山东大学 软件 学院

**操作系统** 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：202000300125 | 姓名：贾星宇 | | 班级：2020级5班 |
| 实验编号：实验五 | | | |
| 实验题目：进程调度算法实验 | | | |
| 实验学时：4 | | 实验日期：2022年5月1日 | |
| 实验目的：  进一步研究和实践操作系统中关于并发进程同步与互斥操作的一些经典问题 的解法，加深对于非对称性互斥问题有关概念的理解。观察和体验非对称性互 斥问题的并发控制方法。进一步了解 Linux 系统中 IPC 进程同步工具的用法，训 练解决对该类问题的实际编程、调试和分析问题的能力。 | | | |
| 硬件环境：  宿主机：lntel(R) Core(TM) i5-10210U CPU @ 1.60G Hz 2 .1 1 G Hz  虚拟机：ubuntu-20.04.1-desktop-amd64 | | | |
| 软件环境：  宿主机：Windows 10 家庭中文版  虚拟机：Code::Blocks | | | |
| 实验步骤与内容：  1.分析并编写示例实验代码，并运行:  Ipc.h    Ipc.c            Writer.c    Reader.c    Control.c      输入运行命令运行控制程序，随后打开两个读者程序，会开始按时间交替申请读默认内容    随后打开一个写者程序，会发现写者申请后会首先处理写者程序，随后处理读者程序    经验证，发现实验结果和实验大纲结果一致。  2.分析独立实验要求：  理发店问题：假设理发店的理发室中有 3 个理发椅子和 3 个理发师，有一个可容 纳4个顾客坐等理发的沙发。此外还有一间等候室，可容纳13位顾客等候进入理发室。顾客如果发现理发店中顾客已满（超过 20 人），就不进入理发店。 在理发店内，理发师一旦有空就为坐在沙发上等待时间最长的顾客理发，同时 空出的沙发让在等候室中等待时间最长的的顾客就坐。顾客理完发后，可向任何一 位理发师付款。但理发店只有一本现金登记册，在任一时刻只能记录一个顾客的付款。理发师在没有顾客的时候就坐在理发椅子上睡眠。理发师的时间就用在理发、收款、睡眠上。 请利用 linux 系统提供的 IPC 进程通信机制实验并实现理发店问题的一个解法。  编写独立实验代码：  Ipc.c同上  Ipc.h    Baber.c      Customer.c      运行： | | | |
| 结论分析与体会：  【对大纲代码的理解】  1.对写者程序的理解  首先是一系列参数：  设置读过程时间和离开时间、设置读内容的共享内存（起始位置，空间大小，共享权限及根据上述参数申请的指针）、关联请求消息的队列（请求权限、请求位置、申请到的实际的指针）、关联响应消息的队列。  然后进入循环：  首先设置消息类型是写请求，随后将此消息发送出去并进行打印  随后进入等待，等待允许写入消息的msgrcv（）方法执行完成（代表允许写）  *从到消息队列中读出一条新消息的系统调用语法： #include int msgrcv(int msqid, struct msgbuf \*msgp, size\_t msgsz, long msgtype, int msgflg); msqid 由消息队列的标识符 msgp 消息缓冲区指针。消息缓冲区结构为： struct msgbuf { long mtype; /\* 消息类型，必须大于０ \*/ char mtext[1]; /\* 消息数据，长度应于 msgsz 声明的一致\*/ ｝ msgsz 消息数据的长度 msgtype 决定从队列中返回哪条消息： =0 返回消息队列中第一条消息 >0 返回消息队列中等于 mtype 类型的第一条消息。 <=type 绝对值最小值的第一条消息。 msgflg 为０表示阻塞方式，设置 IPC\_NOWAIT 表示非阻塞方式 msgrcv 调用成功返回 0，不成功返回-1。*  允许后便开始for循环，向缓冲区连续写STRSIZ个字符（A开始向后到H）；写完后便开始睡眠指定时间，随后发送完成信号说明已完成。发送完后如果有切换时间则再休眠相应的时间  2.对读者程序的理解  基本思路和上述写者程序相似：  首先设置参数，随后获得自己的id后进入循环：  首先设置请求读的信号，随后将其发送出去并打印到控制台。  然后等待响应，当响应允许读时，打印读到的消息到控制台，随后睡眠指定的时间模拟读的过程；随后发送完成信号后继续睡眠模拟切换的过程，然后进行下一次请求读的循环。  **3.对控制程序的理解：**  首先设置参数：  循环使用的游标i、标志了最大缓冲区容量的count、缓冲区属性msg\_arg、延迟响应的时间、建立的共享内存（内存缓冲区键值、缓冲区大小、建立连接权限、根据上述数值产生的缓冲区的指针、缓冲区内的初始内容（AAAAAAAA））、请求消息队列（权限、键值及形成的队列指针）、响应消息队列（权限、键值及形成的队列指针）  Count初值为100（代表最大读者数为100）  收到写者请求-100，写者完成后将其变为100  收到读者请求-1，读者完成后+1  随后进入循环：  首先根据count的值进行不同的控制：  如果count＞0，说明处在初始状态或者有读者在读，但是没有写者在写，此时不需要延迟处理对写者的响应：  此时利用msgrcv函数从响应消息队列中接收其他**读**进程发送的FINISHED消息，如果接受到一条，则将count++，随后打印XX写者完成的消息；如果没有接收到，则msgrcv函数将**因为IPC\_NOWAIT参数**而跳过此if语句。随后再次进行下述判断。  利用msgrcv函数以**非阻塞等待方式**从响应消息队列进行另外两个消息的接收：  如果接收到READRQUEST消息，说明有读者请求读，则将count--，随后打印XX读者请求读  如果接收到WRITEREQUEST消息，说明有写者请求写，则判断是否允许立即写：如果count==MAXVAL则表示此时没有读者在读，可以写，则发送写允许信号。否则表示此时有读者在读，则延迟对写者的响应，同时将count值-MAXVAL，此时count＜0  如果count=0，说明有1个写者在写，则尝试接收FINISHED消息，如果接收到表示写者完成写，则将count置为MAXVAL并打印输出；如果没有接收到则**阻塞等待（参数0）**  如果count＜0，则表示有读者在读而且写者请求写，则阻塞等待消息FINISHED的到来，如果来一个finish则count++，直到加到count==0退出while循环。  参数IPC\_NOWAIT，如果消息队列已满，消息将不被写入队列，控制权返回调用函数的线程。如果不指定这个参数，线程将被阻塞直到消息被可以被写入。  后面的if语句中参数为0，表示有需要延迟处理的消息发送信号，即需要发送信号表明请求读。随后打印输出到控制台，进行下一次循环  4.此程序完成的任务：  **1. 任意多个读者可以同时读；**  **Count设置成了最大读者数，只要没有写者的干预，所有读者发送读申请信号后都可以保证count＞0，而且会让count--；但由于count是最大读者数，因此不会减到≤0。**  **2. 任意时刻只能有一个写者写；**  **如果此时没有读者写者在读/写，则写者申请后将count值减到0并开始写**  **如果此时有读者在读，写者申请后会循环等待着申请前的读者们读结束了开始写**  **如果此时有写者在写，count=0，则由于以阻塞方式接收消息，因此其他写者不可能再次申请到写的权限。**  **3. 如果写者正在写，那么读者就必须等待；**  **写者在写，count=0，则会无法进入写者可以发送请求的count>0或者count<0的条件语句**  **4. 如果读者正在读，那么写者也必须等待；**  **读者在读而写着申请后，count会<0，随后进入了while循环分支一直接收读者完成消息直到读者全部完成**  **5. 允许写者优先；**  **由于写者请求到来后会将count值首先改为<0（如果此时有读者申请读），随后会先完成写者请求将count恢复到＞0**  **6. 防止读者或写者发生饥饿。**  **由于每次循环都会先判断读者再判断写者，而且写者有较高优先级，所以都不会饥饿**  【独立实验分析】  基本思想和示例程序相似，主要实现思路如下：  理发师程序由一个父进程及其创建的两个子进程构成三个理发师；每个理发师做如下事情：  如果有请求送来，则开始服务；随后开始结账，并且在结账时利用down和up函数实现PV操作，以确保在同一时刻只有一个人可以结账；  顾客做如下事情：  首先判断沙发是否满，如果没有则坐上沙发并且申请理发；如果满了则到等候室，并且接收来自沙发的消息。如果消息到来则从等候室来到沙发。  如果沙发满了而等候室没有满，则发送消息申请沙发；  如果都满了则打印无法进入理发店。  如果接受到沙发发来的消息则让沙发数--，收到等候室发来的消息则让等候室--  【结论分析】  解决饥饿现象：  解决方案：  (1)设置优先级上限：凡是进入临界区的进程，给它个高的优先级，便于先执行完，把临界区的控制权还回去，不进临界区的，给个低优先级；  (2)优先级继承：低的阻碍了高的，他可以继承这个高的优先级，先把任务执行完，把临界区还回去  (3)使用中断禁止：凡是进入临界区的进程就不再响应中断了，直到他出了临界区才响应中断，就保护了这个进程，让他去执行，直到把临界区还回去  在本次实验中，通过设置写者优先级高于读者来解决：读者有请求后会让count＜0，随后存在的读者结束读后便可以开始写。 | | | |