实验五、进程互斥实验

5.1 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。

5.2 实验说明

以下示例实验程序应能模拟一个读者 / 写者问题，它应能实现以下功能：

1. 任意多个读者可以同时读；
2. 任意时刻只能有一个写者写；
3. 如果写者正在写，那么读者就必须等待；
4. 如果读者正在读，那么写者也必须等待；
5. 允许写者优先；
6. 防止读者或写者发生饥饿。

为了能够体验 IPC 机制的消息队列的用法，本示例程序采用了 Theaker & Brookes 提出的消息传递算法。该算法中有一控制进程，带有 3 个不同类型的消息信箱，它们分别是：读请求信箱、写请求信箱和操作完成信箱。

读者需要访问临界资源时首先要向控制进程发送读请求消息，写者需要访问临界资源时也要先向控制进程发送写请求消息，在得到控制进程的允许消息后方可进入临界区读或写。读或写者在完成对临界资源的访问后还要向控制进程发送操作完成消息。

控制进程使用一个变量 count 控制读写者互斥的访问临界资源并允许写者优先。count 的初值需要一个比最大读者数还要大的数，本例取值为 100。当 count 大于 0 时说明没有新的读写请求，控制进程接收读写者新的请求，如果收到读者完成消息，对 count 的值加 1，如果收到写者请求消息， count 的值减 100，如果收到读者请求消息，对 count 的值减 1。 当 count 等于 0 时说明写者正在写，控制进程等待写者完成后再次令 count 的值等于 100。当 count 小于 100 时说明读者正在读，控制进程等待读者完成后对 count 的值加 1。

5.3 示例实验

我们可以利用上节实验中介绍的 IPC 机制中的消息队列来实验一下以上使用消息传递算法的读写者问题的解法，看其是否能够满足我们的要求。仍采用共享内存模拟要读写的对象，一写者向共享内存中写入一串字符后，多个读者可同时从共享内存中读出该串字符。

1. 在新建的文件夹中建立以下 ipc.h 头文件

```
/*
* Filename  : ipc.h
* copyright : (C) 2006 by zhonghonglie
* Function  : 声明 IPC 机制的函数原型和全局变量
*/
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/msg.h>
#define BUFSZ 256
#define MAXVAL  100
#define STRSIZ  8
#define WRITERQUEST   1 //写请求标识
#define READERQUEST   2 //读请求标识
#define FINISHED  3 //读写完成标识
/*信号灯控制用的共同体*/
typedef union semuns {
    int val;
} Sem_uns;

/* 消 息 结 构 体 */
typedef struct msgbuf {
    long mtype;
    int mid;
} Msg_buf;
key_t buff_key;
int buff_num;
char *buff_ptr;
int shm_flg;

int quest_flg;
key_t quest_key;
int quest_id;

int respond_flg;
key_t respond_key;
int respond_id;

int get_ipc_id(char *proc_file, key_t key);

char *set_shm(key_t shm_key, int shm_num, int shm_flag);
int set_msq(key_t msq_key, int msq_flag);
int set_sem(key_t sem_key, int sem_val, int sem_flag);
int down(int sem_id);

int up(int sem_id);
```

2. 将消费者生产者问题实验中 ipc.c 文件拷贝到当前目录中。

3. 在当前目录中建立如下的控制者程序 control.c

```c
/*
*
* Filename  : control.c
* copyright : (C) 2006 by zhonghonglie
* Function  : 建立并模拟控制者进程
*/
#include "ipc.h"
int main(int argc, char *argv[]) {
    int i; int rate;
    int w_mid;
    int count = MAXVAL;
    Msg_buf  msg_arg;
    struct msqid_ds msg_inf;
    //建立一个共享内存先写入一串 A 字符模拟要读写的内容
    buff_key = 101;
    buff_num = STRSIZ + 1;
    shm_flg = IPC_CREAT | 0644;
    buff_ptr = (char *)set_shm(buff_key, buff_num, shm_flg);
    for (i = 0; i < STRSIZ; i++) buff_ptr[i] = 'A';
    buff_ptr[i] = '\0';

    //建立一条请求消息队列
    quest_flg = IPC_CREAT | 0644;
    quest_key = 201;
    quest_id = set_msq(quest_key, quest_flg);

    //建立一条响应消息队列
    respond_flg = IPC_CREAT | 0644;
    respond_key = 202;
    respond_id = set_msq(respond_key, respond_flg);

    //控制进程准备接收和响应读写者的消息
    printf("Wait quest \n");
    while (1) {
        //当 count 大于 0 时说明没有新的读写请求，查询是否有任何新请求
        if (count > 0) {
            quest_flg = IPC_NOWAIT; //以非阻塞方式接收请求消息
            if (msgrcv(quest_id, &msg_arg, sizeof(msg_arg), FINISHED,
quest_flg) >= 0) {
                //有读者完成
                count++;
                printf("%d reader finished\n", msg_arg.mid);
            } else if (msgrcv(quest_id, &msg_arg, sizeof(msg_arg),
READERQUEST, quest_flg) >= 0) {
                //有读者请求，允许读者读
                count --;
                msg_arg.mtype = msg_arg.mid;
                msgsnd(respond_id, &msg_arg, sizeof(msg_arg), 0);
```

```
                printf("%d quest read\n", msg_arg.mid);
            } else if (msgrcv(quest_id, &msg_arg, sizeof(msg_arg), WRITERQUEST, quest_flg) >= 0) {
                //有写者请求
                w_mid = msg_arg.mid;
                count -= MAXVAL;

                //有读者正在读，则等待所有读者读完
                while(count < 0) {
                //以阻塞方式接收读完成消息
                    msgrcv(quest_id, &msg_arg, sizeof(msg_arg), FINISHED, 0);
                    count ++;
                    printf("%d reader finish\n", msg_arg.mid);
                }

                //允许写者写
                msg_arg.mtype = w_mid;
                // 恢复被读者覆盖的 mid
                msg_arg.mid = w_mid;
                msgsnd(respond_id, &msg_arg, sizeof(msg_arg), 0);
                printf("%d quest write \n", msg_arg.mid);
            }
        }
        //当 count 等于 0 时说明写者正在写，等待写完成
        if (count == 0) {
            //以阻塞方式接收消息.
            msgrcv(quest_id, &msg_arg, sizeof(msg_arg), FINISHED, 0);
            count = MAXVAL;
            printf("%d write finished\n", msg_arg.mid);
            if (msgrcv(quest_id, &msg_arg, sizeof(msg_arg), READERQUEST, quest_flg) >= 0) {
                //有读者请求，允许读者读
                count --;
                msg_arg.mtype = msg_arg.mid;
                msgsnd(respond_id, &msg_arg, sizeof(msg_arg), 0);
                printf("%d quest read\n", msg_arg.mid);
            }
        }
    }
    return EXIT_SUCCESS;
}
```

4. 在当前目录中建立如下的读者程序 reader.c

```
/*
* Filename  : reader.c
* copyright : (C) 2006 by zhonghonglie
```

```
* Function  : 建立并模拟读者进程
*/
#include "ipc.h"

int main(int argc, char *argv[]) {
    int i; int rate;
    Msg_buf msg_arg;

    //可在在命令行第一参数指定一个进程睡眠秒数，以调解进程执行速度
    if (argv[1] != NULL) rate = atoi(argv[1]);
    else rate = 3;
    //附加一个要读内容的共享内存
    buff_key = 101;
    buff_num = STRSIZ + 1;
    shm_flg = IPC_CREAT | 0644;
    buff_ptr = (char *)set_shm(buff_key, buff_num, shm_flg);
    //联系一个请求消息队列
    quest_flg = IPC_CREAT | 0644;
    quest_key = 201;
    quest_id =  set_msq(quest_key, quest_flg);
    //联系一个响应消息队列
    respond_flg = IPC_CREAT | 0644;
    respond_key = 202;
    respond_id =  set_msq(respond_key, respond_flg);
    //循环请求读
    msg_arg.mid = getpid();
    while (1) {
        //发读请求消息
        msg_arg.mtype = READERQUEST;
        msgsnd(quest_id, &msg_arg, sizeof(msg_arg), 0);
        printf("%d reader quest\n", msg_arg.mid);
        //等待允许读消息
        msgrcv(respond_id, &msg_arg, sizeof(msg_arg), msg_arg.mid, 0);
        printf("%d reading: %s\n", msg_arg.mid, buff_ptr);
        sleep(rate);
        //发读完成消息
        msg_arg.mtype = FINISHED;
        msgsnd(quest_id, &msg_arg, sizeof(msg_arg), quest_flg);
    }
    return EXIT_SUCCESS;
}
```

5. 在当前目录中建立如下的写者程序 writer.c

```
/*
* Filename  : writer.c
* copyright : (C) 2006 by zhonghonglie
* Function  : 建立并模拟写者进程
```

```
*/
#include "ipc.h"

int main(int argc, char *argv[]) {
    int i, j = 0; int rate;
    Msg_buf msg_arg;

    //可在在命令行第一参数指定一个进程睡眠秒数，以调解进程执行速度
    if (argv[1] != NULL) rate = atoi(argv[1]);
    else rate = 3;
    //附加一个要读内容的共享内存
    buff_key = 101;
    buff_num = STRSIZ + 1;
    shm_flg = IPC_CREAT | 0644;
    buff_ptr = (char *)set_shm(buff_key, buff_num, shm_flg);
    //联系一个请求消息队列
    quest_flg = IPC_CREAT | 0644;
    quest_key = 201;
    quest_id =  set_msq(quest_key, quest_flg);
    //联系一个响应消息队列
    respond_flg = IPC_CREAT | 0644;
    respond_key = 202;
    respond_id =  set_msq(respond_key, respond_flg);
    //循环请求写
    msg_arg.mid = getpid();
    while (1) {
        //发写请求消息
        msg_arg.mtype = WRITERQUEST;
        msgsnd(quest_id, &msg_arg, sizeof(msg_arg), 0);
        printf("%d writer quest\n", msg_arg.mid);
        //等待允许写消息
        msgrcv(respond_id, &msg_arg, sizeof(msg_arg), msg_arg.mid, 0);
        //写入 STRSIZ 个相同的字符
        for (i = 0; i < STRSIZ; i++) buff_ptr[i] = 'A' + j;
        j = (j + 1) % STRSIZ ; //按 STRSIZ 循环变换字符
        printf("%d writing: %s\n", msg_arg.mid, buff_ptr);
        sleep(rate);
        //发写完成消息
        msg_arg.mtype = FINISHED;
        msgsnd(quest_id, &msg_arg, sizeof(msg_arg), 0);
    }
    return EXIT_SUCCESS;
}
```

6. 在当前目录中建立如下 Makefile 文件

```
hdrs = ipc.h
c_src = control.c ipc.c
```

```
c_obj = control.o ipc.o
r_src = reader.c ipc.c
r_obj = reader.o ipc.o
w_src = writer.c ipc.c
w_obj = writer.o ipc.o
opts    = -g -c
all:    control reader writer
control: $(c_obj)
          gcc $(c_obj) -o control
control.o: $(c_src) $(hdrs)
          gcc $(opts) $(c_src)

reader: $(r_obj)
            gcc $(r_obj) -o reader
reader.o: $(r_src) $(hdrs)
            gcc $(opts) $(r_src)

writer: $(w_obj)
            gcc $(w_obj) -o writer
writer.o:$(w_src) $(hdrs)
            gcc $(opts) $(w_src)

clean:
        rm control reader writer *.o

```

7. 在当前目录中执行 make 命令编译连接，生成读写者，控制者程序：

```
$ make
gcc -g -c control.c ipc.c
gcc control.o ipc.o -o control
gcc -g -c reader.c ipc.c
gcc reader.o ipc.o -o reader
gcc -g -c writer.c ipc.c
gcc writer.o ipc.o -o writer
```

8. 可打开四个以上的终端模命令窗体，都将进入当前工作目录。先在一窗体中启动./control 程序：

```
$ ./control
Wait quest
```

现在控制进程已经在等待读写者的请求。

9. 再在另两不同的窗体中启动两个读者，一个让它以 1 秒的延迟快一些读，一个让它以 10 秒的延迟慢一些读：

```
$ ./reader 10

3903 reader quest

3903 reading: AAAAAAAA

......
```

```
$ ./reader 1

3904 reader quest

3904 reading: AAAAAAAA

3904 reader quest

3904 reading: AAAAAAAA

3904 reader quest

3904 reading: AAAAAAAA

3904 reader quest

......
```

现在可以看到控制进程开始响应读者请求，让多个读者同时进入临界区读：

```
Wait quest

3903 quest read

3904 quest read

3904 quest read

3904 reader finished

3904 reader finished

3904 quest read

3904 reader finished

3903 reader finished

......
```

10. 再在另一终端窗体中启动一个延迟时间为 8 秒的写者进程：

```
$ ./writer 8

3906 writer quest

3906 writing: AAAAAAAA

3906 writer quest

3906 writing: BBBBBBBB

3906 writer quest

3906 writing: CCCCCCCC

......
```

此时可以看到控制进程在最后一个读者读完后首先响应写者请求：

```

3906 quest write

3904 reader finish

3903 reader finish

3906 write finished

......
```

在写者写完后两个读者也同时读到了新写入的内容：

```

3903 reader quest

3903 reading: BBBBBBBB

3903 reader quest

3903 reading: CCCCCCCC

......

3904 reading: BBBBBBBB

3904 reader quest

```

```
3904 reading: CCCCCCCC

......
```

请仔细观察各读者和写者的执行顺序：可以看出在写者写时不会有读者进入，在有读者读时不会有写者进入，但一旦读者全部退出写者会首先进入。分析以上输出可以看出该算法实现了我们要求的读写者问题的功能。

11. 请按与以上不同的启动顺序、不同的延迟时间，启动更多的读写者。观察和分析是否仍能满足我们要求的读写者问题的功能。

12. 请修改以上程序，制造一个读者或写者的饥饿现象。观察什么是饥饿现象，说明为什么会发生这种现象。

5.4 独立实验

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，即直接离开。

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。

5.5 实验要求

总结和分析示例实验和独立实验中观察到的调试和运行信息，思考以下问题：

1. 说明您对与解决非对称性互斥操作的算法有哪些新的理解和认识?
2. 为什么会出现进程饥饿现象?
3. 本实验的饥饿现象是怎样表现的?
4. 怎样解决并发进程间发生的饥饿现象?
5. 您对于并发进程间使用消息传递解决进程通信问题有哪些新的理解和认识?

根据实验程序、调试过程和结果分析写出实验报告。