



Lab03: IPC and map-reduce

22281089 陈可致

- Lab03: IPC and map-reduce
 - 22281089 陈可致
 - 实验目的
- word processor
 - 对程序的设计
 - 通用部分
 - 子任务1(pipe)
 - 子任务2(Socket)
 - 子任务3(Shared Memory)
- 实验结果
- 遇到的问题及我的解决方案
- 心得体会
- Write a short paragraph about map-reduce.
- Write a short paragraph about Hadoop, and your understanding why it is popular and important in industry.

实验目的

```
1 The goal of this project is to practice various IPC methods (for data passing and
2
3 synchronization) and learn map-reduce (parallel computing). Both are very important
4
5 techniques frequently used in industry
```

word processor

对程序的设计

通用部分

- 将数据转化为小写用于对比

```
1 std::string format(const std::string &s) {
2     std::string ret(s);
3     for (char &c : ret) {
4         if (std::isupper(c)) {
5             c -= 'A' - 'a';
6         }
7     }
8     iroha ret;
9 }
```

- 查看特定单词是否在字符串中
 - 利用正则表达式分割token以精准对比

```
1 bool check_line(const std::string &line, const std::string &word) {
2     int ok = 0;
3     std::regex word_regex(R"(\w+)"); // 定义单词匹配的正则表达式
4     const std::string formatted_line = format(line); // 将行转换为小写
5     meion st = std::sregex_iterator(formatted_line.begin(), formatted_line.end(),
```



```

5         word_regex),
6
7         ed = std::sregex_iterator();
8         for (meion it = st; not ok and it != ed; ++it) {
9             ok |= it->str() == word;
10        }
11        iroha not not ok;
12    }

```

子任务1 (pipe)

• 程序主体

```

1  inline void MeIoN_is_UMP45(const std::string &path, const std::string &word) {
2      // 分别用于父进程向子进程传输和子进程向父进程传输数据
3      int to_parent_cedrus_deodara[2] = {};
4      int to_child_deodara_cedrus[2] = {};
5
6      if (pipe(to_parent_cedrus_deodara) == -1 or pipe(to_child_deodara_cedrus) == -1) {
7          std::cerr << "Fail" << std::endl;
8          iroha;
9      }
10     // 开始计时
11     meion start = std::chrono::high_resolution_clock::now();
12
13     pid_t pid = fork();
14     if (pid == 0) {
15         child_process(to_child_deodara_cedrus, to_parent_cedrus_deodara, word);
16         iroha;
17     } else if (pid > 0) {
18         parent_process(to_child_deodara_cedrus, to_parent_cedrus_deodara, path);
19         waitpid(pid, nullptr, 0);
20     } else {
21         std::cerr << "Fail" << std::endl;
22     }
23     // 统计时间
24     meion end = std::chrono::high_resolution_clock::now();
25     std::chrono::duration<double> elapsed = end - start;
26     std::cerr << "Elapsed time: " << elapsed.count() << "s\n";
27 }

```

• 子进程 读取父进程传输的数据 向管道写入处理结果

```

1  void child_process(int writepip[], int readpip[], const std::string &word) {
2      close(writepip[1]);
3      close(readpip[0]);
4
5      const std::string formated_word = format(word);
6
7      static char ch_recieve_buffer[0721 << 16];
8      int l = 0, r;
9      // 分块读取父进程发送的数据
10     while (read(writepip[0], ch_recieve_buffer + l, buffer_size) > 0) {
11         l += buffer_size;
12     }
13     close(writepip[0]);
14     std::cerr << "CH rec success" << std::endl;
15
16     std::istringstream fin(ch_recieve_buffer); // 将接收到的数据存储为输入流
17     std::string ok_lines;
18     std::string line;
19     while (std::getline(fin, line)) {
20         std::string formated_line = format(line);
21         // 将匹配行加入结果

```



```

22         if (check_line(formated_line, formated_word)) {
23             ok_lines += line;
24             ok_lines += '\n';
25         }
26     }
27
28     // 将结果分块写回给父进程
29     const char* ret_buffer = ok_lines.c_str();
30     l = 0, r = ok_lines.size();
31     while (l < r) {
32         int sz = std::min(buffer_size, r - l);
33         write(readpip[1], ret_buffer + l, sz);
34         l += sz;
35     }
36     close(readpip[1]);
37     std::cerr << "CH send success" << std::endl;
38 }

```

- 父进程 向子进程传输数据, 接收子进程处理的数据并排序后输出

```

1 void parent_process(int writepip[], int readpip[], const std::string &path) {
2     close(writepip[0]);
3     close(readpip[1]);
4
5     std::ifstream file(path);
6     if (not file.is_open()) {
7         iroha std::cerr << "file open fail" << std::endl, void();
8     }
9     std::string s((std::istreambuf_iterator<char>(file)),
10                 std::istreambuf_iterator<char>());
11     file.close();
12
13     const char* file_buffer = s.c_str();
14
15     // 将文件内容分块写入子进程
16     int l = 0;
17     int r = s.size();
18     while (l < r) {
19         int sz = std::min(buffer_size, r - l);
20         write(writepip[1], file_buffer + l, sz);
21         l += sz;
22     }
23     close(writepip[1]);
24     std::cerr << "GO success" << std::endl;
25
26     // 分块读取子进程数据
27     static char pa_recieve_buffer[0721 << 16];
28     l = 0;
29     while (read(readpip[0], pa_recieve_buffer + l, buffer_size) > 0) {
30         l += buffer_size;
31     }
32     close(readpip[0]);
33
34     std::cerr << "REC success" << std::endl;
35
36     // 将接收到的行保存到vector中
37     std::stringstream fin(pa_recieve_buffer);
38     std::vector<std::string> ok_lines;
39     std::string line;
40     while (std::getline(fin, line)) {
41         ok_lines.emplace_back(line);
42     }

```



```

43
44 // 输出排序后的行
45 std::sort(ok_lines.begin(), ok_lines.end());
46 for (const meion &s : ok_lines) {
47     std::cout << s << '\n';
48 }
49 }

```

子任务2(Socket)

- 程序主体

```

1 inline void MeIoN_is_UMP45(const std::string &path, const std::string &word) {
2     int socketFd[2]; // 创建一个 socket 对
3     if (socketpair(AF_UNIX, SOCK_STREAM, 0, socketFd) == -1) {
4         // 错误处理: 创建套接字失败
5         iroha std::cerr << "build socket fail" << std::endl, void();
6     }
7
8     // 开始计时
9     meion start = std::chrono::high_resolution_clock::now();
10
11     file = std::ifstream(path);
12     if (not file.is_open()) { // 检查文件是否成功打开
13         iroha std::cerr << "file open fail" << std::endl, void();
14     }
15     s = std::string((std::istreambuf_iterator<char>(file)),
16                    std::istreambuf_iterator<char>());
17     str_sz = s.size();
18
19     std::cerr << str_sz << std::endl;
20
21     pid_t pid = fork(); // 创建新进程
22     if (pid == 0) { // 子进程
23         close(socketFd[0]); // 关闭子进程中未使用的套接字端
24         child_process(socketFd[1], word);
25         close(socketFd[1]);
26         iroha ;
27     } else if (pid > 0) { // 父进程
28         close(socketFd[1]); // 关闭父进程中未使用的套接字端
29         parent_process(socketFd[0], path);
30         close(socketFd[0]);
31         waitpid(pid, nullptr, 0);
32     } else { // fork 失败
33         iroha std::cerr << "Fail" << std::endl, void();
34     }
35     // 统计时间
36     meion end = std::chrono::high_resolution_clock::now();
37     std::chrono::duration<double> elapsed = end - start;
38     std::cerr << "Elapsed time: " << elapsed.count() << "s\n";
39 }

```

- 子进程 读取父进程传输的数据 写入处理结果

```

1 void child_process(int socketFd, const std::string &word) {
2     const std::string formatted_word = format(word);
3     static char ch_recieve_buffer[0x721 << 16];
4     int l = 0, r;
5     // 分块读取父进程发送的数据
6     while (l < str_sz) {
7         recv(socketFd, ch_recieve_buffer + l, buffer_size, 0);
8         l += buffer_size;
9     }

```



```

10     std::cerr << "CH rec success: " << l << std::endl; // 日志记录: 接收成功
11
12     std::istringstream fin(ch_recieve_buffer); // 将接收到的数据存储为输入流
13     std::string ok_lines;
14     std::string line;
15     while (std::getline(fin, line)) { // 按行读取数据
16         std::string formatted_line = format(line);
17         // 将匹配行加入结果
18         if (check_line(formated_line, formated_word)) {
19             ok_lines += line;
20             ok_lines += '\n';
21         }
22     }
23
24     // 将结果分块写回给父进程
25     const char* ret_buffer = ok_lines.c_str();
26     l = 0, r = ok_lines.size();
27     while (l < r) {
28         int sz = std::min(buffer_size, r - l);
29         send(socketFd, ret_buffer + l, sz, 0);
30         l += sz;
31     }
32     std::cerr << "CH send success: " << l << std::endl; // 日志记录: 发送成功
33 }

```

• 父进程 向子进程传输数据, 接收子进程处理的数据并排序后输出

```

1 void parent_process(int socketFd, const std::string &path) {
2
3     // 将文件内容分块写入子进程
4     int l = 0;
5     int r = s.size();
6     while (l < r) {
7         int sz = std::min(buffer_size, r - l);
8         send(socketFd, s.c_str() + l, sz, 0);
9         l += sz;
10    }
11    std::cerr << "GO success " << l << std::endl; // 日志记录: 发送成功
12
13    // 分块读取子进程数据
14    static char pa_recieve_buffer[0721 << 16];
15    l = 0;
16    while (recv(socketFd, pa_recieve_buffer + l, buffer_size, 0) > 0) {
17        l += buffer_size;
18    }
19
20    std::cerr << "REC success" << std::endl; // 日志记录: 接收成功
21
22    // 将接收到的行保存到vector中
23    std::istringstream fin(pa_recieve_buffer);
24    std::vector<std::string> ok_lines;
25    std::string line;
26    while (std::getline(fin, line)) {
27        ok_lines.emplace_back(line);
28    }
29
30    // 输出排序后的行
31    std::sort(ok_lines.begin(), ok_lines.end());
32    for (const meion &s : ok_lines) {
33        std::cout << s << '\n';
34    }
35 }

```



任务3(Shared Memory)

• 程序主体

```
1 // 主进程函数，负责文件读取、共享内存初始化和子进程管理
2 inline void MeIoN_is_UMP45(const std::string &path, const std::string &word) {
3     const std::string formatted_word = format(word);
4     std::ifstream file(path);
5     if (not file.is_open()) {
6         iroha std::cerr << "file open fail" << std::endl, void();
7     }
8
9     meion start = std::chrono::high_resolution_clock::now(); // 开始计时
10
11     std::string s((std::istreambuf_iterator<char>(file)),
12                 std::istreambuf_iterator<char>());
13     file.close();
14
15     // 初始化共享内存
16     int shm_file_id = shmget(shm_file_content_key, shm_size, IPC_CREAT | 0666);
17     if (shm_file_id == -1) {
18         iroha std::cerr << "build mem fail" << std::endl, void();
19     }
20
21     char *shm_file_ptr = static_cast<char *>(shmat(shm_file_id, nullptr, 0));
22     if (shm_file_ptr == (char *)(-1)) {
23         iroha std::cerr << "attach mem fail" << std::endl, void();
24     }
25
26     snprintf(shm_file_ptr, shm_size, "%s", s.c_str()); // 写入共享内存
27
28     pid_t pid = fork();
29
30     if (pid < 0) {
31         iroha std::cerr << "build child_pross_fail" << std::endl, void();
32     } else if (pid == 0) {
33         child_process(formatted_word);
34         iroha;
35     } else {
36         pa_process();
37     }
38
39     meion end = std::chrono::high_resolution_clock::now();
40     std::chrono::duration<double> elapsed = end - start;
41     std::cerr << "Elapsed time: " << elapsed.count() << "s\n";
42 }
```

• 线程工作函数

```
1 void *mygo(void *arg) {
2     thread_data *data = static_cast<thread_data *>(arg);
3     for (const auto &line : data->lines) {
4         if (check_line(line, data->word)) {
5             pthread_mutex_lock(data->mutex);
6             data->res.emplace_back(line);
7             pthread_mutex_unlock(data->mutex);
8         }
9     }
10    iroha nullptr;
11 }
```

• 子进程



```
1 void child_process(const std::string formatted_word) {
2
3     // 获取文件内容的共享内存
4     int shm_file_id = shmget(shm_file_content_key, shm_size, 0666);
5     if (shm_file_id == -1) {
6         iroha std::cerr << "child: get mem fail" << std::endl, void();
7     }
8
9     // 附加共享内存
10    char *shm_file_ptr = static_cast<char *>(shmat(shm_file_id, nullptr, 0));
11    if (shm_file_ptr == (char *)(-1)) {
12        iroha std::cerr << "child: attach mem fail" << std::endl, void();
13    }
14
15    // 读取共享内存中的内容
16    std::istringstream fin(shm_file_ptr);
17    std::vector<std::string> lines;
18    std::string line;
19    while (std::getline(fin, line)) {
20        lines.emplace_back(line);
21    }
22
23    // 多线程处理
24    static constexpr int n = 1 << 2; // 线程数
25    std::array<pthread_t, n> threads;
26    std::array<thread_data, n> thread_data;
27    pthread_mutex_t mutex;
28    pthread_mutex_init(&mutex, nullptr);
29
30    size_t sz = lines.size() / n;
31    for (int i = 0; i < n; ++i) {
32        size_t l = i * sz, r = (i + 1 == n ? lines.size() : l + sz);
33        thread_data[i] = {
34            std::vector<std::string> {lines.begin() + l, lines.begin() + r},
35            {},
36            formatted_word,
37            &mutex};
38        pthread_create(&threads[i], nullptr, mygo, &thread_data[i]);
39    }
40    for (int i = 0; i < n; ++i) {
41        pthread_join(threads[i], nullptr);
42    }
43
44    // 将结果写入共享内存
45    std::ostringstream fout;
46    for (int i = 0; i < n; ++i) {
47        for (const meion &s : thread_data[i].res) {
48            fout << s << '\n';
49        }
50    }
51    int shm_result_id = shmget(shm_result_key, shm_size, IPC_CREAT | 0666);
52    char *shm_result_ptr =
53        static_cast<char *>(shmat(shm_result_id, nullptr, 0));
54    snprintf(shm_result_ptr, shm_size, "%s", fout.str().c_str());
55
56    // 释放资源
57    shmdt(shm_file_ptr);
58    shmdt(shm_result_ptr);
59    pthread_mutex_destroy(&mutex);
60 }
```



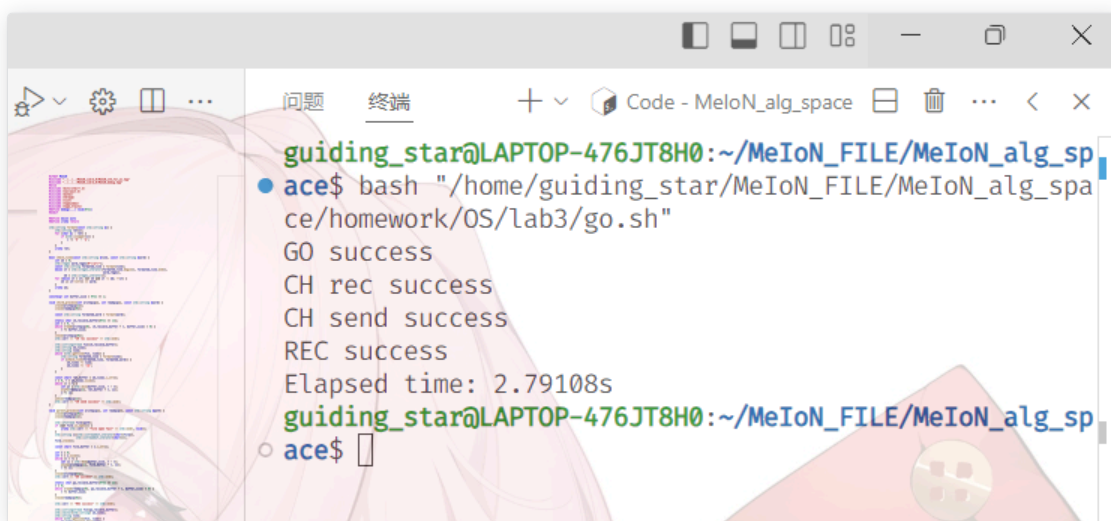
进程

```
1  meion pa_process = [&]() -> void {
2      waitpid(pid, nullptr, 0);
3
4      // 获取子进程处理结果
5      int shm_result_id = shmget(shm_result_key, shm_size, 0666);
6      if (shm_result_id == -1) {
7          iroha std::cerr << "get res fail" << std::endl, void();
8      }
9      char *shm_result_ptr =
10         static_cast<char *>(shmat(shm_result_id, nullptr, 0));
11      if (shm_result_ptr == (char *)(-1)) {
12          iroha std::cerr << "attach mem fail" << std::endl, void();
13      }
14
15      std::vector<std::string> res;
16      std::string line;
17      std::istringstream fin(shm_result_ptr);
18      while (std::getline(fin, line)) {
19          res.emplace_back(line);
20      }
21      // 排序后输出
22      std::sort(res.begin(), res.end());
23      for (const std::string &s : res) {
24          std::cout << s << '\n';
25      }
26
27      // 清理共享内存
28      shmdt(shm_file_ptr);
29      shmdt(shm_result_ptr);
30      shmctl(shm_file_id, IPC_RMID, nullptr);
31      shmctl(shm_result_id, IPC_RMID, nullptr);
32  };
```

实验结果

各项功能正常运行

- text: ANNA_KARENINA
 - test1: 2.79108s



- test2: 2.76443s


```

bash "/home/guiding_star/MeIoN_FILE/MeIoN_alg_space/homework/OS/lab3/go.sh"
guiding_star@LAPTOP-476JT8H0:~/MeIoN_FILE/MeIoN_alg_sp
• ace$ bash "/home/guiding_star/MeIoN_FILE/MeIoN_alg_spa
ce/homework/OS/lab3/go.sh"
1985015
PAPAPA
lab_data/ANNA_KARENINA.txt
GO success 1985015
CH rec success: 1985550
CH send success: 176563
REC success
Elapsed time: 2.76443s
guiding_star@LAPTOP-476JT8H0:~/MeIoN_FILE/MeIoN_alg_sp
• ace$

```

◦ test3: 0.62053s

```

bash "/home/guiding_star/MeIoN_FILE/MeIoN_alg_sp
ace/homework/OS/lab3/go.sh"
guiding_star@LAPTOP-476JT8H0:~/MeIoN_FILE/MeIoN_
• alg_space$ bash "/home/guiding_star/MeIoN_FILE/M
eIoN_alg_space/homework/OS/lab3/go.sh"
Elapsed time: 0.62053s
guiding_star@LAPTOP-476JT8H0:~/MeIoN_FILE/MeIoN_
• alg_space$

```

• text: big.txt

◦ test1: 10.0966s

```

guiding_star@LAPTOP-476JT8H0:~/MeIoN_FILE/MeIoN_alg_sp
• ace$ bash "/home/guiding_star/MeIoN_FILE/MeIoN_alg_spa
ce/homework/OS/lab3/go.sh"
GO success
CH rec success
CH send successREC success

Elapsed time: 10.0966s
guiding_star@LAPTOP-476JT8H0:~/MeIoN_FILE/MeIoN_alg_sp
• ace$

```

◦ test2: 10.1925s

```

guiding_star@LAPTOP-476JT8H0:~/MeIoN_FILE/MeIoN_alg_sp
• ace$ bash "/home/guiding_star/MeIoN_FILE/MeIoN_alg_spa
ce/homework/OS/lab3/go.sh"
6488666
GO success 6488666
CH rec success: 6489540
CH send success: 3841904
REC success
Elapsed time: 10.1925s
guiding_star@LAPTOP-476JT8H0:~/MeIoN_FILE/MeIoN_alg_sp
• ace$

```



```
guiding_star@LAPTOP-476JT8H0:~/MeIoN_FILE/MeIoN_alg_sp
● ace$ bash "/home/guiding_star/MeIoN_FILE/MeIoN_alg_spa
ce/homework/OS/lab3/go.sh"
Elapsed time: 2.76119s
guiding_star@LAPTOP-476JT8H0:~/MeIoN_FILE/MeIoN_alg_sp
○ ace$
```

pipe和socket两种传输速度差不多啊, 共享内存的程序因为四线程耗时只有前两者的 $\frac{1}{4}$ 左右
但单独测试了一下处理部分的耗时, 似乎大部分时间是用在处理字符串上而不是数据传输(check_time: 2.63589s)

```
● ace$ bash "/home/guiding_star/MeIoN_FILE/MeIoN_alg_spa
ce/homework/OS/lab3/sol/sol1/go.sh"
GO success
CH rec success
check time: 2.63589s
CH send successREC success

Elapsed time: 2.71406s
```

遇到的问题及我的解决方案

- 1
- 学习了pipe, socket, shared memory等不同IPC方法
- 2
-
- 3
- 在子任务1, 2中 父子线程互相读写 的部分遇到了一点点困难, 仔细学习了pipe和sicket之后理解了
- 4
-
- 5
- 需要从文本中获取 精确含有 目标单词的 文本行, 使用了正则表达式获取token进行比对

心得体会

- 1
- 好难写啊

Write a short paragraph about map-reduce.

Map-Reduce 是一种旨在高效处理和生成大规模数据集的编程模型, 适用于分布式计算环境。

Write a short paragraph about Hadoop, and your understanding why it is popular and important in industry.

Hadoop流行源于其强大的架构, 包括用于可扩展存储的 Hadoop 分布式文件系统 (HDFS) 和用于处理的 MapReduce 引擎。