

《操作系统》

上机报告

班 级： 1403012

学 号： 14030120020

姓 名： 杨 帆

实验 1:

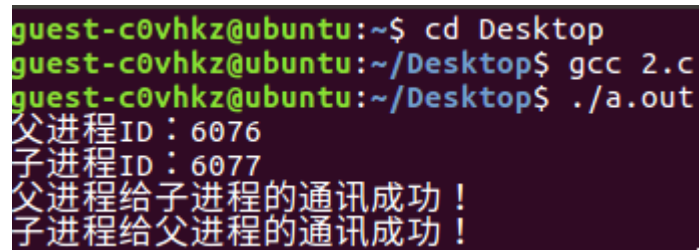
实验题目	进程的建立
实验目的	创建进程及子进程 在父子进程间实现进程通信
实验软硬件环境	Linux 、 Windows98、 Windows2000
实验内容	创建进程并显示标识等进程控制块的属性信息； 显示父子进程的通信信息和相应的应答信息。 (进程间通信机制任选)
实验步骤	创建进程； 显示进程状态信息； 实现父子进程通信；
考核指标	显示创建的进程及控制块参数； 显示进程间关系参数

源程序:

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main()
{
    int fd1[2],fd2[2],cld_pid,pid,status;
    char buf[200], len;
    if (pipe(fd1) == -1) { // 创建管道 1
        printf("creat pipe1 error\n");
        exit(1);
    }
    if (pipe(fd2) == -1) { // 创建管道 2
        printf("creat pipe2 error\n");
        exit(1);
    }
    if ((cld_pid=fork()) == 0) { //子进程
        cld_pid=getpid();
        printf("子进程 ID: %d\n",cld_pid);
        close(fd1[1]); // 子进程关闭管道 1 的写入端
        close(fd2[0]); // 子进程关闭管道 2 的读出端
```

```
        len = read(fd1[0], buf, sizeof(buf)); //子进程读管道 1
        printf("%s", buf);
        strcpy(buf, "子进程给父进程的通讯成功! \n");
        write(fd2[1], buf, strlen(buf)); //子进程写管道 2
        exit(0);
    }
    else { //父进程
        pid=getpid();
        printf("父进程 ID: %d\n", pid);
        close(fd1[0]); // 父进程关闭管道 1 的读出端
        close(fd2[1]); // 父进程关闭管道 2 的写入端
        strcpy(buf, "父进程给子进程的通讯成功! \n");
        write(fd1[1], buf, strlen(buf)); //父进程写管道 1
        len = read(fd2[0], buf, sizeof(buf)); //父进程读管道 2
        printf("%s", buf);
        exit(0);
    }
}
```

运行结果



```
guest-c0vhkz@ubuntu:~$ cd Desktop
guest-c0vhkz@ubuntu:~/Desktop$ gcc 2.c
guest-c0vhkz@ubuntu:~/Desktop$ ./a.out
父进程ID：6076
子进程ID：6077
父进程给子进程的通讯成功！
子进程给父进程的通讯成功！
```

实验 2:

实验题目	进程间的同步
实验目的	理解进程同步和互斥模型及其应用
实验软硬件环境	Linux 、Windows98、Windows2000
实验内容	利用通信 API 实现进程之间的同步： 建立司机和售票员进程； 并实现他们间的同步运行。
实验步骤	创建进程； 实现同步操作或函数； 实现公共汽车司机和售票员开关车门及行车运行过程的同步模型； 显示同步运行的结果。
考核指标	显示司机和售票员进程的同步运行轨迹。

司机进程代码：

```
#include <sys/types.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <semaphore.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

char SEM_NAME1[] = "s1";
char SEM_NAME2[] = "s2";

int main()
{
    int i;
    sem_t *s1;          //信号量的数据类型为结构 sem_t
    sem_t *s2;

    s1 = sem_open(SEM_NAME1, O_CREAT, 0644, 0);
    //创建并初始化有名信号
    //sem_t *sem_open(const char *name, int oflag, mode_t mode, unsigned int
```

```

value);
//mode 权限位  value 信号灯初始值

if(s1 == SEM_FAILED) //创建失败
{
    perror("unable to create semaphore");
    extern int sem_unlink (const char *__s1) __THROW;//从系统中删除信号
    exit(-1);
}

s2 = sem_open(SEM_NAME2, 0_CREAT, 0644, 0); //创建并初始化有名信号
if(s2 == SEM_FAILED)
{
    perror("unable to create semaphore");
    extern int sem_unlink (const char *__s2) __THROW;
    exit(-1);
}

for (i=0; i<=2; i++) {
    printf("[driver] reach station, stop car\n");
    sem_post(s2); //V2 操作
    printf("[driver] waiting closing door\n");
    sem_wait(s1); // P1 操作
    printf("[driver] leaving station\n");
    printf("car is running\n");
}

sleep(2);
sem_close(s1); //关闭有名信号
sem_close(s2);
sem_unlink(SEM_NAME1); //从系统中删除信号
sem_unlink(SEM_NAME2);
_exit(0);
}

```

售票员进程代码:

```

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

```

```

#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <semaphore.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

char SEM_NAME1[] = "s1";
char SEM_NAME2[] = "s2";

int main()
{
    int i;
    sem_t *s1;
    sem_t *s2;

    s1 = sem_open(SEM_NAME1, 0, 0644, 0);
    if(s1 == SEM_FAILED)
    {
        perror("unable to create semaphore");
        sem_close(s1);
        exit(-1);
    }

    s2 = sem_open(SEM_NAME2, 0, 0644, 0);
    if(s2 == SEM_FAILED)
    {
        perror("unable to create semaphore");
        sem_close(s2);
        exit(-1);
    }

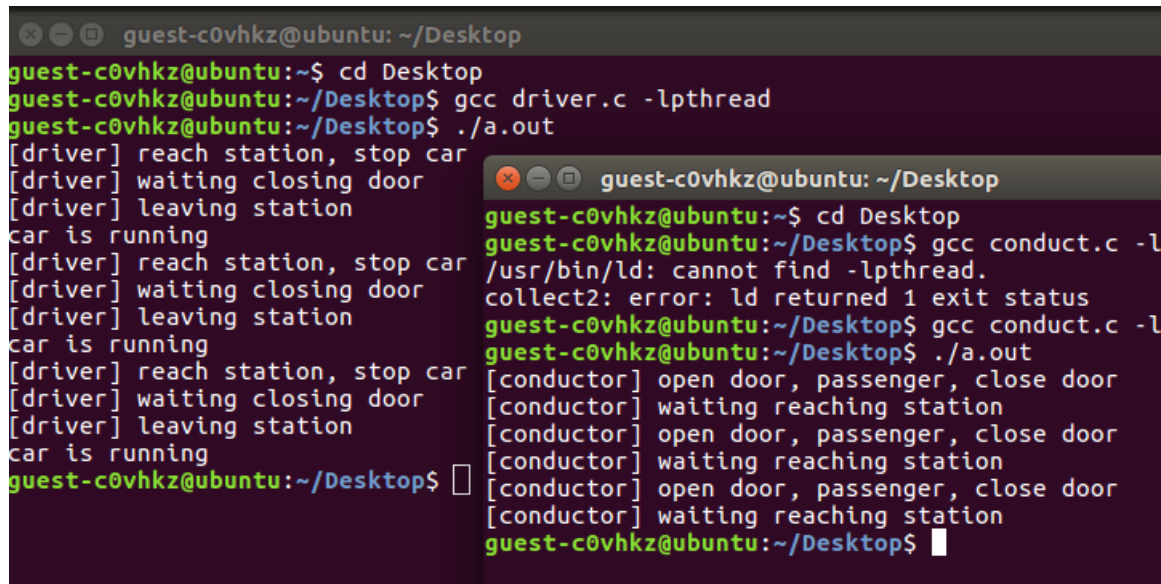
    for (i=0; i<=2; i++) {
        sem_wait(s2); // P 2 操作
        printf("[conductor] open door, passenger in , close door\n");
        sem_post(s1); // V 1 操作
        printf("[conductor] waiting reaching station\n");
    }

    sem_close(s1);

```

```
sem_close(s2);  
_exit(0);  
}
```

运行结果:



```
guest-c0vhkz@ubuntu: ~/Desktop  
guest-c0vhkz@ubuntu:~$ cd Desktop  
guest-c0vhkz@ubuntu:~/Desktop$ gcc driver.c -lpthread  
guest-c0vhkz@ubuntu:~/Desktop$ ./a.out  
[driver] reach station, stop car  
[driver] waiting closing door  
[driver] leaving station  
car is running  
[driver] reach station, stop car  
[driver] waiting closing door  
[driver] leaving station  
car is running  
[driver] reach station, stop car  
[driver] waiting closing door  
[driver] leaving station  
car is running  
guest-c0vhkz@ubuntu:~/Desktop$  
guest-c0vhkz@ubuntu:~/Desktop$ gcc conduct.c -lpthread  
/usr/bin/ld: cannot find -lpthread.  
collect2: error: ld returned 1 exit status  
guest-c0vhkz@ubuntu:~/Desktop$ gcc conduct.c -lpthread  
guest-c0vhkz@ubuntu:~/Desktop$ ./a.out  
[conductor] open door, passenger, close door  
[conductor] waiting reaching station  
[conductor] open door, passenger, close door  
[conductor] waiting reaching station  
[conductor] open door, passenger, close door  
[conductor] waiting reaching station  
guest-c0vhkz@ubuntu:~/Desktop$
```

结果分析:

利用 PV 操作实现司机进程和售票员进程之间的同步。设司机进程的信号量为 S1, 售票员进程信号量为 S2。当车到站时, 停车, 进行操作 V(S2), P(S1)。此时司机进程被挂起, 售票员进程开始进行。售票员进程先进行 P(S2) 操作, 若之前未进行 V(S2) 则被挂起, 否则进程继续执行, 打开车门。待人上完后关车门, 进行 V(S1) 操作, 唤醒被挂起的司机进程。汽车再次离站开出。

实验 3:

实验题目

线程共享进程数据

实验目的

了解线程与进程之间的数据共享关系。

创建一个线程，在线程中更改进程中的数据。

实验软件环境

VC++6.0 或者Linux 操作系统

实验内容

在进程中定义全局共享数据，在线程中直接引用该数据进行更改并输出该数据。

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <pthread.h>
void *thread_function(void *arg); //创建线程
int count=10;

int main(int argc, char const *argv[]) {
    int res;
    pthread_t  a_thread; //用于声明线程 ID
    void *thread_result;//
    printf("Process is running...\n");
    printf("Process count=%d\n",count);

    //创建线程 pthread_create, 0 则创建成功
    res=pthread_create(&a_thread,NULL, thread_function, (void*)&count);
    if (res!=0) {
        perror("Thread creation failed");
```



```

        exit(EXIT_FAILURE);
    }

    res=pthread_join(a_thread,&thread_result);//. 等待线程结束,0 成功
        //将返回的结果放到 thread_result 中

    if (res!=0) {
        perror("Thread join failed");
        exit(EXIT_FAILURE);
    }

    printf("return count=%d\n", (*(int*)thread_result)); //打印出从线程传来的信息
    printf("Thread finished\n");
    printf("Process finished\n");
}

void *thread_function(void *arg) { // 创建线程
    printf("New thread running\n");
    int sum=0;
    pthread_t pt;    //用于声明线程 ID
    pt=pthread_self();    //获取线程标示符
    printf("Thread %x ran\n", (int)pt);

    for (count=1;count<=6;count++) {
        printf("Thread count=%d\n", count);
        sum+=count;
    }
    printf("sum=%d\n", sum);
    printf("thread sleep 5s\n");//
    sleep(5);
    pthread_exit(arg); // 终止线程 pthread_exit
}

```

实验结果

```
guest-c0vhkz@ubuntu:~/Desktop$ gcc -pthread xiancheng.c -o xiancheng -lpthread
guest-c0vhkz@ubuntu:~/Desktop$ ./xiancheng
Process is running...
Process count=10
New thread running
Thread b75a7b40 ran
Thread count=1
Thread count=2
Thread count=3
Thread count=4
Thread count=5
Thread count=6
sum=21
thread sleep 5s
return count=7
Thread finished
Process finished
```

实验心得

通过这次实验，我学到了如何应用 C 语言实现进程的创建、进程之间的同步与通信，进程与线程之间的数据共享等。并且掌握了如何在 Linux 系统下如何编译及运行该程序和使用虚拟机 Ubuntu。该门上机实验很好的将实践与我们上课所学内容结合起来，例如本来在书本上的 PV 操作，现在也可以以代码的形式展现出来。虽然在实验也遇到了不少问题，但通过查阅资料和咨询老师，都得到了解决。总之，本次上机实验让我对操作系统这门课有了更加深刻的理解，以后仍需继续深入挖掘。