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：201705130120 | 姓名：苑宗鹤 | | 班级： 1班 |
| 实验题目： 进程同步实验 | | | |
| 实验学时：2 | | 实验日期：2019/12/22 | |
| 实验目的：  加深对并发协作进程同步与互斥概念的理解，观察和体验并发进程同步与互斥 操作的效果，分析与研究经典进程同步与互斥问题的实际解决方案。了解 Linux 系 统中 IPC 进程同步工具的用法，练习并发协作进程的同步与互斥操作的编程与调试 技术。 | | | |
| 硬件环境：  pc | | | |
| 软件环境：  Win10 clion  Linux ubuntu | | | |
| 实验内容与设计：  抽烟者问题。假设一个系统中有三个抽烟者进程，每个抽烟者不断地卷烟并抽 烟。抽烟者卷起并抽掉一颗烟需要有三种材料：烟草、纸和胶水。一个抽烟者有烟 草，一个有纸，另一个有胶水。系统中还有两个供应者进程，它们无限地供应所有 三种材料，但每次仅轮流提供三种材料中的两种。得到缺失的两种材料的抽烟者在 卷起并抽掉一颗烟后会发信号通知供应者，让它继续提供另外的两种材料。这一过 程重复进行。 请用以上介绍的 IPC 同步机制编程，实现该问题要求的功能。  实验代码： ipc.h *// // Created by Yuan on 18/12/2019. //* #ifndef **PROJECT3\_IPC\_H** #define **PROJECT3\_IPC\_H** #define **BUFSZ** 256 key\_t buff\_key; **int** buff\_num; **char** \*buff\_ptr; key\_t pput\_key; **int** pput\_num; **int** \*pput\_ptr;  key\_t mutexkey; key\_t smoker1key; key\_t smoker2key; key\_t smoker3key; key\_t finish1k; key\_t finish2k; key\_t finish3k;  **int** mutex; **int** mutexForSeller; **int** smoker1; **int** smoker2; **int** smoker3; **int** finish1; **int** finish2; **int** finish3;    **int** sem\_val; **int** sem\_flg;   **int** get\_ipc\_id(**char** \*proc\_file, key\_t key); **typedef union** semuns {  **int** val; } Sem\_uns;  **int** set\_sem(key\_t sem\_key, **int** sem\_val, **int** sem\_flg) {  **int** sem\_id;  Sem\_uns sem\_arg;  **if** ((sem\_id = semget(sem\_key, 1, sem\_flg)) < 0) {  perror(**"semaphore create error"**);  exit(**EXIT\_FAILURE**);  }  sem\_arg.val = sem\_val;  **if** (semctl(sem\_id, 0, **SETVAL**, sem\_arg) < 0) {  perror(**"semaphore set error"**);  exit(**EXIT\_FAILURE**);  }  **return** sem\_id; } **int** wait(**int** sem\_id) {  **struct** sembuf buf;  buf.sem\_op = -1;  buf.sem\_num = 0;  buf.sem\_flg = **SEM\_UNDO**;  **if** ((semop(sem\_id, &buf, 1)) < 0) {  perror(**"down error "**);  exit(**EXIT\_FAILURE**);  }  **return EXIT\_SUCCESS**; }  **int** signal(**int** sem\_id) {  **struct** sembuf buf;  buf.sem\_op = 1;  buf.sem\_num = 0;  buf.sem\_flg = **SEM\_UNDO**;  **if** ((semop(sem\_id, &buf, 1)) < 0) {  perror(**"up error "**);  exit(**EXIT\_FAILURE**);  }  **return EXIT\_SUCCESS**; }  **char** \* set\_shm(key\_t shm\_key,**int** shm\_num,**int** shm\_flg) {  */\*  创建具有n字节的共享内存,成功返回首地址指针。  key指定key,val指定长度，flag指定权限  \*/* **int** i,shm\_id;  **char** \* shm\_buf;  *//测试由 shm\_key 标识的共享内存区是否已经建立* **if**((shm\_id = get\_ipc\_id(**"/proc/sysvipc/shm"**,shm\_key)) < 0 )  {  *//shmget 新建 一个长度为 shm\_num 字节的共享内存,其标号返回到 shm\_id* **if**((shm\_id = shmget(shm\_key,shm\_num,shm\_flg)) <0)  {  perror(**"shareMemory set error"**);  exit(**EXIT\_FAILURE**);  }  *//shmat 将由 shm\_id 标识的共享内存附加给指针 shm\_buf* **if**((shm\_buf = (**char** \*)shmat(shm\_id,0,0)) < (**char** \*)0)  {  perror(**"get shareMemory error"**);  exit(**EXIT\_FAILURE**);  }  **for**(i=0; i<shm\_num; i++) shm\_buf[i] = 0; *//初始为 0* }  *//shm\_key 标识的共享内存区已经建立,将由 shm\_id 标识的共享内存附加给指针 shm\_buf* **if**((shm\_buf = (**char** \*)shmat(shm\_id,0,0)) < (**char** \*)0)  {  perror(**"get shareMemory error"**);  exit(**EXIT\_FAILURE**);  }  **return** shm\_buf; } **int** get\_ipc\_id(**char** \*proc\_file,key\_t key) {  */\*  key是要获取IPC的id号。  PROC\_FILE对应三个文件，msg,sem,shm  原理就是循环取key进行判断，如果存在返回id，否则返回-1.  \*/* FILE \*pf;  **int** i,j;  **char** line[**BUFSZ**],colum[**BUFSZ**];  **if**((pf = fopen(proc\_file,**"r"**)) == **NULL**)  {  perror(**"Proc file not open"**);  exit(**EXIT\_FAILURE**);  }  fgets(line, **BUFSZ**,pf);  **while**(!feof(pf))  {  i = j = 0;  fgets(line, **BUFSZ**,pf);  **while**(line[i] == **' '**) i++;  **while**(line[i] !=**' '**) colum[j++] = line[i++];  colum[j] = **'\0'**;  **if**(atoi(colum) != key) **continue**;  j=0;  **while**(line[i] == **' '**) i++;  **while**(line[i] !=**' '**) colum[j++] = line[i++];  colum[j] = **'\0'**;  i = atoi(colum);  fclose(pf);  **return** i;  }  fclose(pf);  **return** -1; } #endif *//PROJECT3\_IPC\_H*  producer.h #include **<stdio.h>** #include **<stdlib.h>** #include **<sys/types.h>** #include **<sys/ipc.h>** #include **<sys/shm.h>** #include **<sys/sem.h>** #include **<sys/msg.h>** #include **<unistd.h>** #include**"ipc.h"   int** main(**int** argc, **char** \*argv[]) {  **int** type =0;  **if**(argc>=2){  type=atoi(argv[1]) - 1;  }  **int** combine1 = 0;  printf(**"producer %d\n"**,type+1);   mutexkey = 101;  smoker1key = 102;  smoker2key = 103;  smoker3key = 104;  finish1k = 201;  finish2k = 202;  finish3k = 203;   buff\_key = 301;  buff\_num = 2;    sem\_flg = **IPC\_CREAT** | 0644;   buff\_ptr = (**char** \*)set\_shm(buff\_key,buff\_num,sem\_flg);   sem\_val = 1;   mutex = set\_sem(mutexkey, sem\_val, sem\_flg);   sem\_val = 0;  smoker1 = set\_sem(smoker1key, sem\_val, sem\_flg);  smoker2 = set\_sem(smoker2key, sem\_val, sem\_flg);  smoker3 = set\_sem(smoker3key, sem\_val, sem\_flg);  finish1 = set\_sem(finish1k, sem\_val, sem\_flg);  finish2 = set\_sem(finish2k, sem\_val, sem\_flg);  finish3 = set\_sem(finish3k, sem\_val, sem\_flg);   **int** cnt=0;  **int** finish[3]={finish1,finish2,finish3};  **int** smoker[3]={smoker1,smoker2,smoker3};   **while** (cnt<1000) {  wait(mutex);  buff\_ptr[type]=**'A'**+(cnt+type)%3;  signal(mutex);  printf(**"producer %d for somker %d\n"**,type+1,cnt+1);  *//拉起吸烟者* signal(smoker[cnt]);  *//等待吸烟结束* wait(finish[cnt]);  cnt++;  cnt%=3;  }  **return EXIT\_SUCCESS**; } smoker.cpp #include **<stdio.h>** #include **<stdlib.h>** #include **<sys/types.h>** #include **<sys/ipc.h>** #include **<sys/shm.h>** #include **<sys/sem.h>** #include **<sys/msg.h>** #include **<unistd.h>** #include**"ipc.h"   int** main(**int** argc, **char** \*argv[]) {   **int** combine1 = 0;  **int** type =0;  **if**(argc>=2){  type=atoi(argv[1]) - 1;  }  printf(**"smoker %d\n"**,type+1);  mutexkey = 101;  smoker1key = 102;  smoker2key = 103;  smoker3key = 104;  finish1k = 201;  finish2k = 202;  finish3k = 203;  buff\_key = 301;  buff\_num = 2;    sem\_flg = **IPC\_CREAT** | 0644;   buff\_ptr = (**char** \*)set\_shm(buff\_key,buff\_num,sem\_flg);   sem\_val = 1;  mutex = set\_sem(mutexkey, sem\_val, sem\_flg);   sem\_val = 0;  smoker1 = set\_sem(smoker1key, sem\_val, sem\_flg);  smoker2 = set\_sem(smoker2key, sem\_val, sem\_flg);  smoker3 = set\_sem(smoker3key, sem\_val, sem\_flg);  finish1 = set\_sem(finish1k, sem\_val, sem\_flg);  finish2 = set\_sem(finish2k, sem\_val, sem\_flg);  finish3 = set\_sem(finish3k, sem\_val, sem\_flg);   **int** finish[3] = {finish1, finish2, finish3};  **int** smoker[3] = {smoker1, smoker2, smoker3};   **while** (type<4) {  wait(smoker[type]);  wait(smoker[type]); *// wait(erforsmoker2);* sleep(3);   printf(**"somker %d have somked %c %c\n"**,type+1,buff\_ptr[0],buff\_ptr[1]);  *///告诉两个进程我已吸完，所以signal两次* signal(finish[type]);  signal(finish[type]);    }  **return EXIT\_SUCCESS**; } server.py 同时拉起5个进程 **import** multiprocessing **import** os **import** signal   **def** produce(id):  os.system(**"/tmp/tmp.FPB8OXWmxk/cmake-build-debug/project3 "** + str(id))   **def** smoker(id):  os.system(**"/tmp/tmp.dr3A3550ut/cmake-build-debug/project31 "** + str(id))   **def** sigint\_handler(signum, frame):  **global** p  **for** i **in** p:  i.terminate()  **global** s  **for** i **in** s:  i.terminate()  print(**"end"**)   **if** \_\_name\_\_ == **"\_\_main\_\_"**:  s = []  **for** i **in** range(3):  s.append(multiprocessing.Process(target=smoker, args=(str(i + 1),)))  s[i].start()  signal.signal(signal.SIGINT, sigint\_handler)  p = []  **for** i **in** range(2):  p.append(multiprocessing.Process(target=produce, args=(str(i + 1),)))  p[i].start()  运行结果： | | | |
| 结论分析与体会：  smoker接受到两个producer发来的就绪信号后才开始工作  两个producer互斥 按各自的顺序向buff[0] buff[1]中添入数据  smoker工作完成后发信号拉起两个producer | | | |