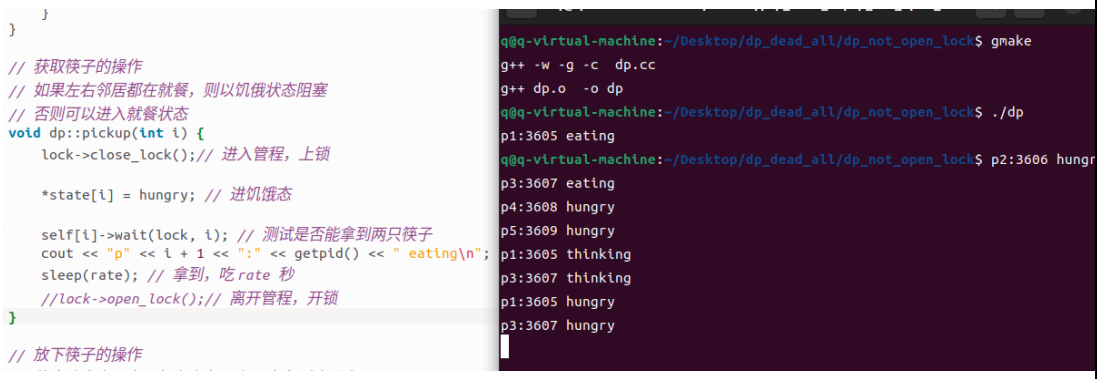
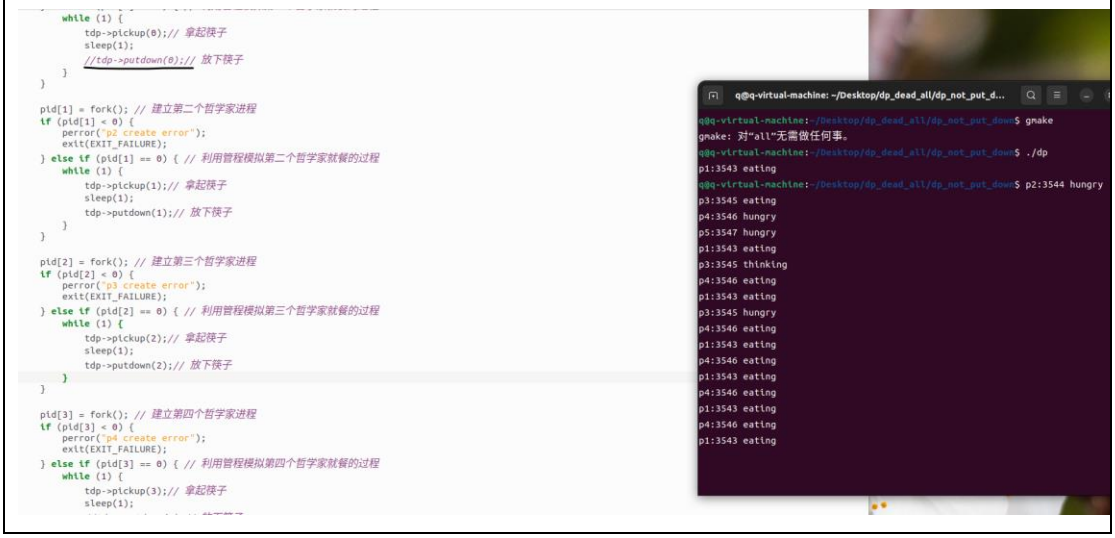
山东大学 计算机科学与技术 学院
操作系统 课程实验报告

学号：202200130223	姓名：李洋永	班级：4 班
实验题目：实验 6 死锁问题		
实验学时：2	实验日期：2024.11.13	
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饿问题的发生。		
实验环境： Ubuntu		
源程序清单：  		
编译及运行结果： 6）请修改以上 dp.cc 程序，制造出几种不同的的死锁现象和饥饿现象，记录并分析 各种死锁、饥饿现象和产生死锁、饥饿现象的原因。  执行完拿起筷子操作后不开锁会导致死锁，下一次拿起筷子时无法执行。在放下筷子函数处注释开锁操作也会导致死锁。这是因为上锁后不开锁会导致下一次函数无法执行。  		

在哲学家线程中注释掉放下筷子操作会导致死锁，这是因为一个哲学家吃完后会唤醒邻居，没有放下筷子操作当前哲学家一直处于 eating，其余哲学家饥饿。

6.4 独立实验

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。请构造一个管程来解决该问题。

南北两个车站各创建一进程，进程依次发出所有车辆，如果所有车辆均到达则唤醒另一车站。为避免一进程长时间等待为每次发车次数设置最大值。

```
rate = (argc > 1) ? atoi(argv[1]) : 2;//等待时间
max = (argc > 2) ? atoi(argv[2]) : 6;//控制每个车站每次发车次数，防止长时间等待

tdp = new dp(rate, max); //建立一个railway管程对象

for (int i = 0; i < 2; i++) {
    pid[i] = fork();
    if (pid[i] == 0) {
        while (1) {
            int k=rand() % max;//车次
            for(int j=0;j<=k;j++){//依次发出列车
               tdp->depart(i);
            }
            for(int j=0;j<=k;j++){//列车依次到达
               tdp->arrive(i);
            }
        }
        exit(0);
    }
}
```

发车函数:

```
void dp::depart(int i) {
    lock->close_lock(); //进入管程，上锁

    self[i]->Wait(lock, i); //测试是否发车
    if (!i) {//由南向北发车
        s_n[s_num]=*num;//记下列车号
        s_num++;
        cout << getpid() << ": "<<"train "<<*num<<" south->north"<<"\n";
    }
    else {//由北向南发车
        n_s[n_num]=*num;
        n_num++;
        cout << getpid() << ": "<<"train "<<*num<<" north->south"<<"\n";
    }
    *num = *num + 1;
    sleep(rate);//每次发车等待间隔
    lock->open_lock(); //离开管程，开锁
}
```

到站函数:

```
void dp::arrive(int i) {

    lock->close_lock();//进入管程，上锁
    if (!i) {
        sleep(rate);//列车依次到达
        cout << getpid() <<": "<<"train "<<s_n[s_index]<<" arrive"<<"\n";
        s_index++;
        if(s_index==s_num){//已发列车均到达，唤醒另一车站
          sleep(rate);
          *state[i] = waiting;
          self[i ^ 1]->Signal(i ^ 1);
          cout <<"south->north stop, north->south begin\n";
        }
    }
    else {
        sleep(rate);
        cout << getpid() <<": "<<"train "<<n_s[n_index]<<" arrive"<<"\n";
        n_index++;
        if(n_index==n_num){
          sleep(rate);
          *state[i] = waiting;
          self[i ^ 1]->Signal(i ^ 1);
          cout <<"north->south stop, south->north begin\n";
        }
    }

    lock->open_lock();//离开管程，开锁

}
```

如果已发列车均已到达则唤醒对方。

```
q@q-virtual-machine:~/Desktop/lab6/mine/train$ ./dp
3286: train 1 south->north
3286: train 2 south->north
3286: train 1 arrive
3286: train 2 arrive
south->north stop, north->south begin
3287: train 3 north->south
3287: train 4 north->south
3287: train 3 arrive
3287: train 4 arrive
north->south stop, south->north begin
3286: train 5 south->north
3286: train 6 south->north
3286: train 7 south->north
3286: train 8 south->north
3286: train 9 south->north
3286: train 5 arrive
3286: train 6 arrive
3286: train 7 arrive
3286: train 8 arrive
3286: train 9 arrive
south->north stop, north->south begin
3287: train 10 north->south
3287: train 11 north->south
3287: train 12 north->south
3287: train 13 north->south
3287: train 14 north->south
3287: train 10 arrive
3287: train 11 arrive
3287: train 12 arrive
3287: train 13 arrive
3287: train 14 arrive
```

问题及收获:

实例程序不会产生死锁，因为上锁与开锁都是一一对应的，进程死锁是因为上锁后未开锁，其它进程无法执行对应函数，进程饥饿是因为未唤醒其它进程，只有当前进程一直执行。管程通过封装共享资源、提供互斥和同步机制来避免死锁和饥饿现象。条件变量用于进程间的同步，特别是在等待某个条件时使用。进程可以在条件变量上等待，直到满足特定条件才被唤醒。信号量是一个计数器，用来控制对共享资源的访问。条件变量在管程中能够提供清晰、简洁的同步控制，避免死锁和竞争条件，因此管程中通常使用条件变量来实现进程同步。实例实验中的条件变量是如果左右都不处于 eating 且自身 hungry 则 eating，锁提供互斥，同一时间只执行一个。为解决单行道问题保证同一时间只有一个车站进程运行，所有列车到站后唤醒另一个车站进程。