Exercises — epoll

version **#dirty**

Assistants C/Unix 2024 <assistants@tickets.assistants.epita.fr>

Copyright

This document is for internal use at EPITA (website) only.

Contents

*https://intra.forge.epita.fr

File Tree

```
epoll/
├── Makefile  (to submit)
└── epoll.c  (to submit)
```

Compilation : Your code must compile with the following flags

- -std=c99 -Werror -Wall -Wextra -Wvla

Main function : Required

Makefile

- epoll: Produce the epoll binary

Authorized functions : You are only allowed to use the following functions

- read
- open
- epoll_create1
- epoll_wait
- epoll_ctl
- close

Authorized headers : You are only allowed to use the functions defined in the following headers

- assert.h
- ctype.h
- errno.h
- err.h
- fcntl.h
- stdbool.h
- stdio.h
- stdlib.h
- string.h
- sys/epoll.h

1 Epoll

1.1 Epoll

Epoll is a Linux kernel syscall that allows you to monitor multiple file descriptors and receive an event when an I/O operation is possible on them. You **SHOULD** read the `epoll(7)` man page carefully, as it contains all the information you need about this syscall.

1.1.1 Workflow

The epoll usage flow, from a high-level perspective, is as follows:

1. Create an `epoll(7)` instance with `epoll_create1(2)`.
2. Declare your interest in a file descriptor to epoll using `epoll_ctl(2)`.
3. Wait for new events using `epoll_wait(2)`.

Tips

You **SHOULD** read all syscalls manpages to understand their behaviors!

1.1.2 Edge-Triggered vs. Level-Triggered Mode

`epoll(7)` has two triggering modes that control under which conditions `epoll_wait(2)` will stop blocking.

Tips

This subject is not covered in detail here. However, you **SHOULD** read the `epoll(7)` manpage and conduct your own research to ensure you understand these two modes.

Level-Triggered Mode

In level-triggered mode, `epoll_wait(2)` always returns if at least one of the registered file descriptors is available for an I/O operation. This mode is the default behavior of `epoll(7)`.

Be careful!

A file descriptor being ready for an I/O operation does not always guarantee that there is relevant data to read from it.

Edge-Triggered Mode

Edge-triggered mode, denoted as `EPOLLET`, instructs `epoll_wait(2)` to return as soon as there is a new event available for your file descriptors.

Tips

We recommend using this mode in this exercise because it will prevent `epoll_wait(2)` from returning when your file descriptor is ready for a read operation but has no data to be read on it.

1.2 Named pipe

In this exercise we are going to monitor whether a named pipe is ready for a read operation using `epoll(7)`. You can think of named pipe being a file that hold all the data that is being written to it until we read them. Named pipes behave like FIFO s.

Tips

You can create a named pipe on your machine using the *mkfifo(1)* command

Be careful!

You need to run *mkfifo* **outside** your AFS to avoid getting errors.

1.3 Goal

The goal of this exercise is to make you practice `epoll(7)` kernel API. You **MUST** use `epoll(7)` to monitor writes on a named pipe and react to messages.

Message	Output	Exit
ping	pong!	No
pong	ping!	No
quit	quit	Yes

If you receive any other message you **MUST** print the following message on the standard output: `Unknown: <received message>` with an additionnal line feed.

Be careful!

As always you **SHOULD** check all syscalls errors and make sure your program does not leak file descriptors.

1.4 Example

Here is an example of how your program must behave:

```
42sh$ ./epoll
./epoll: Bad usage ./epoll <pipe_name>
42sh$ echo $?
1
42sh$ mkfifo my_pipe
42sh$ ./epoll my_pipe &
[1] 43945
42sh$ echo -ne 'pong' > my_pipe
ping!
42sh$ echo -ne 'yay' > my_pipe
Unknown: yay
42sh$ echo -ne 'ping' > my_pipe
pong!
42sh$ echo -ne 'quit' > my_pipe
quit
[1]  + 43945 done       ./epoll my_pipe
```

I must not fear. Fear is the mind-killer.