[toc]

课上测试

ch05

作业题目：进程间通信

完成下面任务（14分）

1 在 Ubuntu 或 openEuler 中完成任务（推荐openEuler）

2 提交运行 ls | sort -r的结果，总结管道的功能 （2分）

3 使用Linux系统调用编写实现管道（pipe）功能时，需要什么系统调用？提交man -k 相关截图。（2分）

4 使用系统调用创建一个管道，父进程向管道写入数据，子进程从管道读取数据。在父进程中使用 write 系统调用写入字符串 “你的八位学号+姓名”，并在子进程中使用 read 系统调用读取数据并打印。提交代码，编译运行过程截图（可文本）（7分）

5 提交git log结果（3分）

作业提交要求 (1')

0. 记录实践过程和 AI 问答过程，尽量不要截图，给出文本内容
1. (选做)推荐所有作业托管到 gitee或 github 上
2. (必做)提交作业 markdown文档，命名为“学号-姓名-作业题目.md”
3. (必做)提交作业 markdown文档转成的 PDF 文件，命名为“学号-姓名-作业题目.pdf”

- github链接

作业内容

提交运行 ls | sort -r的结果，总结管道的功能

- ls | sort -r的结果：系统首先运行 ls 命令获取当前目录下所有文件和子目录的列表，然后将这个列表传递给 sort -r 命令，该命令会对列表中的条目按照字母顺序进行逆序排列（Z 到 A 或 9 到 0），最后将排序后的结果输出到屏幕上。

```
root@Youer:~/shiyan/test/bestidiocs2024/ch05# ls | sort -r
thread
process
network
fs
```

- 总结管道的功能：
  - 数据传递：从一个命令的标准输出（stdout）到下一个命令的标准输入（stdin），形成一个连续的数据流。

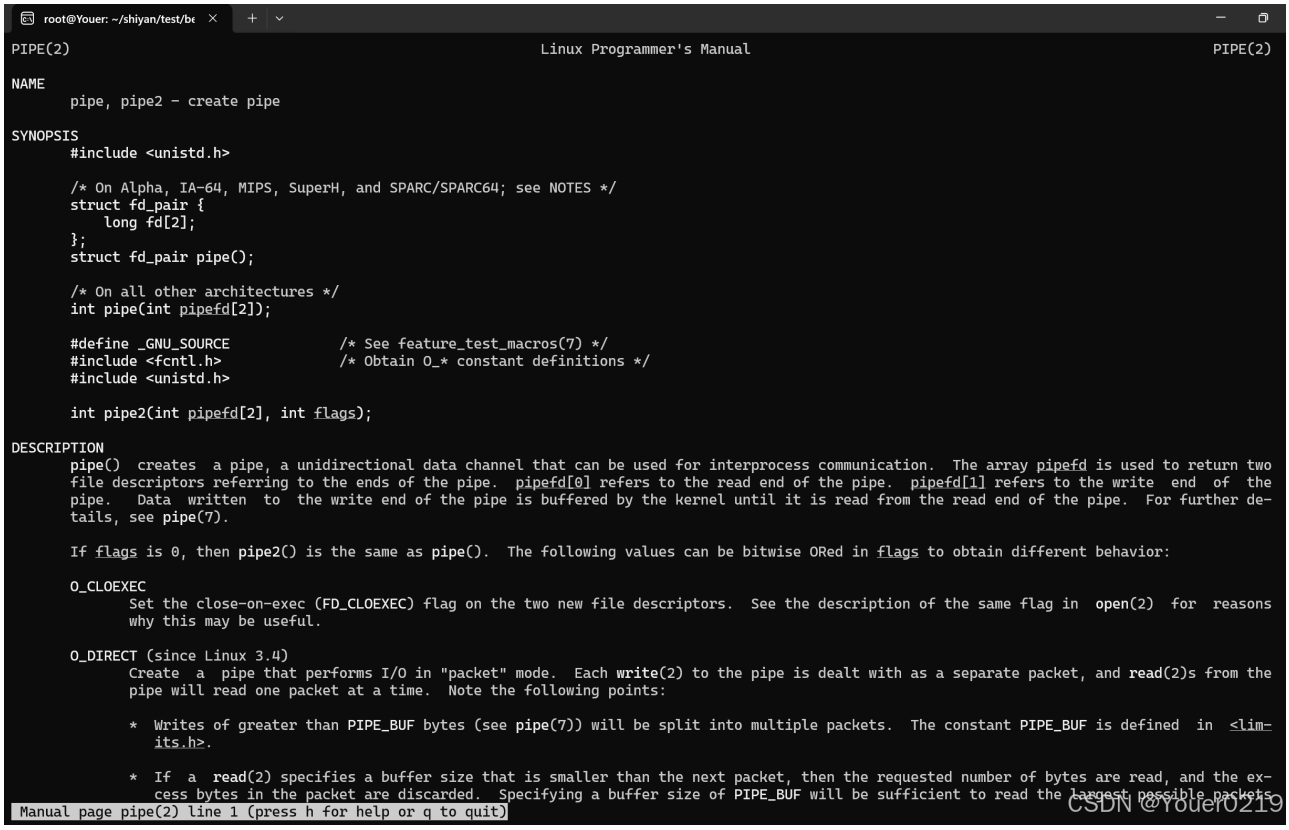
- 组合命令：通过简单的符号 | 将多个命令串联起来，构成更复杂、功能更强的任务，而无需编写脚本或程序。
- 提高效率：可以在一行命令中完成多步骤的数据处理任务，减少中间文件的创建和管理，提升工作效率。
- 灵活处理：可以根据需要随意组合不同的命令，以适应各种各样的需求和场景。
- 资源节约：由于是直接传递数据流，避免了不必要的磁盘读写操作，节省了系统资源。

使用Linux系统调用编写实现管道（pipe）功能时，需要什么系统调用？提交man -k 相关截图。

- 需要的系统调用：
  - pipe()：创建一个管道。这个系统调用会创建一对文件描述符（fd[0] 和 fd[1]），其中 fd[0] 是读端，fd[1] 是写端。
  - 一般配合以下系统调用使用：
    - fork()：创建子进程。管道通常用于父子进程之间的通信，因此需要创建至少一个子进程来形成通信的两端。
    - dup2()：用于复制文件描述符，并可以用来重定向标准输入、输出或错误到管道的某一端。
    - exec*()：执行新程序。子进程中通常会使用 exec 系列函数之一来替换当前进程映像为新的程序，从而执行不同的命令。
    - close()：关闭不再需要的文件描述符。确保关闭所有不使用的文件描述符以避免资源泄漏。
- man -k pipe截图

```
root@Youer:~/shiyan/test/bestidiocs2024/ch05# man -k pipe
devlink-dpipe (8)    - devlink dataplane pipeline visualization
fifo (7)             - first-in first-out special file, named pipe
funzip (1)           - filter for extracting from a ZIP archive in a pipe
gst-launch-1.0 (1)   - build and run a GStreamer pipeline
lesspipe (1)         - "input preprocessor" for less.
mdig (1)             - DNS pipelined lookup utility
mkfifo (1)           - make FIFOs (named pipes)
mkfifo (3)           - make a FIFO special file (a named pipe)
mkfifoat (3)         - make a FIFO special file (a named pipe)
pclose (3)           - pipe stream to or from a process
pipe (2)             - create pipe
pipe (7)             - overview of pipes and FIFOs
pipe2 (2)            - create pipe
pipewire (1)         - The PipeWire media server
pipewire.conf (5)    - The PipeWire server configuration file
popen (3)            - pipe stream to or from a process
pw-cat (1)           - Play and record media with PipeWire
pw-cli (1)           - The PipeWire Command Line Interface
pw-dot (1)           - The PipeWire dot graph dump
pw-metadata (1)      - The PipeWire metadata
pw-mididump (1)      - The PipeWire MIDI dump
pw-midiplay (1)      - Play and record media with PipeWire
pw-midirecord (1)    - Play and record media with PipeWire
pw-mon (1)           - The PipeWire monitor
pw-play (1)          - Play and record media with PipeWire
pw-profiler (1)      - The PipeWire profiler
pw-record (1)        - Play and record media with PipeWire
splice (2)           - splice data to/from a pipe
systemd-cat (1)      - Connect a pipeline or program's output with the journal
tee (2)              - duplicating pipe content
vmsplice (2)         - splice user pages to/from a pipe
root@Youer:~/shiyan/test/bestidiocs2024/ch05#
```

- man 2 pipe截图



使用系统调用创建一个管道，父进程向管道写入数据，子进程从管道读取数据。在父进程中使用 write 系统调用写入字符串“你的八位学号+姓名”，并在子进程中使用 read 系统调用读取数据并打印。提交代码，编译运行过程截图（可文本）

- 代码：

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#define MESSAGE "20221414xlm" // 要写入管道的字符串
#define oops(m, x) { perror(m); exit(x); }

int main() {
    int pipefd[2];          // 管道文件描述符：pipefd[0] 读端，pipefd[1] 写端
    pid_t pid;              // 存储 fork 返回的进程 ID
    char buffer[50];        // 存储读取的数据
    ssize_t read_len;       // 读取的字节数

    // 创建管道
    if (pipe(pipefd) == -1)
        oops("pipe failed", 1);

    // 创建子进程
    pid = fork();
    if (pid == -1)
        oops("fork failed", 2);
```

```
    if (pid == 0) { // 子进程
        close(pipefd[1]); // 关闭写端
        read_len = read(pipefd[0], buffer, sizeof(buffer)); // 从管道读取数据
        if (read_len > 0) {
            buffer[read_len] = '\0'; // 确保字符串以 \0 结束
            printf("子进程读取到的数据: %s\n", buffer);
        } else {
            oops("read failed", 3);
        }
        close(pipefd[0]); // 关闭读端
    } else { // 父进程
        close(pipefd[0]); // 关闭读端
        if (write(pipefd[1], MESSAGE, strlen(MESSAGE) + 1) == -1) // 写入字符
串，包括 \0
            oops("write failed", 4);
        close(pipefd[1]); // 关闭写端
    }

    return 0;
}
```

- 运行结果：

```
root@Youer:~/shiyan/test/bestidiocs2024/ch05/process/pipe# nano test_pipe.c
root@Youer:~/shiyan/test/bestidiocs2024/ch05/process/pipe# mv test_pipe.c
pipe_example.c
root@Youer:~/shiyan/test/bestidiocs2024/ch05/process/pipe# gcc -o pipe_example
pipe_example.c
root@Youer:~/shiyan/test/bestidiocs2024/ch05/process/pipe# ./pipe_example
子进程读取到的数据: 20221414xlm
```

提交git log 结果

- git log 结果：

```
root@Youer:~/shiyan/test/bestidiocs2024/ch05/process/pipe# git log
commit 85498ad5f761bb18f65ee936b2cf15863606c9ae (HEAD -> master)
Author: 徐鹿鸣 <xlm20040219@qq.com>
Date:   Tue Dec 17 10:25:11 2024 +0800

    finish pipe code

commit c75de08ce3efb9c48a22c8e882215293fb68727d
Author: 徐鹿鸣 <xlm20040219@qq.com>
Date:   Tue Dec 17 10:20:27 2024 +0800

    first
```