操作系统实验报告

姓名：陶略

学号：031510319

班号：1615102

2018

目录

[文件读写编程题目 2](#_Toc517360572)

[题一：myecho.c 2](#_Toc517360573)

[题二：mycat.c 2](#_Toc517360574)

[题三：mycp.c 4](#_Toc517360575)

[多进程题目 5](#_Toc517360576)

[题一：mysys.c 5](#_Toc517360577)

[题二：sh1.c 7](#_Toc517360578)

[题三：sh2.c 9](#_Toc517360579)

[题四：sh3.c 11](#_Toc517360580)

[多线程题目 15](#_Toc517360581)

[题一：pi1.c 15](#_Toc517360582)

[题二：pi2.c 16](#_Toc517360583)

[题三：sort.c 17](#_Toc517360584)

[题四：pc1.c 19](#_Toc517360585)

[题五：pc2.c 22](#_Toc517360586)

[题六：ring.c 24](#_Toc517360587)

## 文件读写编程题目

### 题一：myecho.c

题目要求：

1. myecho.c的功能与系统echo程序相同
2. 接受命令行参数，并将参数打印出来，例子如下：

|  |
| --- |
| $ ./myecho x  x  $ ./myecho a b c  a b c |

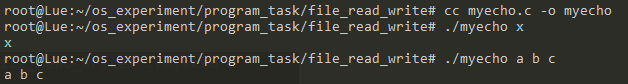
实现思路：

①使用main函数里的argc和argv传参数，命令行输入的字符串会自动被分割成字符串数组，argc记录字符串个数，argv指向字符串数组。只需用一个循环输出以空格为间隔输出所有字符串即可，最后输出一个回车。

代码：

1. **int** main(**int** argc, **char**\*\* argv)
2. {
3. **for**(**int** i = 1; i < argc; i++)
4. printf("%s ", argv[i]);
5. printf("\n");
6. **return** 0;
7. }

实验截图：



### 题二：mycat.c

题目要求：

1. mycat.c的功能与系统cat程序相同
2. mycat将指定的文件内容输出到屏幕，例子如下：
3. 要求使用系统调用open/read/write/close实现

|  |
| --- |
| $ cat /etc/passwd  root:x:0:0:root:/root:/bin/bash  daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin  bin:x:2:2:bin:/bin:/usr/sbin/nologin  ...  $ ./mycat /etc/passwd  root:x:0:0:root:/root:/bin/bash  daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin  bin:x:2:2:bin:/bin:/usr/sbin/nologin  ... |

实现思路：

①首先实现了一个函数如下。该函数接受任意两个文件描述符，将infile文件的内容全部送入outfile文件。文件描述符infile可以是用open函数打开的文件，也可以时标准输入；文件描述符outfile可以是用open函数打开的文件，也可以是标准输出。

1. **int** copy\_file(**int** infile, **int** outfile)
2. {
3. **int** num;
4. **char** buf[1];
5. **do**{
6. num = read(infile, buf, 1);
7. write(outfile, buf, num);
8. }**while**(num == 1);
9. **return** num;
10. }

② cat接受的参数个数可以是任意自然数，即包括0。

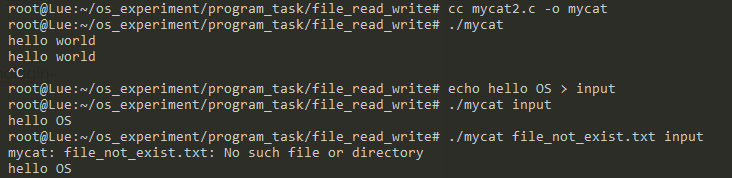
当参数个数为0时，只需把标准输入的字符串送入标准输出即可。

1. **if**(argc == 1)
2. {
3. copy\_file(0, 1);
4. }

当参数个数大于0时，对参数个数做一个循环，并判断文件打开是否成功，成功则调用copy\_file(infile, 1)将文件内容送入标准输出，失败则提示失败并跳过该文件。

1. **char** \*filename;
2. **int** infile;
3. **for**(**int** i = 1; i < argc; i++)
4. {
5. filename = argv[i];
6. **if**((infile = open(argv[1], O\_RDONLY)) == -1)
7. {
8. printf("mycat: %s: No such file or directory\n", filename);
9. **continue**;
10. }
11. copy\_file(infile, 1);
12. close(infile);
13. }

实验截图：



### 题三：mycp.c

题目要求：

1. mycp.c的功能与系统cp程序相同
2. 将源文件复制到目标文件，例子如下：
3. 要求使用系统调用open/read/write/close实现

|  |
| --- |
| cat /etc/passwd  root:x:0:0:root:/root:/bin/bash  daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin  bin:x:2:2:bin:/bin:/usr/sbin/nologin  ...  $ ./mycp /etc/passwd passwd.bak  $ cat passwd.bak  root:x:0:0:root:/root:/bin/bash  daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin  bin:x:2:2:bin:/bin:/usr/sbin/nologin  ... |

实现思路：

①如题二，实现了同一个函数copy\_file(int, int)。该函数接受任意两个文件描述符，将infile文件的内容全部送入outfile文件。文件描述符infile可以是用open函数打开的文件，也可以时标准输入；文件描述符outfile可以是用open函数打开的文件，也可以是标准输出。

1. **int** copy\_file(**int** infile, **int** outfile)

②cp接受的参数个数为2个。

①当参数个数为0时，报错。

1. **if**(argc != 3)
2. {
3. printf("The format must be:cp file\_src file\_des");
4. exit(0);
5. }

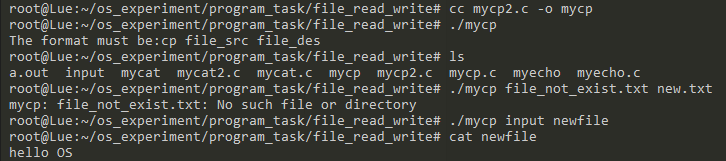
②分别打开argv[1]和argv[2]，打开失败，则报错。

1. **if**((infile = open(argv[1], O\_RDONLY)) == -1)
2. {
3. printf("mycp: %s: No such file or directory\n", argv[1]);
4. exit(0);
5. }
6. **if**((outfile = open(argv[2], O\_CREAT | O\_EXCL | O\_WRONLY, 0644)) == -1)
7. {
8. printf("mycp: %s: Can't create such file\n", argv[2]);
9. exit(0);
10. }

③将打开成功的两个文件描述符传入copy\_file函数，而后关闭文件。

1. copy\_file(infile, outfile);
3. close(infile);
4. close(outfile);

实验截图：



## 多进程题目

### 题一：mysys.c

题目要求：

1. 实现函数mysys，用于执行一个系统命令，要求如下
2. mysys的功能与系统函数system相同，要求用进程管理相关系统调用自己实现
3. 使用fork/exec/wait系统调用实现mysys
4. 不能通过调用系统函数system实现mysys

实现思路：本题有两种实现方式

①直接解析出命令和参数，在子进程中使用execvp装入并执行。

解析命令字符串：以空格为分隔符，把空格字符替换成’\0’字符，将字符串code解析成字符串数组argv.

1. **char** code[MAX\_BUFFLEN];
2. **char** \*argv[MAX\_NUM];    // no more than 100 arguments
3. **int** count = 0;      // N.O. of arguments
4. **char** \*next = NULL;
5. **char** \*rest = code;
6. argv[count++] = code;
7. **while**(next = strchr(rest, ' '))
8. {
9. next[0] = '\0';
10. rest = next + 1;
11. **if**(rest[0] != '\0' && rest[0] != ' ')
12. argv[count++] = rest;
13. **if**(count + 2 > MAX\_NUM)
14. **return** 127;
15. }
16. argv[count++] = NULL;

在子进程中用execvp装入命令和argv。

1. **int** pid;
2. pid = fork();
3. **if**(pid == 0)
4. {
5. **int** error = execvp(code, argv);
6. **if**(error < 0)
7. {
8. perror("execvp");
9. **return** 127;
10. }
11. **else**
12. **return** 0;
13. }

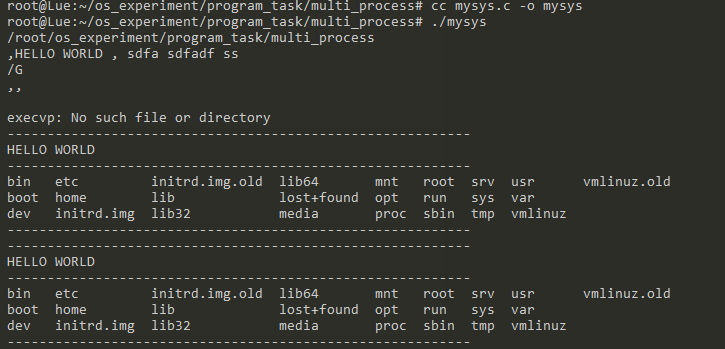
②无需解析出命令和参数，直接用execl调用/bin/sh执行命令。

1. **if** (pid == 0)
2. {
3. execl("/bin/sh", "sh", "-c", cmdstring, (**char** \*)0);
4. exit(127);
5. }

测试程序：

1. **int** main()
2. {
3. mysys("pwd");
4. mysys("echo ,HELLO  WORLD ,  sdfa sdfadf        ss   ");
5. mysys("echo /G");
6. mysys("echo ,,");
7. mysys("echo");
8. mysys("asdfasdf");
9. printf("-----------------------------------------------------\n");
10. mysys("echo HELLO WORLD");
11. printf("-----------------------------------------------------\n");
12. mysys("ls /");
13. printf("-----------------------------------------------------\n");
15. **return** 0;
16. }

实验结果：



### 题二：sh1.c

题目要求：

1. 实现shell程序，要求具备如下功能
2. 支持命令参数

|  |
| --- |
| $ echo arg1 arg2 arg3  $ ls /bin /usr/bin /home |

1. 实现内置命令cd、pwd、exit

|  |
| --- |
| $ cd /bin  $ pwd  /bin |

实现思路：

①使用一个循环来产生一个可交互的终端，对输入的命令解释执行并输出相应结果。

1. **char** buff[MAX\_BUFFLEN];
2. printf("[%s]$ ", dir);
3. **while**(gets(buff))
4. {
5. **int** res = judge\_buff(buff);
6. **if**(res == 0)
7. mysys(buff);
8. **else** **if**(res == 1)
9. cd(buff);
10. **else** **if**(res == 2)
11. **return** 0;
12. printf("[%s]$ ", dir);
13. }

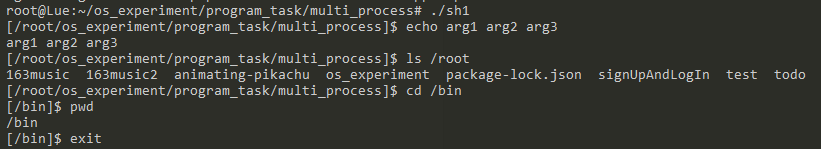
②使用judeg\_buff(char \*)函数来判断输入的命令是否是内置命令。若是cd则返回1，若是exit则返回2，否则返回0。

1. **int** judge\_buff(**char** \*buff)
2. {
3. **char** code[MAX\_BUFFLEN];
4. strcpy(code, buff);
5. **char** \*next = strchr(code, ' ');
6. **if**(next != NULL)
7. next[0] = '\0';
8. **if**(strcmp(code, "cd") == 0)
9. **return** 1;
10. **else** **if**(strcmp(code, "exit") == 0)
11. **return** 2;
12. **else**
13. **return** 0;
14. }

③若不是内置命令则将命令送入mysys函数执行，若是cd则送入cd函数执行，若是exit则程序退出。函数cd(char \*)判断参数，若存在参数，则调用chdir函数进入路径，若不存在参数（参数为空格），则调用chdir进入home目录。

1. // count==2意味着命令本身+NULL，若有参数存在，则count应大于2
2. **if**(count == 2)
3. {
4. chdir(home);
5. dir = getcwd(NULL, 0);
6. }
7. **else**
8. {
9. **int** res = chdir(argv[count - 2]);
10. dir = getcwd(NULL, 0);
11. **if**(res == -1)
12. {
13. printf("cd: No such path %s\n", argv[count - 2]);
14. **return** -1;
15. }
16. }

实验结果：



### 题三：sh2.c

题目要求：

1. 实现shell程序，要求在第1版的基础上，添加如下功能
2. 实现文件重定向

|  |
| --- |
| $ echo hello >log  $ cat log  hello |

实现思路：

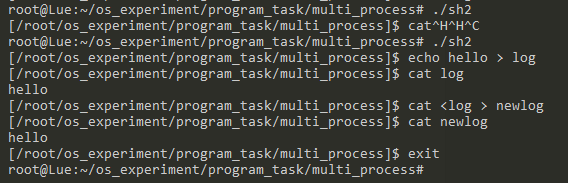
①同样使用while循环来实现可交互的终端，print\_prefix函数用于打印前缀。函数go\_dup用于解释执行命令，支持文件重定向。

1. //in main
2. print\_prefix();
3. **while**(gets(buff))
4. {   go\_dup(buff);
5. print\_prefix();  }
6. **void** print\_prefix()
7. {
8. **if**(strcmp(home, dir) == 0)
9. printf("[~]$ ");
10. **else**
11. printf("[%s]$ ", dir);
12. }

②go\_dup(char \*buff)函数中，检查命令中是否含有字符’<’和‘>’，若二者都有，则将标准输入和标准输出都重定向到输入文件和输出文件；若只有‘<’，则将标准输入重定向到输入文件；若只有‘>’，则将标准输出重定向到输出文件。而后调用mysys函数执行代码。若二者都无，则不进行重定向，并调用go函数执行代码，go函数里进行了sh1.c中内置命令的集成。

1. **if**(a != NULL && b != NULL)
2. {
3. **char** \*in = a + 1 - buff + code;
4. **char** \*out = b + 1 - buff + code;
5. strip\_char(in, ' ');
6. strip\_char(out, ' ');
7. **int** fdin, fdout;
8. fdin = open(in, O\_RDWR, 0666);
9. fdout = open(out, O\_CREAT|O\_RDWR, 0666);
10. **if**(fdin == -1)
11. {
12. printf("File %s open faild\n", in);
13. **return** -1;
14. }
15. **if**(fdout == -1)
16. {
17. printf("File %s open faild\n", out);
18. **return** -1;
19. }
20. dup2(fdin, 0);
21. dup2(fdout, 1);
22. close(fdin);
23. close(fdout);
24. **return** mysys(code);
25. }
26. **else** **if**(a != NULL)
27. {
28. **char** \*in = a + 1 - buff + code;
29. strip\_char(in, ' ');
30. **int** fdin;
31. fdin = open(in, O\_RDWR, 0666);
32. dup2(fdin, 0);
33. close(fdin);
34. **return** mysys(code);
35. }
36. **else** **if**(b != NULL)
37. {
38. **char** \*out = b + 1 - buff + code;
39. strip\_char(out, ' ');
40. **int** fdout;
41. fdout = open(out, O\_CREAT|O\_RDWR, 0666);
42. dup2(fdout, 1);
43. close(fdout);
44. **return** mysys(code);
45. }
46. **else**
47. {
48. **return** go(buff);
49. }

实验结果：



### 题四：sh3.c

题目要求：

1. 实现shell程序，要求在第2版的基础上，添加如下功能
2. 实现管道

|  |
| --- |
| $ echo arg1 arg2 arg3  $ ls /bin /usr/bin /home |

1. 实现管道和文件重定向

|  |
| --- |
| $ cat input.txt  3  2  1  3  2  1  $ cat <input.txt | sort | uniq | cat >output.txt  $ cat output.txt  1  2  3 |

实现思路：

①同样使用while循环来实现可交互的终端，print\_prefix函数用于打印前缀。函数go\_pipe用于解释执行命令，支持管道和文件重定向。由于管道是半双工的，定义了两个管道，用于子进程中新老管道的交替。每执行一行命令更新一次管道。

1. print\_prefix();
2. **while**(fgets(buff, **sizeof**(buff), stdin))
3. {
4. strip(buff);
5. go\_pipe(buff);
6. pipe(fd);
7. pipe(fd\_tmp);
8. print\_prefix();
9. }

②函数go\_pipe中判断字符’|’的个数，若不存在‘|’,则调用sh2.c中的go\_dup函数。

1. **if**(count == 1)  // count == 1表示只存在一个命令，即不存在管道
2. {
3. // recover\_in和recover\_out用于恢复标准输入和标准输出
4. fdin = recover\_in;
5. fdout = recover\_out;
6. **return** go\_dup(buff);
7. }

③若字符’|’的个数不为0，则在子进程之间使用管道。在计算管道个数时，利用loc数组存下每个管道前后命令的位置，以方便为每个命令调用一个子进程。用一个for循环来创建每一个子进程。标志flag初始值为-1，用于告诉当前子进程，前后管道的数量。处于中间的子进程前面有管道，后面也有管道，令flag=2；第一个子进程只有后面有管道，flag=0，最后一个子进程只有前面有管道，falg=1。

在进入第二个子进程之前，关闭管道的入口（写端）。

若前一个子进程用了两个管道，则在进入下一个子进程之前，抛弃老管道fd，将临时管道fd\_tmp设置成当前管道fd，并关闭当前管道的入口（写端）。

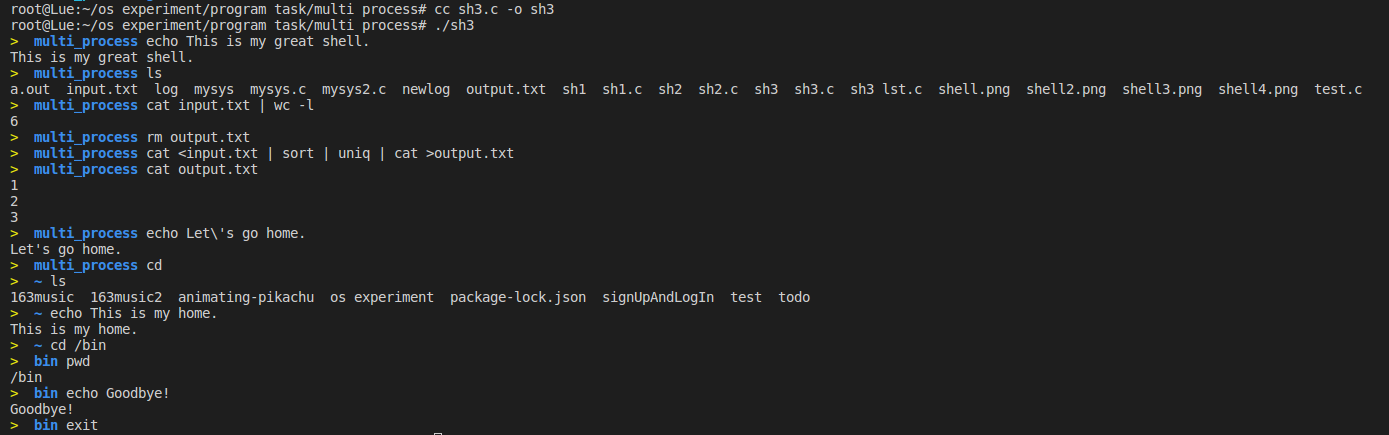
调用pipe\_sys函数执行子进程。

1. **for**(**int** i = 0; i < count; i++)
2. {
3. //printf("[debug] %d pipe: %s\n", i, code+loc[i]);
4. **if**(flag == 2)
5. {
6. dup2(fd\_tmp[0], fd[0]);
7. dup2(fd\_tmp[1], fd[1]);
8. close(fd\_tmp[0]);
9. close(fd\_tmp[1]);
10. pipe(fd\_tmp);
11. close(fd[1]);
12. }
13. **if**(flag == 0)
14. {
15. close(fd[1]);
16. }
17. **if**(i == 0)
18. {
19. flag = 0;
20. }
21. **else** **if**(i == count - 1)
22. {
23. flag = 1;
24. }
25. **else**
26. {
27. flag = 2;
28. }
29. res = pipe\_sys(code + loc[i]);
30. }

④子进程中根据flag判断当前命令前后的管道数量，若前面无管道，则将标准输出重定向到管道入口；若后面无管道，则将标准输入重定向到管道出口；若前后均有管道，则将标准输入重定向到当前管道，将标准输出重定向到临时短道。

1. **int** pipe\_sys(**const** **char** \*cmdstring)
2. {
3. pid\_t pid;
5. pid = fork();
6. **if** (pid == 0)
7. {
8. **if**(flag == 0)
9. {
10. dup2(fd[1], 1);
11. close(fd[0]);
12. close(fd[1]);
13. execl("/bin/sh", "sh", "-c", cmdstring, (**char** \*)0);
14. exit(127);
15. }
16. **else** **if**(flag == 1)
17. {
18. dup2(fd[0], 0);
19. close(fd[0]);
20. close(fd[1]);
21. execl("/bin/sh", "sh", "-c", cmdstring, (**char** \*)0);
22. exit(127);
23. }
24. **else** **if**(flag == 2)
25. {
26. dup2(fd[0], 0);
27. close(fd[0]);
28. close(fd[1]);
29. // 输出进入临时管道
30. dup2(fd\_tmp[1], 1);
31. close(fd\_tmp[0]);
32. close(fd\_tmp[1]);
33. execl("/bin/sh", "sh", "-c", cmdstring, (**char** \*)0);
34. exit(127);
35. }
36. }
37. wait(NULL);
38. **return** 0;
39. }

实验结果：



## 多线程题目

### 题一：pi1.c

题目要求：

1. 使用2个线程根据莱布尼兹级数计算PI
2. 莱布尼兹级数公式: 1 - 1/3 + 1/5 - 1/7 + 1/9 - ... = PI/4
3. 主线程创建1个辅助线程
4. 主线程计算级数的前半部分
5. 辅助线程计算级数的后半部分
6. 主线程等待辅助线程运行結束后,将前半部分和后半部分相加

实现思路：

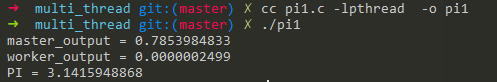
①使用两个线程计算PI，主线程计算前半部分，辅助线程计算后半部分。将最后的计算结果相加后乘以4得到PI的估计值。采用pthread\_create函数创建辅助线程，使用pthread\_join函数等待辅助线程结束。

1. **int** main()
2. {
3. pthread\_t worker\_tid;
4. **float** total;
6. pthread\_create(&worker\_tid, NULL, worker, NULL);
7. master();
8. pthread\_join(worker\_tid, NULL);
9. total = worker\_output + master\_output;
10. printf("PI = %.10f\n", total \* 4);
11. **return** 0;
12. }

②worker函数和master函数分别计算级数的后半段和前半段。

1. **void** \*worker(**void** \*arg)
2. {
3. **int** i;
4. **for**(i = N / 2; i < N; i++)
5. worker\_output += (**float**)sign(i) / (2\*i + 1);
7. printf("worker\_output = %.10f\n", worker\_output);
8. **return** NULL;
9. }
11. **void** master()
12. {
13. **for**(**int** i = 0; i < N / 2; i++)
14. master\_output += (**float**)sign(i) / (2\*i + 1);
16. printf("master\_output = %.10f\n", master\_output);
17. **return**;
18. }

实验结果：



### 题二：pi2.c

题目要求：

1. 使用N个线程根据莱布尼兹级数计算PI
2. 与上一题类似，但本题更加通用化，能适应N个核心，需要使用线程参数来实现
3. 主线程创建N个辅助线程
4. 每个辅助线程计算一部分任务，并将结果返回
5. 主线程等待N个辅助线程运行结束，将所有辅助线程的结果累加

实现思路：

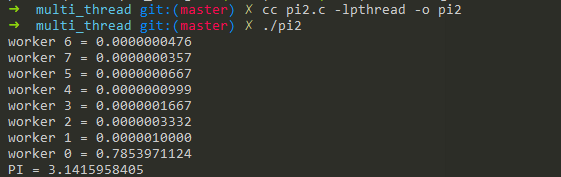
①主线程采用for循环产生N个线程来计算PI。每个线程计算1/N的部分，计算起止点作为参数传入辅助线程的线程入口函数。

1. // 参数结构体的定义
2. **typedef** **struct** param {
3. **int** start;
4. **int** end;
5. }Param;
6. //返回值结构体的定义
7. **typedef** **struct** result {
8. **float** sum;
9. }Result;

②主线程采用for循环，将每个辅助线程的计算结果利用pthread\_join函数接受线程入口函数的返回值。

1. **int** main()
2. {
3. pthread\_t workers[NR\_CPU];
4. Param params[NR\_CPU];
5. **float** total = 0;
7. **for**(**int** i = 0; i < NR\_CPU; i++)
8. {
9. Param \*param;
10. param = ¶ms[i];
11. param->start = i \* NR\_CHILD;
12. param->end = (i + 1) \* NR\_CHILD;
13. pthread\_create(&workers[i], NULL, compute, param);
14. }
16. **for**(**int** i = 0; i < NR\_CPU; i++)
17. {
18. Result \*result;
19. pthread\_join(workers[i], (**void** \*\*)&result);
20. total += result->sum;
21. free(result);
22. }
24. printf("PI = %.10f\n", total \* 4);
25. **return** 0;
26. }

实验结果：



### 题三：sort.c

题目要求：

1. 多线程排序
2. 主线程创建一个辅助线程
3. 主线程使用选择排序算法对数组的前半部分排序
4. 辅助线程使用选择排序算法对数组的后半部分排序
5. 主线程等待辅助线程运行結束后,使用归并排序算法归并数组的前半部分和后半部分

实现思路：

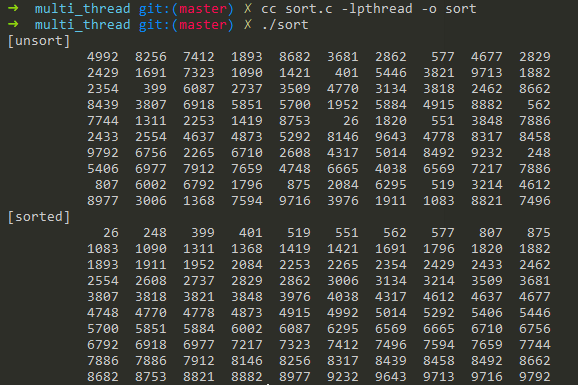
①主线程采用pthread\_create函数创建辅助线程，将辅助线程的排序任务起止点利用线程入口函数的参数传入。线程入口函数采用冒泡排序。

1. **int** main()
2. {
3. generate\_nums();
4. show\_nums(nums, "unsort");
6. pthread\_t worker\_tid;
7. Param params[2];
8. params[0].start = 0;
9. params[0].end = NUMMAX / 2;
10. params[1].start = NUMMAX / 2;
11. params[1].end = NUMMAX;
13. pthread\_create(&worker\_tid, NULL, sort, &params[1]);
14. sort(&params [0]);
16. pthread\_join(worker\_tid, NULL);
17. merge(0, NUMMAX / 2 - 1, NUMMAX - 1);
19. show\_nums(nums, "sorted");
20. **return** 0;
21. }

②使用pthread\_join函数等待辅助线程结束。使用merge函数归并数组的前半部分和后半部分。

1. **void** merge(**const** **int** left, **const** **int** mid, **const** **int** right)
2. {
3. **int** temp[NUMMAX];
4. memcpy(temp, nums, NUMMAX \* **sizeof**(**int**));
5. **int** s1 = left;
6. **int** s2 = mid + 1;
7. **int** t = left;
8. **while**(s1 <= mid && s2 <= right)
9. {
10. **if**(temp[s1] < temp[s2])
11. nums[t++] = temp[s1++];
12. **else**
13. nums[t++] = temp[s2++];
14. }
15. **while**(s1 <= mid)
16. nums[t++] = temp[s1++];
17. **while**(s2 <= right)
18. nums[t++] = temp[s2++];
19. }

实验结果：



### 题四：pc1.c

题目要求：

1. 使用条件变量解决生产者、计算者、消费者问题
2. 系统中有3个线程：生产者、计算者、消费者
3. 系统中有2个容量为4的缓冲区：buffer1、buffer2
4. 生产者生产'a'、'b'、'c'、‘d'、'e'、'f'、'g'、'h'八个字符，放入到buffer1
5. 计算者从buffer1取出字符，将小写字符转换为大写字符，放入到buffer2
6. 消费者从buffer2取出字符，将其打印到屏幕上

实现思路：

①这道题可以被分解为两个生产者-消费者问题，计算者同时承担两个角色。计算者先 作为buffer1的消费者，给buffer1加锁并取数；计算者将小写字母变成大写字母；计 最后再作为buffer2的生产者，给buffer2加锁并存数。

1. **void** \*compute(**void** \*arg)
2. {
3. **char** item;
4. **for**(**int** i = 0; i < ITEM\_COUNT; i++)
5. {
6. pthread\_mutex\_lock(&mutex1);
7. **while**(buffer\_is\_empty(1))
8. pthread\_cond\_wait(&wait\_full\_buffer1, &mutex1);
9. item = get\_item(1);
10. pthread\_cond\_signal(&wait\_empty\_buffer1);
11. pthread\_mutex\_unlock(&mutex1);
13. item += 'A' - 'a';
15. pthread\_mutex\_lock(&mutex2);
16. **while**(buffer\_is\_full(2))
17. pthread\_cond\_wait(&wait\_empty\_buffer2, &mutex2);
18. put\_item(item, 2);
19. printf("\033[33m compute item: %c\n\033[0m", item);     //黄色为计算者
21. pthread\_cond\_signal(&wait\_full\_buffer2);
22. pthread\_mutex\_unlock(&mutex2);
24. }
25. **return** NULL;
26. }

生产者作为buffer1的生产者，给buffer1加锁并存数。

1. **void** \*produce(**void** \*arg)
2. {
3. **char** item;
4. **for**(**int** i = 0; i < ITEM\_COUNT; i++)
5. {
6. pthread\_mutex\_lock(&mutex1);
7. **while**(buffer\_is\_full(1))
8. pthread\_cond\_wait(&wait\_empty\_buffer1, &mutex1);
9. item = 'a' + i;
10. put\_item(item, 1);
11. printf("\033[31m produce item: %c\n\033[0m", item);//红色为生产者
13. pthread\_cond\_signal(&wait\_full\_buffer1);
14. pthread\_mutex\_unlock(&mutex1);
15. }
16. **return** NULL;
17. }

消费者作为buffer2的消费者，给buffer2加锁并取数。

1. **void** \*consume(**void** \*arg)
2. {
3. **int** item;
4. **for**(**int** i = 0; i < ITEM\_COUNT; i++)
5. {
6. pthread\_mutex\_lock(&mutex2);
7. **while**(buffer\_is\_empty(2))
8. pthread\_cond\_wait(&wait\_full\_buffer2, &mutex2);
10. item = get\_item(2);
11. printf("\033[34m consume item: %c\n\033[0m", item);     //蓝色为消费者
13. pthread\_cond\_signal(&wait\_empty\_buffer2);
14. pthread\_mutex\_unlock(&mutex2);
15. }
16. **return** NULL;
17. }

②设置两个互斥量、两个空信号和两个满信号：

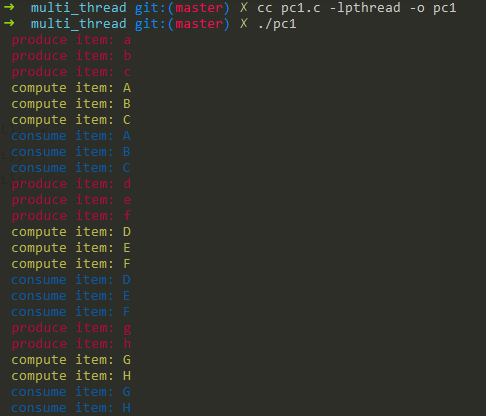
1. pthread\_mutex\_t mutex1, mutex2;
2. pthread\_cond\_t wait\_empty\_buffer1, wait\_empty\_buffer2;
3. pthread\_cond\_t wait\_full\_buffer1, wait\_full\_buffer2;

③创建三个线程分别用于承担生产者、计算者和消费者：

1. **int** main()
2. {
3. pthread\_t producer\_tid, computer\_tid, consumer\_tid;
5. pthread\_mutex\_init(&mutex1, NULL);
6. pthread\_mutex\_init(&mutex2, NULL);
7. pthread\_cond\_init(&wait\_empty\_buffer1, NULL);
8. pthread\_cond\_init(&wait\_empty\_buffer2, NULL);
9. pthread\_cond\_init(&wait\_full\_buffer1, NULL);
10. pthread\_cond\_init(&wait\_full\_buffer2, NULL);
12. pthread\_create(&producer\_tid, NULL, produce, NULL);
13. pthread\_create(&computer\_tid, NULL, compute, NULL);
14. pthread\_create(&consumer\_tid, NULL, consume, NULL);
16. pthread\_join(producer\_tid, NULL);
17. pthread\_join(computer\_tid, NULL);
18. pthread\_join(consumer\_tid, NULL);
20. **return** 0;
21. }

实验结果：

红色为生产者，黄色为计算者，蓝色为消费者。



### 题五：pc2.c

题目要求：

1. 使用信号量解决生产者、计算者、消费者问题
2. 功能和前面的实验相同，使用信号量解决

实现思路：

①这道题与上一题思路相同，实现的时候利用信号量。

信号量的定义、初始化、wait和signal定义如下，初始化时可以送入信号量的初始个数，wait一次减少一次信号量个数，signal一次则增加一次信号量个数。

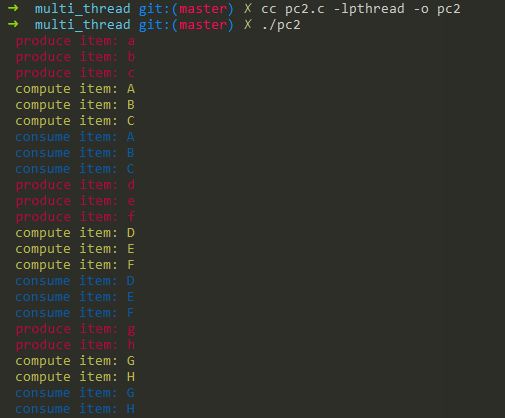
1. **typedef** **struct** {
2. **int** value;
3. pthread\_mutex\_t mutex;
4. pthread\_cond\_t cond;
5. }sema\_t;
7. **void** sema\_init(sema\_t \*sema, **int** value)
8. {
9. sema->value = value;
10. pthread\_mutex\_init(&sema->mutex, NULL);
11. pthread\_cond\_init(&sema->cond, NULL);
12. }
14. **void** sema\_wait(sema\_t \*sema)
15. {
16. pthread\_mutex\_lock(&sema->mutex);
17. **int** i = 1;
18. **while**(sema->value <= 0)
19. {
20. pthread\_cond\_wait(&sema->cond, &sema->mutex);
21. }
22. sema->value--;
23. pthread\_mutex\_unlock(&sema->mutex);
24. }
26. **void** sema\_signal(sema\_t \*sema)
27. {
28. pthread\_mutex\_lock(&sema->mutex);
29. sema->value += 1;
30. pthread\_cond\_signal(&sema->cond);
31. pthread\_mutex\_unlock(&sema->mutex);
32. }

②同样创建三个线程分别用于承担生产者、计算者和消费者，empty\_buffer\_sema1和empty\_buffer\_sema2信号量初始值给为buffer容量-1的大小，表示初始时buffer1和buffer2为空；full\_buffer\_sema1和full\_buffer\_sema2信号量初始值设为0，表示buffer1和buffer2初始时都没有元素供取出。

1. **int** main()
2. {
3. pthread\_t producer\_tid, computer\_tid, consumer\_tid;
5. sema\_init(&mutex\_sema1, 1);
6. sema\_init(&mutex\_sema2, 1);
7. sema\_init(&empty\_buffer\_sema1, CAPACITY - 1);
8. sema\_init(&empty\_buffer\_sema2, CAPACITY - 1);
9. sema\_init(&full\_buffer\_sema1, 0);
10. sema\_init(&full\_buffer\_sema2, 0);
12. pthread\_create(&producer\_tid, NULL, produce, NULL);
13. pthread\_create(&computer\_tid, NULL, compute, NULL);
14. pthread\_create(&consumer\_tid, NULL, consume, NULL);
16. pthread\_join(producer\_tid, NULL);
17. pthread\_join(computer\_tid, NULL);
18. pthread\_join(consumer\_tid, NULL);
20. **return** 0;
21. }

实验结果：

红色为生产者，黄色为计算者，蓝色为消费者。



### 题六：ring.c

题目要求：

1. 创建N个线程，它们构成一个环
2. 创建N个线程：T1、T2、T3、… TN
3. T1向T2发送整数1
4. T2收到后将整数加1
5. T2向T3发送整数2
6. T3收到后将整数加1
7. T3向T4发送整数3
8. …
9. TN收到后将整数加1
10. TN向T1发送整数N

实现思路：

①创建N个缓冲区B，分别对应着N个线程的接受信号口。T1给B2存入整数1，并释放T2已满的信号，T2发现B2满了或者收到B2已满的信号便从T2取数，T3如上。T100取数后，将数加一送入B1，此时等待已久的T1收到T100释放出的B1已满的信号，便从B1中取数。

1. sema\_t mutex\_sema[N];
2. sema\_t full\_buffer\_sema[N];
4. **void** \*add(**void** \*arg)
5. {
6. **int** receive;
7. Param \*param = (Param \*)arg;
8. **int** order = param->order;
9. **if**(order == 0)
10. {
11. sema\_wait(&mutex\_sema[order + 1]);
12. buff[order + 1] = 1;
13. sema\_signal(&mutex\_sema[order + 1]);
14. sema\_signal(&full\_buffer\_sema[order + 1]);
16. sema\_wait(&full\_buffer\_sema[order]);
17. sema\_wait(&mutex\_sema[order]);
18. receive = buff[order];
19. printf("Thread %d received: %d\n", order + 1, receive);
20. sema\_signal(&mutex\_sema[order]);
21. }
22. **else** **if**(order == N - 1)
23. {
24. sema\_wait(&full\_buffer\_sema[order]);
25. sema\_wait(&mutex\_sema[order]);
26. receive = buff[order];
27. printf("Thread %d received: %d\n", order + 1, receive);
28. sema\_signal(&mutex\_sema[order]);
30. sema\_wait(&mutex\_sema[0]);
31. buff[0] = receive + 1;
32. sema\_signal(&mutex\_sema[0]);
33. sema\_signal(&full\_buffer\_sema[0]);
34. }
35. **else**
36. {
37. sema\_wait(&full\_buffer\_sema[order]);
38. sema\_wait(&mutex\_sema[order]);
39. receive = buff[order];
40. printf("Thread %d received: %d\n", order + 1, receive);
41. sema\_signal(&mutex\_sema[order]);
43. sema\_wait(&mutex\_sema[order + 1]);
44. buff[order + 1] = receive + 1;
45. sema\_signal(&mutex\_sema[order + 1]);
46. sema\_signal(&full\_buffer\_sema[order + 1]);
47. }
48. }

②主线程初始化N个互斥量和N个full\_buffer\_sema信号量，创建N个线程，并等待N个线程结束。线程初始化时，传递参数给线程入口函数，告诉新创建的线程它是第几个线程。

1. **int** main()
2. {
3. pthread\_t ring\_tid[N];
4. Param params[N];
5. **for**(**int** i = 0; i < N; i++)
6. {
7. sema\_init(&mutex\_sema[i], 1);
8. sema\_init(&full\_buffer\_sema[i], 0);
9. }
11. **for**(**int** i = 0; i < N; i++)
12. {
13. params[i].order = i;
14. pthread\_create(&ring\_tid[i], NULL, add, &params[i]);
15. }
17. **for**(**int** i = 0; i < N; i++)
18. pthread\_join(ring\_tid[i], NULL);
20. **return** 0;
21. }

实验结果：

此处N设置为100.

