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： 死锁问题实验 | | | |
| 实验学时：2 | | 实验日期：2019/12/22 | |
| 实验目的：  通过本实验观察死锁产生的现象，考虑解决死锁问题的方法。从而进一步加深对 于死锁问题的理解。掌握解决死锁问题的几种算法的编程和调试技术。练习怎样构 造管程和条件变量，利用管程机制来避免死锁和饥俄问题的发生。 | | | |
| 硬件环境：  pc | | | |
| 软件环境：  Win10 clion  Linux ubuntu | | | |
| 实验内容与设计：  在两个城市南北方向之间存在一条铁路，多列火车可以分别从两个城市的车站 排队等待进入车道向对方城市行驶，该铁路在同一时间，只能允许在同一方向上行 车，如果同时有相向的火车行驶将会撞车。请模拟实现两个方向行车，而不会出现 撞车或长时间等待的情况。您能构造一个管程来解决这个问题吗？  实验代码： main.cpp #include **<iostream>** #include **"dp.h" int** main() {  **int** maxCars=1000;  **int** maxSingelDirect=10;   OneWay \*oneWay = **new** OneWay(maxCars, maxSingelDirect); *//建立管程,判断可不可进、决定方向,进入单行道* **int** i;  **int** pid[maxCars];  **for** (i = 0; i < maxCars; i++) *//创建子进程* {  sleep(1);  pid[i] = fork();  **if** (pid[i] == 0)  {  srand(time(**NULL**));  **int** direct = **int**(rand())%2; *//决定东西方向* oneWay->Arrive(direct); *//进入* oneWay->Cross(direct); *//通过* oneWay->Quit(direct); *//离开* exit(**EXIT\_SUCCESS**);  }  }  **for** (i = 0; i < maxCars; i++)  {  waitpid(pid[i], **NULL**, 0);  }   **delete** oneWay;  **return EXIT\_SUCCESS**;   } dp.h *// // Created by qq295 on 2019/12/18. //* #ifndef **PROJECT6\_DP\_H** #define **PROJECT6\_DP\_H** #include **<iostream>** #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>** #include **<sys/wait.h>  using namespace** std; */\*信号灯控制用的共同体\*/* **typedef union** semuns {  **int** val; } Sem\_uns;  *//管程中使用的信号量* **class** Sema { **public**:  Sema(**int** id);   ~Sema();   **int** down(); *//信号量加 1* **int** up(); *//信号量减 1* **private**:  **int** sem\_id; *//信号量标识符* };  *//管程中使用的锁* **class** Lock { **public**:  Lock(Sema \*lock);   ~Lock();   **void** close\_lock();   **void** open\_lock();  **private**:  Sema \*sema; *//锁使用的信号量* };  **class** Condition { **public**:  Condition(Sema \*sema1, Sema \*sema2);   ~Condition();   **void** Wait(Lock \*conditionLock, **int** direct); *//过路条件不足时阻塞* **int** Signal(**int** direc); *//唤醒相反方向阻塞车辆* **private**:  Sema \*sema0; *// 一个方向阻塞队列* Sema \*sema1; *// 另一方向阻塞队列* Lock \*lock; *// 进入管程时获取的锁* };  **class** OneWay { **public**:  OneWay(**int** maxall, **int** maxcur);   ~OneWay();   **void** Arrive(**int** direc);  *// 车辆准备上单行道,direc 为行车方向* **void** Cross(**int** direc);  *// 车辆正在单行道上* **void** Quit(**int** direc);  *// 车辆通过了单行道* **int** \*eastCount;  **int** \*westCount;  **int** \*eastWait;  **int** \*westWait;  **int** \*sumPassedCars;*//已经通过的车辆总数* **private**: *//建立或获取 ipc 信号量的一组函数的原型说明* **int** get\_ipc\_id(**char** \*proc\_file, key\_t key);   **int** set\_sem(key\_t sem\_key, **int** sem\_val, **int** sem\_flag);  *//创建共享内存* **char** \*set\_shm(key\_t shm\_key, **int** shm\_num, **int** shm\_flag);   **int** rate; *//车速* **int** \*maxCars;*//最大同向车数* **int** \*numCars; *//当前正在通过的车辆数* **int** \*currentDire;*//当前通过的车辆的方向* Condition \*condition; *//通过单行道的条件变量* Lock \*lock;*//单行道管程锁* };   **using namespace** std;  Sema::Sema(**int** id) {  sem\_id = id; }  Sema::~Sema() {  }  */\* \* 信号灯上的 down/up 操作 \* semid:信号灯数组标识符 \* semnum:信号灯数组下标 \* buf:操作信号灯的结构 \*/* **int** Sema::down() {  **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** Sema::up() {  Sem\_uns arg;  **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**; }  */\* \* 用于单行道管程的互斥执行 \*/* Lock::Lock(Sema \*s) {  sema = s; }  Lock::~Lock() {  }  *//上锁* **void** Lock::close\_lock() {  sema->down(); }  *//开锁* **void** Lock::open\_lock() {  sema->up(); }  **int** OneWay::get\_ipc\_id(**char** \*proc\_file, key\_t key) { #define **BUFSZ** 256  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; }  */\* \* set\_shm 函数建立一个具有 n 个字节 的共享内存区 \* 如果建立成功,返回 一个指向该内存区首地址的指针 shm\_buf \* 输入参数: \* shm\_key 共享内存的键值 \* shm\_val 共享内存字节的长度 \* shm\_flag 共享内存的存取权限 \*/* **char** \*OneWay::set\_shm(key\_t shm\_key, **int** shm\_num, **int** shm\_flg) {  **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* **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* } *//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; }  */\* \* set\_sem 函数建立一个具有 n 个信号灯的信号量 \* 如果建立成功,返回 一个信号量的标识符 sem\_id \* 输入参数: \* sem\_key 信号量的键值 \* sem\_val 信号量中信号灯的个数 \* sem\_flag 信号量的存取权限 \*/* **int** OneWay::set\_sem(key\_t sem\_key, **int** sem\_val, **int** sem\_flg) {  **int** sem\_id;  Sem\_uns sem\_arg; *//测试由 sem\_key 标识的信号量是否已经建立* **if** ((sem\_id = get\_ipc\_id(**"/proc/sysvipc/sem"**, sem\_key)) < 0) { *//semget 新建一个信号灯,其标号返回到 sem\_id* **if** ((sem\_id = semget(sem\_key, 1, sem\_flg)) < 0) {   perror(**"semaphore create error"**);  exit(**EXIT\_FAILURE**);  }  } *//设置信号量的初值* sem\_arg.val = sem\_val;  **if** (semctl(sem\_id, 0, **SETVAL**, sem\_arg) < 0) {   perror(**"semaphore set error"**);   exit(**EXIT\_FAILURE**);  }  **return** sem\_id; }  Condition::Condition(Sema \*semax1, Sema \*semax2) {  sema0 = semax1;  sema1 = semax2; }  */\*\* \* 看看是否能通过 \*/* **void** Condition::Wait(Lock \*lock, **int** direc) {  **if** (direc == 0) {  cout << getpid() << **" wait ->"** << **"\n"**;  lock->open\_lock();  sema0->down();  lock->close\_lock();  } **else if** (direc == 1) {  cout << getpid() << **" wait <-"** << **"\n"**;  lock->open\_lock();  sema1->down();  lock->close\_lock();  } }  **int** Condition::Signal(**int** direc) {  **int** i;  **if** (direc == 0) *//唤醒一个方向* {  i = sema0->up();  } **else if** (direc == 1) {  i = sema1->up();  }  **return** i; }  */\* \* get\_ipc\_id() 从/proc/sysvipc/文件系统中获取 IPC 的 id 号 \* pfile: 对应/proc/sysvipc/目录中的 IPC 文件分别为 \* msg-消息队列,sem-信号量,shm-共享内存 \* key: 对应要获取的 IPC 的 id 号的键值 \*/* Condition::~Condition() {  };  */\* \* set\_shm 函数建立一个具有 n 个字节 的共享内存区 \* 如果建立成功,返回 一个指向该内存区首地址的指针 shm\_buf \* 输入参数: \* shm\_key 共享内存的键值 \* shm\_val 共享内存字节的长度 \* shm\_flag 共享内存的存取权限 \*/* OneWay::OneWay(**int** maxall, **int** maxcur) {  Sema \*sema0;  Sema \*sema1;  Sema \*semaLock;  **int** ipc\_flg = **IPC\_CREAT** | 0644;  maxCars = (**int** \*) set\_shm(100, 1, ipc\_flg); *//最大单向* numCars = (**int** \*) set\_shm(200, 1, ipc\_flg); *//当前方向上通过的总的车辆数* currentDire = (**int** \*) set\_shm(300, 1, ipc\_flg); *//当前方向 0 东 1 西* eastCount = (**int** \*) set\_shm(501, 1, ipc\_flg);  westCount = (**int** \*) set\_shm(502, 1, ipc\_flg);  sumPassedCars = (**int** \*) set\_shm(700, 1, ipc\_flg);  eastWait = (**int** \*) set\_shm(801, 1, ipc\_flg);  westWait = (**int** \*) set\_shm(802, 1, ipc\_flg);  **int** sema0\_id = set\_sem(401, 0, ipc\_flg);  **int** sema1\_id = set\_sem(402, 0, ipc\_flg);  **int** semaLock\_id = set\_sem(601, maxcur, ipc\_flg); *//初始化所有变量* \*maxCars = maxcur;  \*numCars = 0;  \*currentDire = 0;  \*eastCount = 0;  \*westCount = 0;  \*sumPassedCars = 0;  \*eastWait = 0;  \*westWait = 0;  sema0 = **new** Sema(sema0\_id);  sema1 = **new** Sema(sema1\_id);  semaLock = **new** Sema(semaLock\_id);  lock = **new** Lock(semaLock);  condition = **new** Condition(sema0, sema1); }  **void** OneWay::Arrive(**int** direc) {  cout << **"car born "** << getpid() << (direc == 0 ? **" >>"** : **" <<"**) << **"🚗 "** << endl;   lock->close\_lock();   *//堆积车辆过多 换向* **if** ((\*currentDire != direc || \*numCars >= \*maxCars)||(\*sumPassedCars >= 3&&(\*eastWait>3 || \*westWait>3) && \*numCars > 0)) {  \*sumPassedCars=\*sumPassedCars+1;  cout<<**"sum="**<<\*sumPassedCars<<endl;  cout<<**"wait= "**<<\*westWait <<**" "**<<\*eastWait<<endl;  **if** (\*sumPassedCars == 3) {  cout << getpid() << **"start stop all================="** << endl;  }  **if** (direc == 0) {  \*eastWait += 1;  } **else if** (direc == 1) {  \*westWait += 1;  }  condition->Wait(lock, direc);  **if**( \*sumPassedCars>3 )\*sumPassedCars= 0;  }   cout << **" solve car "** << getpid() << (direc == 0 ? **" >>"** : **" <<"**) << **"🚗 "** << endl;   **if** (direc == 0) *//东 +1* {  \*eastWait -= 1;  \*eastCount = \*eastCount + 1;  } **else if** (direc == 1) *//西 +1* {  \*westCount = \*westCount + 1;  \*westWait -= 1;  }  \*numCars = \*numCars + 1;  \*currentDire = direc;  lock->open\_lock(); }  **void** OneWay::Cross(**int** direc) {   lock->close\_lock();  sleep(1);  cout << getpid() << (direc == 0 ? **" -->>"** : **" <<--"**) << **" road cars:"**;  **for** (**int** i = 1; i <= \*numCars; ++i) {  cout << (direc == 0 ? **" -->>"** : **" <<--"**) << **" 🚗 "**;  }  cout << endl;  lock->open\_lock();  sleep(10);  }  **void** OneWay::Quit(**int** direc) {  lock->close\_lock();  \*numCars -= 1;  **if** (direc == 0) {  cout << getpid() << **" left "** << **"\n"**;  } **else if** (direc == 1) {  cout << getpid() << **" left "** << **"\n"**;  }  **if** (\*numCars == 0) {  **if** (direc == 0) {  **if** (\*westWait > 0) {  condition->Signal(1);  } **else if** (\*eastWait > 0) {  condition->Signal(0);  }  } **else if** (direc == 1) {  **if** (\*eastWait > 0) {  condition->Signal(0);  } **else if** (\*westWait > 0) {  condition->Signal(1);  }  }  }  lock->open\_lock(); }  OneWay::~OneWay() {  **delete** condition; }  #endif *//PROJECT6\_DP\_H*  运行结果： | | | |
| 结论分析与体会：  为了防止饥饿 需要在一侧车道通行车辆次数过多时暂停当前方向的车辆  等待车道清空 然后更改车道方向  当只有一侧有车辆到来时可以持续单方向通行 | | | |