山东大学 计算机科学与技术 学院

操作系统 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：201818130189 | 姓名：王硕 | | 班级：18计科2班 |
| 实验题目：线程和管道通信实验 | | | |
| 实验学时：2 | | 实验日期：2020 10 23 | |
| 实验目的：  通过 Linux 系统中线程和管道通信机制的实验，加深对于线程控制和管道通信概念的理解，观察和体验并发进/线程间的通信和协作的效果 ，练习利用无名管道进行进/线程程通信的编程和调试技术。 | | | |
| 硬件环境：  Ubuntu 20.04 | | | |
| 软件环境：  VS Code、Sublime | | | |
| 实验内容与设计：  设有二元函数f(x,y) = f(x) + f(y) 其中：      请编程建立 3 个并发协作进程或线程，它们分别完成 f(x,y)、f(x)、f(y) | | | |
| 结论分析与体会：   1. 程序运行展示   输入x，y 得到f(x)、f(y)、f(x,y)相应的结果  （1）进程版本：    （2）线程版本     1. 实验分析与总结   Q1. 根据示例实验程序和独立实验程序观察和记录的调试和运行的信息，它们反映出操作系统教材中讲解的进/线程协作和进/线程通信概念的哪些特征和功能？  进程是具有并发性和独立性的。进程是拥有资源的独立单位，而线程是调度和分派的基本单位。  进程协作可以通过共享内存和消息传递的方式通信，此实验通过消息传递的方式进行通信，其特征是通信过程是同步的。进程通信是通过管道进行通信，管道是采用阻塞机制，一方发送未被读取便产生阻塞，一方接受时管道没有信息则产生阻塞，从而达到同步。  Q2. 在真实的操作系统中它是怎样实现和反映出教材中进/线程通信概念的？  可以通过管道通信的方式进行。它是一种信息流缓冲机构， UNIX系统中管道基于文件系统，在内核中通过文件描述符表示，以先进先出(FIFO)方式组织数据传输。调用pipe()函数创建管道int pipe(int fd[2])，fd[0]为管道里的读取端，fd[1]则为管道的写入端。  通过write()函数写入信息，int write (int handle, char \*buf, unsigned len)；进程通过read()函数读取信息，int read (int handle, void \*buf, unsigned len)。  Q3. 对于进/线程协作和进/线程通信的概念和实现有哪些新的理解和认识?  在多道程序设计系统中，同一时刻可能有许多进程，这些进程之间存在两种基本关系：竞争关系和协作关系。进程的互斥、同步、通信都是基于这两种基本关系而存在的，为了解决进程间竞争关系（间接制约关系）而引入进程互斥；为了解决进程间松散的协作关系( 直接制约关系)而引入进程同步；为了解决进程间紧密的协作关系而引入进程通信。竞争关系就是系统中的多个进程之间彼此无关，它们并不知道其他进程的存在，并且也不受其他进程执行的影响，由于这些进程共用了一套计算机系统资源，因而， 必然要出现多个进程竞争资源的问题。当多个进程竞争共享硬设备、存储器、处理器 和文件等资源时，操作系统必须协调好进程对资源的争用；协作关系某些进程为完成同一任务需要分工协作，由于合作的每一个进程都是独立地以不可预知的速度推进，这就需要相互协作的进程在某些协调点上协 调各自的工作。当合作进程中的一个到达协调点后，在尚未得到其伙伴进程发来的消息或信号之前应阻塞自己，直到其他合作进程发来协调信号或消息后方被唤醒并继续执行。这种协作进程之间相互等待对方消息或信号的协调关系称为进程同步。  Q4. 管道机制的机理是什么?  管道实际上是一种固定大小的缓冲区，管道对于管道两端的进程而言，就是一个文件，但它不是普通的文件，它不属于某种文件系统，而是自立门户，单独构成一种文件系统，并且只存在于内存中。它类似于通信中半双工信道的进程通信机制，一个管道可以实现双向 的数据传输，而同一个时刻只能最多有一个方向的传输，不能两个方向同时进行。管道的容 量大小通常为内存上的一页，它的大小并不是受磁盘容量大小的限制。当管道满时，进程在 写管道会被阻塞，而当管道空时，进程读管道会被阻塞,不能同时进行双向传输。匿名管道只能单向；命名管道可以双向；管道是内存中的，可以有多个进程对其进行读操作；也可以有多个进程写，只不过不能同时写。  Q5. 怎样利用管道完成进/线程间的协作和通信?  管道可以实现父子进程之间的通信，并且读取进程会阻塞等到所有的写入进程关闭后，才会收到文件结束符。由此通过一下步骤可以实现父子间同步（1）父进程在创建子进程之前构建管道；(2)每个子进程会继承管道的写入端的文件描述符并在完成动作之后关闭这些描述符；(3)当所有的子进程都关闭了管道的写入端描述符之后，父进程在管道的read（）就会结束并返回文件结束。 | | | |

**附录：**

**pct2.c**

1. #include <stdio.h>
2. #include <unistd.h>
3. #include <stdlib.h>
4. #include <pthread.h>
6. **int** fx(**int** x) {
7. **if** (x == 1) **return** 1;
8. **else** **return** fx(x - 1) \* x;
9. }
11. **int** fy(**int** y) {
12. **if** (y == 1 || y == 2) **return** 1;
13. **else** **return** fy(y - 1) + fy(y - 2);
14. }
16. **void** task1(**int** \*);
17. **void** task2(**int** \*);
18. **void** task3(**int** \*);
20. **int** x, y, z;
21. **int** pipe1[2], pipe2[2], pipe3[2], pipe4[2];
22. pthread\_t thrd1, thrd2, thrd3;
24. **int** main(**int** argc, **char** \*argv[]) {
25. **int** ret;
26. **int** num1, num2, num3;
27. printf("please enter the x, y \n");
28. scanf("%d %d", &x, &y);
30. // pipe()系统调用四个无名管道，建立不成功就退出，执行终止
31. **if** (pipe(pipe1) < 0) {
32. perror("pipe1 not create");
33. exit(EXIT\_FAILURE);
34. }
35. **if** (pipe(pipe2) < 0) {
36. perror("pipe1 not create");
37. exit(EXIT\_FAILURE);
38. }
39. **if** (pipe(pipe3) < 0) {
40. perror("pipe1 not create");
41. exit(EXIT\_FAILURE);
42. }
43. **if** (pipe(pipe4) < 0) {
44. perror("pipe1 not create");
45. exit(EXIT\_FAILURE);
46. }
48. num1 = 1;
49. ret = pthread\_create(&thrd1, NULL, (**void** \*) task1, (**void** \*) &num1);
50. **if** (ret) {
51. perror("phread\_create: task1");
52. exit(EXIT\_FAILURE);
53. }
55. num2 = 2;
56. ret = pthread\_create(&thrd2, NULL, (**void** \*) task2, (**void** \*) &num2);
57. **if** (ret) {
58. perror("phread\_create: task2");
59. exit(EXIT\_FAILURE);
60. }
62. num2 = 3;
63. ret = pthread\_create(&thrd3, NULL, (**void** \*) task3, (**void** \*) &num3);
64. **if** (ret) {
65. perror("phread\_create: task3");
66. exit(EXIT\_FAILURE);
67. }
69. pthread\_join(thrd2, NULL);
70. pthread\_join(thrd3, NULL);
71. pthread\_join(thrd1, NULL);
72. exit(EXIT\_SUCCESS);
73. }
75. //线程1执行函数，它首先向管道1的1端写x，管道3的1端写y，
76. //然后从管道2的0端读f(x)，管道4的1端读f(y)
77. **void** task1(**int** \*num) {
78. write(pipe1[1], &x, **sizeof**(**int**));
79. write(pipe3[1], &y, **sizeof**(**int**));
80. read(pipe2[0], &x, **sizeof**(**int**));
81. read(pipe4[0], &y, **sizeof**(**int**));
83. z = x + y;
84. printf("Received f(x): %d\n", x);
85. printf("Received f(y): %d\n", y);
86. printf("The f(x,y): %d\n", z);
88. close(pipe1[1]);
89. close(pipe3[1]);
90. close(pipe2[0]);
91. close(pipe4[0]);
93. }
95. //线程2执行函数，它首先向管道1的0端读x，然后向管道2的1端写f(x)
96. **void** task2(**int** \*num) {
97. **int** x1;
98. read(pipe1[0], &x1, **sizeof**(**int**));
99. printf("Received x: %d\n", x);
100. x1 = fx(x1);
101. write(pipe2[1], &x1, **sizeof**(**int**));
103. close(pipe1[0]);
104. close(pipe2[1]);
105. }
107. //线程2执行函数，它首先向管道3的0端读y，然后向管道4的1端写f(y)
108. **void** task3(**int** \*num) {
109. **int** y1;
110. read(pipe3[0], &y1, **sizeof**(**int**));
111. printf("Received y: %d\n", y);
112. y1 = fy(y1);
113. write(pipe4[1], &y1, **sizeof**(**int**));
115. close(pipe3[0]);
116. close(pipe4[1]);
117. }