

# 《操作系统原理》实验报告

姓名	张旭	学号	U201817106	专业班级	1805	时间	2020.04.25
----	----	----	------------	------	------	----	------------

## 一、实验目的

- (1) 理解线程/进程的通信机制和编程；
- (2) 理解线程/进程的死锁概念和如何解决死锁

## 二、实验内容

(2) (考虑信号通信机制) 在 Ubuntu 或 Fedora 环境创建父子 2 个进程 A, B。进程 A 不断获取用户从键盘输入的字符串或整数, 通过信号机制传给进程 B。如果输入的是字符串, 进程 B 将其打印出来; 如果输入的是整数, 进程 B 将其累加起来, 并输出该数和累加和。当累加和大于 100 时结束子进程, 子进程输出 “My work done!” 后结束, 然后父进程也结束。

(3) 在 windows 环境使用创建一对父子进程, 使用管道 (pipe) 的方式实现进程间的通信。父进程提供数据 (1-100, 递增), 子进程读出来并显示。

(5) 在 windows 环境下, 利用高级语言编程环境 (限定为 VS 环境或 VC 环境或 QT) 调用 CreateThread 函数哲学家就餐问题的 演示。要求: (1) 提供死锁的解法和非死锁的解法; (2) 有图形界面直观显示哲学家取筷子, 吃饭, 放筷子, 思考等状态。(3) 为增强结果的随机性, 各个状态之间的维持时间采用随机时间, 例如 100ms-500ms 之间。

## 三、实验过程

### (一) 实验步骤

#### (2) 信号通信

##### 1. 创建一块共享内存

由于不同进程空间之前是隔离的, 所以需要一块共享空间用来存放数据。

```

int main()
{
    pid_t child;
    key_t key = 0;
    int shmid = 0; // 共享内存

    char buffer[SIZE] = {'\0'};
    char str[100];

    shmid = shmget((key_t)1234, 1024, 0644 | IPC_CREAT);
    if(shmid == -1)
    {
        cout << "共享内存创建失败" << endl;
        return -1;
    }

    pshm = (char *)shmat(shmid, NULL, 0);
    if(pshm == (void *) - 1)
    {
        cout << "映射失败" << endl;
        shmctl(shmid, IPC_RMID, NULL);
        return -1;
    }
}

```

## 2. 创建子进程

```

// 创建进程
child = fork();
if(child == -1)
{
    cout << "创建进程失败!" << endl;
    shmctl(shmid, IPC_RMID, NULL);
    return -1;
}

```

## 3. 分别为父子进程注册信号处理函数

```

const int SIZE = 1024;
bool flag = true;
char * pshm = NULL;
void signHandlePa(int signum)
{
    exit(0);
}
int sum = 0;
void signHandleSon(int signum)
{
    sum += atoi(pshm);

    cout << "你输入的是: " << pshm << endl;
    cout << "当前sum=" << sum << endl;
    if(sum > 100)
    {
        cout << "My work done!" << endl;
    }
}

```

```

else if (child > 0)
{
    sleep(2);
    signal(SIGUSR1, signHandlePa);
    while (flag)
    {
        cin >> str;
        sprintf(pshm, "%s", str);
        kill(child, SIGUSR2);
    }
}

```

```

else
{
    pid_t pid = getppid();
    signal(SIGUSR2, signHandleSon);
    while(sum <= 100);
    kill(pid, SIGUSR1);
    exit(0);
}

return 0;
}

```

父进程写入共享内存，发送信号到子进程。

子进程接收信号，读取数据，处理，输出。发送信号到父进程

父进程接收到信号，继续发送

## 4. 运行程序输出结果

### (3) Windows 管道通信

#### 1. 编写接收进程

```
#include<iostream>
#include<Windows.h>

using namespace std;
DWORD dwRead = 0;
const int size = 500;
int buffer[size];
int main()
{
    HANDLE readHandle = GetStdHandle(STD_INPUT_HANDLE);

    if (!ReadFile(readHandle, buffer, 400, &dwRead, NULL))
    {
        cout << "读取失败" << endl;
    }
    else
    {
        for(int i = 1; i <= 100; i++)
        {
            cout << buffer[i-1] << endl;
        }
    }
    Sleep(100000000);
    return 0;
}
```

从管道中读取数据输出

#### 2. 编写发送进程

创建管道

```
HANDLE readHandle, writeHandle;
SECURITY_ATTRIBUTES sa;
sa.bInheritHandle = TRUE;
sa.lpSecurityDescriptor = NULL;
sa.nLength = sizeof(SECURITY_ATTRIBUTES);
if (!CreatePipe(&readHandle, &writeHandle, &sa, NULL))
{
    cout << "创建管道失败" << endl;
    return -1;
}
else
{
    cout << "管道创建成功" << endl;
}
```

创建进程（接收进程），注意 dwFlags 的设置。显示窗口

```
STARTUPINFO startInfo;
PROCESS_INFORMATION piInfo;
ZeroMemory(&piInfo, sizeof(piInfo));
ZeroMemory(&startInfo, sizeof(startInfo));
startInfo.cb = sizeof(startInfo);
startInfo.wShowWindow = SW_SHOW;
startInfo.dwFlags |= STARTF_USESTDHANDLES | STARTF_USESHOWWINDOW;
startInfo.hStdError = GetStdHandle(STD_ERROR_HANDLE);
startInfo.hStdInput = readHandle;

TCHAR exe[] = TEXT("son.exe");
if (!CreateProcess(NULL, exe, NULL, NULL, true, CREATE_NEW_CONSOLE, NULL, NULL, &startInfo, &piInfo))
{
    cout << "子进程创建失败" << endl;
}
else
{
    cout << "子进程创建成功" << endl;
    CloseHandle(piInfo.hThread);
    CloseHandle(piInfo.hProcess);
}
```

#### 3. 将数据写入管道

```

int buffer[105];
for (int i = 1; i <= 100; i++)
{
    buffer[i - 1] = i;
}

DWORD dwWrite = 0;
if (!WriteFile(writeHandle, buffer, sizeof(buffer), &dwWrite, NULL))
{
    cout << "写入失败" << endl;
}
else
{
    cout << "成功写入数据" << endl;
}

Sleep(300);

system("pause");
return 0;

```

#### 4. 运行程序，得出实验结果

##### (5) 哲学家问题

##### 1. 新建工程

哲学家问题由于限定环境。所以我选择了 QT。建了一个 QT 工程。

##### 2. 设定绘图频率

使用定时器，设定 FPS 值，每隔一个 FPS 刷新

```

//paintEvent,
void paintEvent(QPaintEvent *);    this->timerID = startTimer(50);
void timerEvent(QTimerEvent *);

void Widget::timerEvent(QTimerEvent *)
{
    repaint();
}

```

重写绘图函数，在绘图函数中对界面进行绘制。

##### 3. 编写其他的类

创建 People 和 ChopStick 类，people 继承 QThread 重写 run 函数，在 run 函数中实现 pv 等各种操作，同时修改筷子状态，修改哲学家状态。

设定类的成员，坐标，旋转的角度。

使用 translate 和 rotate，实现旋转角度贴图

```

pen.translate(this->x, this->y);
pen.rotate(this->rotate);

```

在 `people` 和 `chopstick` 中编写 `draw` 函数。实现贴图，然后在窗口类的 `paintEvent` 直接调用即可。

在窗口类中加入哲学家，筷子等成员，同时加入信号量。

```
int timerID;
People* peoples[MAXN];
ChopStick* chopsticks[MAXN];
QSemaphore* sem[5];
explicit Widget(QWidget *parent
```

#### 4. 不同解法

死锁解法：

```
void People::run()
{
    int timeOfThink = QRandomGenerator::global()->bounded(1000) + 1000;
    int timeOfRest = QRandomGenerator::global()->bounded(1000) + 1000;
    int timeOfEat = QRandomGenerator::global()->bounded(1000) + 1000;
    while(true)
    {
        // 休息
        this->status = REST;
        QThread::msleep(timeOfRest);
        // 思考
        this->status = THINK;
        QThread::msleep(timeOfThink);
        // 吃饭
        // 拿左侧筷子
        this->sem[this->id]->acquire();
        this->status = LEFT;
        this->chopsticks[this->id]->setPos(this->id, LEFT); // 修改坐标
        // 拿右侧筷子
        QThread::msleep(1000);
        this->sem[(this->id + 4) % MAXN]->acquire();
        this->status = FULL;
        this->chopsticks[(this->id + 4) % MAXN]->setPos(this->id, RIGHT); // 修改坐标
        QThread::msleep(timeOfEat);

        // 释放筷子
        this->sem[this->id]->release();
        this->sem[(this->id + 4) % MAXN]->release();
        this->chopsticks[this->id]->initPos(this->id); // 恢复左侧坐标
        this->chopsticks[(this->id + 4) % MAXN]->initPos((this->id + 4) % MAXN); // 恢复右侧坐标
    }
}
```

注意需要修改筷子坐标，恢复筷子坐标。使用 QT 中的信号量。

使用 `acquire` 和 `release`（pv 操作）

破坏环路法：最多有四人同时拿起左侧筷子

增加一个信号量，限制左侧筷子

```

void People::run()
{
    int timeOfThink = QRandomGenerator::global()->bounded(1000) + 1000;
    int timeOfRest = QRandomGenerator::global()->bounded(1000) + 1000;
    int timeOfEat = QRandomGenerator::global()->bounded(1000) + 1000;
    while(true)
    {
        // 休息
        this->status = REST;
        QThread::msleep(timeOfRest);
        // 思考
        this->status = THINK;
        QThread::msleep(timeOfThink);
        // 吃饭
        // 拿左侧筷子
        this->full->acquire();
        this->sem[this->id]->acquire();
        this->status = LEFT;
        this->chopsticks[this->id]->setPos(this->id, LEFT); // 修改坐标
        // 拿右侧筷子
        //QThread::msleep(1000);
        this->sem[(this->id + 4) % MAXN]->acquire();
        this->status = FULL;
        this->chopsticks[(this->id + 4) % MAXN]->setPos(this->id, RIGHT); // 修改坐标
        QThread::msleep(timeOfEat);

        // 释放筷子
        this->sem[this->id]->release();
        this->sem[(this->id + 4) % MAXN]->release();
        this->chopsticks[this->id]->initPos(this->id); // 恢复左侧坐标
        this->chopsticks[(this->id + 4) % MAXN]->initPos((this->id + 4) % MAXN); // 恢复右侧坐标
        this->full->release();
    }
}

```

有序资源分配法：左右两侧筷子，先拿下标大的。

0 号哲学家先拿右侧 4 号筷子，再拿 0 号筷子。其他的哲学家刚好左侧下标较大。

```

void People::run()
{
    int timeOfThink = QRandomGenerator::global()->bounded(1000) + 1000;
    int timeOfRest = QRandomGenerator::global()->bounded(1000) + 1000;
    int timeOfEat = QRandomGenerator::global()->bounded(1000) + 1000;
    while(true)
    {
        // 休息
        this->status = REST;
        QThread::msleep(timeOfRest);
        // 思考
        this->status = THINK;
        QThread::msleep(timeOfThink);
        if(this->id == 0)
        {
            // 拿右侧筷子
            QThread::msleep(1000);
            this->sem[(this->id + 4) % MAXN]->acquire();
            this->status = FULL;
            this->chopsticks[(this->id + 4) % MAXN]->setPos(this->id, RIGHT); // 修改坐标
            // 吃饭
            // 拿左侧筷子
            this->sem[this->id]->acquire();
            this->status = LEFT;
            this->chopsticks[this->id]->setPos(this->id, LEFT); // 修改坐标
        }
        else
        {
            // 吃饭
            // 拿左侧筷子
            this->sem[this->id]->acquire();
            this->status = LEFT;
            this->chopsticks[this->id]->setPos(this->id, LEFT); // 修改坐标
            // 拿右侧筷子
            QThread::msleep(1000);
            this->sem[(this->id + 4) % MAXN]->acquire();
            this->status = FULL;
            this->chopsticks[(this->id + 4) % MAXN]->setPos(this->id, RIGHT); // 修改坐标
        }
        QThread::msleep(timeOfEat);
        // 释放筷子
        this->sem[this->id]->release();
        this->sem[(this->id + 4) % MAXN]->release();
        this->chopsticks[this->id]->initPos(this->id); // 恢复左侧坐标
    }
}

```

注意 id == 0

时，拿筷子顺序

5. 运行程序得到结果。

6. 切换 release 版本，构建使用 QT 自带的发布工具，打包发布。

由于发布后文件较大，不包含在上交的文件里面了

## （二）解决错误和优化

### （2）信号通信

1. 【操作过程错误】之前没有创建共享内存，使用的是 `sigqueue` 直接传递一个 `void*` 指针。

```
int sigqueue(pid_t pid, int sig, const union sigval value);
```

这个 `value` 联合体中有一个 `int` 成员，有一个 `void*` 指针。通过 `int` 进行父子进程间传递数据（值传递可以），但是只能传递数字。想要传递输入的字符串时我之前用的 `void*` 指针，但是发现传过去的内容和输入的不同。查看地址后发现，传到子进程后地址发生了改变。原因是

从结果可以看到指针可以传递过去，但是指针所指的值变化了，为什么呢？答案很简单，在进程那部分，我们一直在强调一点：父子进程的地址空间是相互独立的！所以在父子进程间传递指针是没有意义的。

后来创建一块共享内存解决了隔离问题。

### （3）Windows 管道通信

1. 【操作过程错误】

现象：创建新的进程后，子进程不显示窗口，无法看到是否接收了数据

解决：查询资料，修改了 `dwflags`，窗口成功显示。

### （5）哲学家问题

1. 【语法错误】

遇到很多语法错误，通过查阅相关函数用法，官方 API 文档解决

2. 【现象不明显】

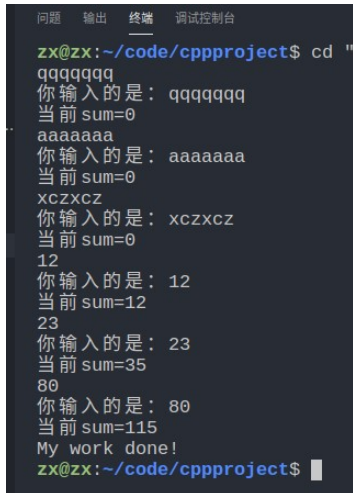
现象：死锁解法长时间正常运行，没有发生死锁



解决：在拿到左侧筷子后先 `QThread::msleep` 一段时间等待其他的线程。结果很明显，运行后很快出现死锁现象。

## 四、实验结果

### (2) 信号通信



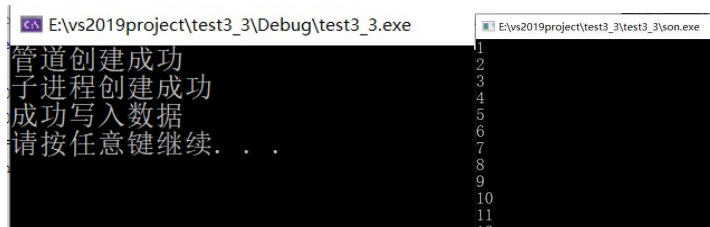
```
zx@zx:~/code/cppproject$ cd "
qqqqqqq
你输入的是: qqqqqqq
当前 sum=0
aaaaaaa
你输入的是: aaaaaaa
当前 sum=0
xczxcz
你输入的是: xczxcz
当前 sum=0
12
你输入的是: 12
当前 sum=12
23
你输入的是: 23
当前 sum=35
80
你输入的是: 80
当前 sum=115
My work done!
zx@zx:~/code/cppproject$
```

当前 `sum>100` 时，程序退出，运行正常

### (3) Windows 管道通信

父进程中

子进程中



```
E:\vs2019project\test3_3\Debug\test3_3.exe
管道创建成功
子进程创建成功
成功写入数据
请按任意键继续. . .

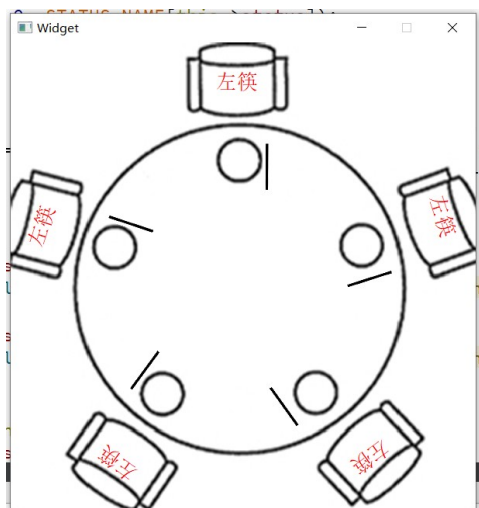
E:\vs2019project\test3_3\test3_3son.exe
1
2
3
4
5
6
7
8
9
10
11
12
```

子进程成功读取数据

### (5) 哲学家问题

死锁解法：

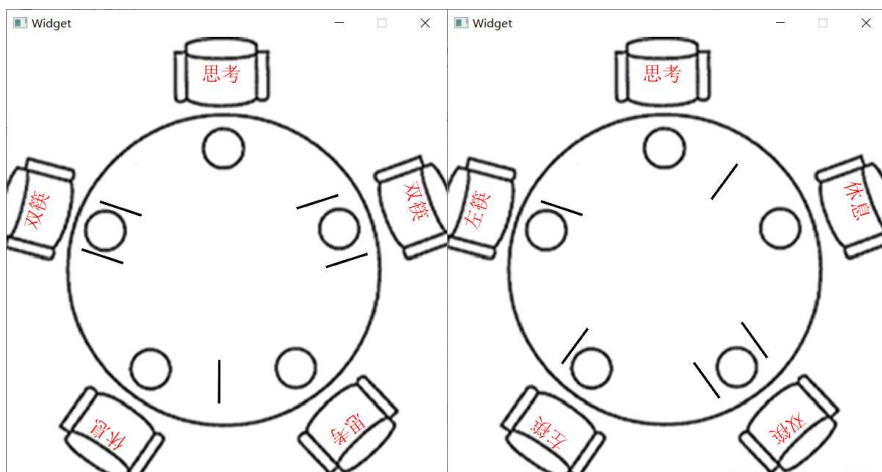




每人一个左侧筷子，出现死锁。

破坏环路解法：

有序资源法：



运行正常

## 五、体会

学习了信号通信机制，同时对如何处理进程间的通信有了更深的理解。对管道通信也有了更深的理解，对管道读写操作，进程创建等也有了更多的了解。在哲学家问题中，巩固了 QT 窗体编程，同时对如何使用信号量处理进程间关系也有了更深刻的认识。进程间关系的处理时非常重要的，之后我会继续学习相关知识。