memory content from the kernel to the user-space process, please use the `copy_to_user` (<https://elixir.bootlin.com/linux/v6.6.17/source/include/linux/uaccess.h#L188>) function.
6. To generate a random integer, you may consider using the function `get_random_u32()` (<https://elixir.bootlin.com/linux/v6.6.17/source/include/linux/random.h#L42>) function. Note that it returns a 32-bit unsigned integer.
7. To lock shared resources between processors, you may use the `mutex_lock` (<https://elixir.bootlin.com/linux/v6.6.17/source/include/linux/mutex.h#L200>) and the `mutex_unlock` (<https://elixir.bootlin.com/linux/v6.6.17/source/include/linux/mutex.h#L219>) functions. The lock can be declared as a static global variable using the macro `DEFINE_MUTEX(name_of_the_lock_variable)`.
8. (Optional, not really required) You may want to have **private data** associated with an opened file. To do this, you will need to ...
 - Define a customized data structure for your private data.
 - When opening a file, allocate a memory space using `kzalloc` (<https://elixir.bootlin.com/linux/v6.6.17/source/include/linux/slab.h#L718>) for the customized data structure.
 - Assign the pointer of the allocated space to `file->private_data`.
 - Remember to release the spaces of the `file->private_data` using the `kfree` (<https://elixir.bootlin.com/linux/v6.6.17/source/include/linux/slab.h#L227>) function.
9. We have implemented a testcase program to test your module. The testcase program contains 7 testcases (with an id ranges from 0 to 6). Once your module is installed into the kernel (using `insmod`), run the `mazetest` program binary with the testcase-id as its argument. Sample outputs can be found in the (Grading)[#Grading] section. The program can be downloaded from here (mazetest) (<https://up.zoolab.org/unixprog/lab02/mazetest>). Part of its source code is available here (mazetest.c) (<https://up.zoolab.org/code.html?file=unixprog/lab02/mazetest.c>).

Grading

We have many test cases here. You don't have to complete them in order. Just demonstrate what you have completed. Before you run the `mazetest` testcase program, please ensure its md5 checksum is `94529d9dcf761eedd703f3fb00e3650d`.

1. [10 pts] You can boot the Linux VM in a supported environment.
2. [10 pts] You can put your `maze.ko` module and the `mazetest` binary into the emulator. Load the module and it should automatically create the required `/dev/`

maze and /proc/maze files.

3. [10 pts] The maze layout and the player's positions are all randomly generated. You have to ensure there is a valid path from the start position to the end position. If you skip this scoring item, you have to generate an empty maze with only the outmost walls. For example, a 11x7 maze would look like:

```
#####
#.....#
#.....#
#.....#
#.....#
#.....#
#.....#
#####
```

4. [10 pts] Run and pass `mazetest 0`. A sample output is pasted here for your reference.

```
### case - cat /proc/maze ###
#00: vacancy

#01: vacancy

#02: vacancy
```

5. [10 pts] Run and pass `mazetest 1`. A sample output is pasted here for your reference.

```
### case - illegal operations (w/o create first) ###
OP: MAZE_CREATE -1 -1
mz_create(-1, -1): Invalid argument
OP: MAZE_GETSIZE
mz_get_size: No such file or directory
OP: MAZE_GETPOS
mz_get_pos: No such file or directory
OP: MAZE_GETSTART
mz_get_start: No such file or directory
OP: MAZE_GETEND
mz_get_end: No such file or directory
OP: MAZE_MOVE 0 -1
mz_move: No such file or directory
```

6. [10 pts] Run and pass `mazetest 2`. A sample output is pasted here for your reference.

```

### case - create a maze ###
OP: MAZE_CREATE 19 19
- create maze done.
#00: pid 75 - [19 x 19]: (3, 11) -> (17, 7) @ (3, 11)
- 000: #####
- 001: #.....#
- 002: #####.###.#.#####.#
- 003: #...#...#.#.#.#...#
- 004: #.#.###.###.#.#.###
- 005: #.#...#...#.#...#.#
- 006: #####.###.#.#.###.###
- 007: #.....#...#.#.#..E#
- 008: #.###.#.###.#.###.#
- 009: #...#.#.#...#.....#
- 010: #.#.###.###.#####.###
- 011: #.#*...#.#...#.....#
- 012: #.###.#.#.###.#####
- 013: #...#.#.#...#.....#
- 014: ###.#.#.###.#####.###
- 015: #.#.#.#.#...#.#...#
- 016: #.#.#.#.#.###.#.###
- 017: #...#.....#.....#
- 018: #####

#01: vacancy

#02: vacancy

OP: MAZE_GETSIZE
==> SIZE: 19 19
OP: MAZE_GETSTART
==> START: 3 11
OP: MAZE_GETEND
==> END: 17 7
OP: MAZE_GETPOS
==> POS: 3 11
OP: MAZE_DESTROY

```

7. [10 pts] Run and pass `mazetest 3` . A sample output is pasted here for your reference.


```

### case - create a maze and then move ###
OP: MAZE_CREATE 33 7
- create maze done.
#00: pid 77 - [33 x 7]: (29, 3) -> (15, 1) @ (29, 3)
- 000: #####
- 001: #.#.....#E..#...#...#.....#
- 002: #.#####.#####.#.#.#.#.#.#####
- 003: #.....#.#...#.#.#.#.#...#.#..*..#
- 004: #####.#.#.####.#.#.#.#####.#####.#
- 005: #.....#.....#...#.....#
- 006: #####

#01: vacancy

#02: vacancy

OP: MAZE_MOVE -1 0
#00: pid 77 - [33 x 7]: (29, 3) -> (15, 1) @ (28, 3)
OP: MAZE_MOVE 1 0
#00: pid 77 - [33 x 7]: (29, 3) -> (15, 1) @ (29, 3)
OP: MAZE_MOVE 0 -1
#00: pid 77 - [33 x 7]: (29, 3) -> (15, 1) @ (29, 3)
OP: MAZE_MOVE 0 1
#00: pid 77 - [33 x 7]: (29, 3) -> (15, 1) @ (29, 3)
OP: MAZE_RESET

```

8. [10 pts] Run and pass `mazetest 4`. A sample output is pasted here for your reference.

```

### case - multiple user ###
OP: MAZE_CREATE 29 9
waiting for child processes ...
OP: MAZE_CREATE 29 9
OP: MAZE_CREATE 29 9
OP: MAZE_CREATE 29 9
child: Cannot allocate memory
#00: pid 114 - [29 x 9]: (17, 5) -> (23, 7) @ (17, 5)
- 000: #####
- 001: #.#.....#.#.....#...#
- 002: #.#.#####.#.#.#####.#.#.#
- 003: #...#...#.#...#.#.#...#...#.#
- 004: #####.#.#.#####.#.#.#.#####.#
- 005: #.....#.#.....#.*#.#.#.....#
- 006: #.#.#####.#.#.#.#.#.#####
- 007: #.#.....#...#..E....#
- 008: #####

#01: pid 115 - [29 x 9]: (25, 7) -> (3, 7) @ (25, 7)
- 000: #####
- 001: #...#.#.....#.....#...#
- 002: ###.#.#.#.###.#####.#.#.#.#
- 003: #...#...#.#.#.#.....#...#.#
- 004: #.#####.#.#.#.#####.#
- 005: #.#.....#.#.#.#.....#...#
- 006: #.###.#####.#.#.#.#.#####.###
- 007: #..E..#.....#.#.....*...#
- 008: #####

#02: pid 116 - [29 x 9]: (17, 7) -> (13, 1) @ (17, 7)
- 000: #####
- 001: #...#.....#E.....#.#
- 002: ###.###.###.#####.#.#.#
- 003: #.#...#...#.....#.....#.#.#
- 004: #.###.#.#.#####.#.#####.#.#
- 005: #.#...#.#.#.....#...#...#...#
- 006: #.#.#####.#.#.#####.#.###.#
- 007: #.....#.#.....*.....#.....#
- 008: #####

```

9. [10 pts] Run and pass `mazetest 5`. A sample output is pasted here for your reference.

```

### case - create a maze and then read ###
OP: MAZE_CREATE 35 7
- Create maze done.
#00: pid 122 - [35 x 7]: (27, 1) -> (31, 3) @ (27, 1)
- 000: #####
- 001: #.#...#.....#...#*.....#
- 002: #.#.#.#.#####.#####.#.#.#####.#.#
- 003: #...#.#.....#...#.#.#.#.....#E#.#
- 004: #####.#####.#.#.#.#####.###.#
- 005: #.....#.....#.....#.....#
- 006: #####

```

#01: vacancy

#02: vacancy

```

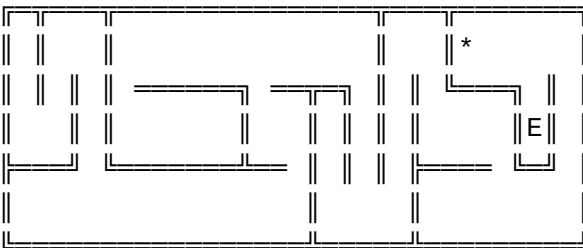
OP: MAZE_GETSIZE
OP: MAZE_GETSTART
OP: MAZE_GETEND
OP: MAZE_GETPOS
- Size [35 x 7]: (27, 1) -> (31, 3) @ (27, 1)

```

```

000:
001:
002:
003:
004:
005:
006:

```



10. [10 pts] Run and pass `mazetest 6` . A sample output is pasted here for your reference.

- Check PASSED!