课程编号：B080000070

《操作系统》实验报告

姓　　名	薛　　旗	学　　号	2 0 1 5 5 3 6 2
班　　级	软信-1503	指　导　教　师	王学毅
实　验　名　称	《操作系统》作业		
开　设　学　期	2016-2017 第二学期		
开　设　时　间	第 11 周——第 18 周		
报　告　日　期	2017 年 7 月 3 日		
评　定　成　绩		评　定　人	
		评　定　日　期	2017 年 7 月 5 日

东北大学软件学院

资源分配

一、实验目的

多个进程动态地共享系统的资源可能会产生死锁现象。死锁的产生，必须同时满足四个条件，第一个是互斥条件，即一个资源每次只能由一个进程占用；第二个为等待条件，即一个进程请求资源不能满足时，它必须等待，但它仍继续保持已得到的所有其它资源；第三个是非出让条件，任何一个进程不能抢占另一个进程已经获得且未释放的资源；第四个为循环等待条件，系统中存在若干个循环等待的进程，即其中每一个进程分别等待它前一个进程所持有的资源。防止死锁的机构只须确保上述四个条件之一不出现，则系统就不会发生死锁。

在实验中假定系统中任一资源在每一时刻只能则由一个进程使用，任何进程不能抢占它进程正在使用的资源，当进程得不到资源时必须等待。因此只要资源分配策略能保证进程不出现循环等待，则系统就不会发生死锁。

本实验要求学生编写和调试一个系统动态分配资源的简单模拟程序，观察死锁产生的条件，并采用适当的算法，有效地防止和避免死锁的发生。

二．预习内容

死锁的四个条件（同时满足，引起死锁）

1，互斥

2，占有并等待

3，非抢占

4，循环等待

死锁预防：只要上述四个条件至少有一个不成立，就能预防死锁发生。

死锁避免：2 个死锁避免算法:资源分配图算法

三．实验题目

第一题：用银行家算法实现资源分配。要求：

(1) 假定系统中有五个进程{P0, P1, P2, P3, P4}和三类资源{A, B, C}，各种资源的数量分别为 10、5、7，在 T0 时刻的资源分配情况如图所示。

进程 \ 资源情况	Max A B C	Allocation A B C	Need A B C	Available A B C
P_0	7 5 3	0 1 0	7 4 3	3 3 2 (2 3 0)
P_1	3 2 2	2 0 0 (3 0 2)	1 2 2 (0 2 0)	
P_2	9 0 2	3 0 2	6 0 0	
P_3	2 2 2	2 1 1	0 1 1	
P_4	4 3 3	0 0 2	4 3 1	

利用银行家算法设计系统，进程可动态地申请资源和释放资源，系统按各进程的申请动态地分配资源。

(2)设计用银行家算法和随机分配算法，实现资源分配的两个资源分配程序，应具有显示或打印各进程依次要求申请的资源数以及依次分配资源的情况。

(3)确定一组各进程依次申请资源数的序列，在相同的情况下分别

运行上述两种资源分配程序，观察运行结果。

第二题：用按序分配策略实现资源分配。要求：

(1)设计一个 3 个进程共享 10 个资源的系统，进程可动态地申请资源和释放资源，系统按各进程的申请动态地分配资源。

(2)设计用按序分配算法实现资源分配的资源分配程序，应具有显示或打印各进程依次要求申请的资源号以及依次分配资源地情况。

(3)确定两组各进程依次要求申请的资源号，要求其中的一组中各进程按序地申请资源，另一组中各进程申请资源不受序号限制，分别运行上述设计的资源分配程序，观察运行结果。

四：源代码及运行结果

第一题

源代码：

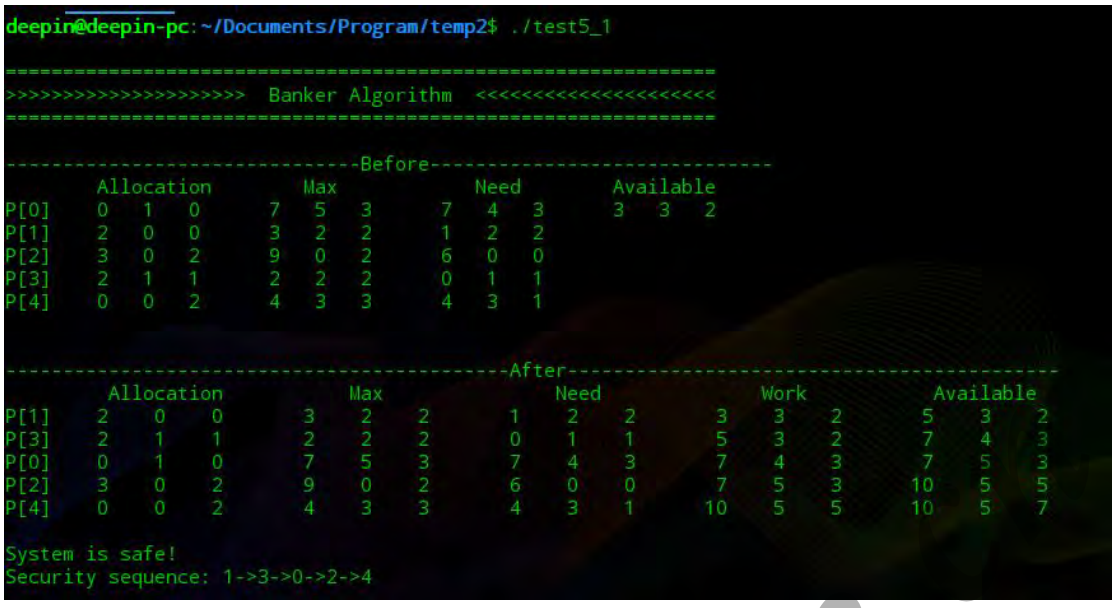
```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

int Available[3] = { 3, 3, 2 };
int Max[5][3] = {
    7, 5, 3,
    3, 2, 2,
    9, 0, 2,
    2, 2, 2,
    4, 3, 3
};
int Allocation[5][3] = {
    0, 1, 0,
    2, 0, 0,
    3, 0, 2,
```

```
    2, 1, 1,
    0, 0, 2
};
int Need[5][3] = {
    7, 4, 3,
    1, 2, 2,
    6, 0, 0,
    0, 1, 1,
    4, 3, 1
};
int Request[5][3];
int flag[5];
int list[5];

void display();
void safe();
void banker();
void output();

void display() {
    int i, j;
    printf("\n------------------------------Before-----------------------------\n");
    printf("        Allocation        Max        Need        Available\n");
    for (i = 0; i < 5; i++) {
        printf("P[%d]    ", i);
        for (j = 0; j < 3; j++)
            printf("%d   ", Allocation[i][j]);
        printf("   ");
        for (j = 0; j < 3; j++)
            printf("%d   ", Max[i][j]);
        printf("   ");
        for (j = 0; j < 3; j++)
            printf("%d   ", Need[i][j]);
        printf("   ");
        if (i == 0) {
            for (j = 0; j < 3; j++)
                printf("%d   ", Available[j]);
        }
        printf("\n");
    }

}
void safe() {
    printf("\n\n-----------------------------------------After--------------------
```

```
----------\n");
    printf("        Allocation          Max             Need            Work
Available\n");
    int i, j, k;
    int m, n;
    int l = 0;
    int work[3];
    for (i = 0; i < 3; i++)
        work[i] = Available[i];
    for (i = 0; i < 5; i++)
        flag[i] = 0;
    for (i = 0; i < 5; i++) {
        if (flag[i] == 1)
            continue;
        else {
            for (j = 0; j < 3; j++) {
                if (Need[i][j] > work[j])
                    break;
            }
            if (j == 3) {
                flag[i] = 1;
                printf("P[%d]   ", i);
                for (m = 0; m < 3; m++)
                    printf("%2d   ", Allocation[i][m]);
                printf("   ");
                for (m = 0; m < 3; m++)
                    printf("%2d   ", Max[i][m]);
                printf("   ");
                for (m = 0; m < 3; m++)
                    printf("%2d   ", Need[i][m]);
                printf("   ");
                for (m = 0; m < 3; m++)
                    printf("%2d   ", work[m]);
                printf("   ");
                for (m = 0; m < 3; m++) {
                    work[m] = work[m] + Allocation[i][m];
                    printf("%2d   ", work[m]);
                }
                printf("   ");
                list[l++] = i;
                i = -1;
            }
            else
                continue;
```

```
        }
        if (l == 5) {
            printf("\n\nSystem is safe!\n");
            printf("Security sequence: ");
            for (i = 0; i < l; i++) {
                printf("%d", list[i]);
                if (i != l - 1)
                    printf("->");
            }
            printf("\n\n");
            return;

        }

        printf("\n");
    }
    printf("System is not safe!\n");
    printf("\n\n");
    return;
}

void main() {
    printf("\n");
    printf("========================================================\n");
    printf(">>>>>>>>>>>>>>>>>>>> Banker Algorithm <<<<<<<<<<<<<<<<<<<<\n");
    printf("========================================================\n");
    display();
    safe();
}
```

实验结果：



第二题

源代码：

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define N 10
int ready = 0;          //进程为就绪态
int end = 1;            //进程为完成态
int waiting = 2;        //进程为等待态

struct  PCB {
    int number;             //进程号
    int state;              //状态
    int need;               //资源最大需求量
    int allocation;         //资源已占有量
    int apply;              //当前资源申请号
    int lapply;             //定义进程上次资源申请号
}P[3];

struct  RCB {
```

```
    int R_number;           //资源号
    int R_state;            //该资源申请与否
}R[10];

struct  PCB  *q, *h;
struct  PCB  *pr; //用于打印
struct  RCB  *r, *k, *s;
int NUM[300];
int t = 0;


void initial();
void order();
void dis_info();
void display();

void initial() {
    int t;
    int m = 0, n = 0, a = 0;
    printf("Process_ID        Resource_Max\n");
    for (q = P; q < P + 3; q++) {
        a++;
        printf("    %d                  ", a);
        scanf("%d", &q->need);
        while (q->need < 1 || q->need > 10) {
            setbuf(stdin, NULL);
            printf("Out of range! Please input again!\n");
            scanf("%d", &q->need);
        }
    }

    for (q = P; q < P + 3; q++) {
        n++;
        q->number = n;
        q->state = ready;
        q->allocation = 0;
        q->apply = 0;
        q->lapply = 0;
    }
    for (r = R; r < R + 10; r++) {
        m++;
        r->R_number = m;
        r->R_state = 0;
    }
```

```
    for (t = 0; t < 300; t++)
        NUM[t] = 0;
    t = 0;
}

void order() {
    //printf("\n**************************按序分配算法
**************************************");
    for (q = P - 1; ;) {
        q++;
        if (q->state == ready) {
            if (q->apply == 0) {
                printf("\nInput the process %d applys for(%d -- %d): ", q->number,
q->lapply + 1, 11 - (q->need - q->allocation));
                scanf("%d", &q->apply);
                NUM[t] = q->apply;
                t++;
                while (q->apply > 10 || q->apply < 1) {
                    setbuf(stdin, NULL);
                    printf("Out of range! Please input again!\n");
                    scanf("%d", &q->need);
                }
                while ((q->need - q->allocation) >(11 - q->apply)) {
                    t--;
                    setbuf(stdin, NULL);
                    printf("May be a deadlock! Please input again!\n");
                    scanf("%d", &q->need);
                }
            }

            else {
                t++;
            }

            r = R + (q->apply) - 1; //指向当前申请的资源
            if (q->apply >= q->lapply)//此时为按序申请
            {
                if (r->R_state == 0)//当前申请的资源没被占用
                {
                    q->lapply = q->apply;
                    q->apply = 0;
                    r->R_state = q->number;
                    q->allocation++;
                    printf("Assine resource %d to process %d\n", r->R_number,
```

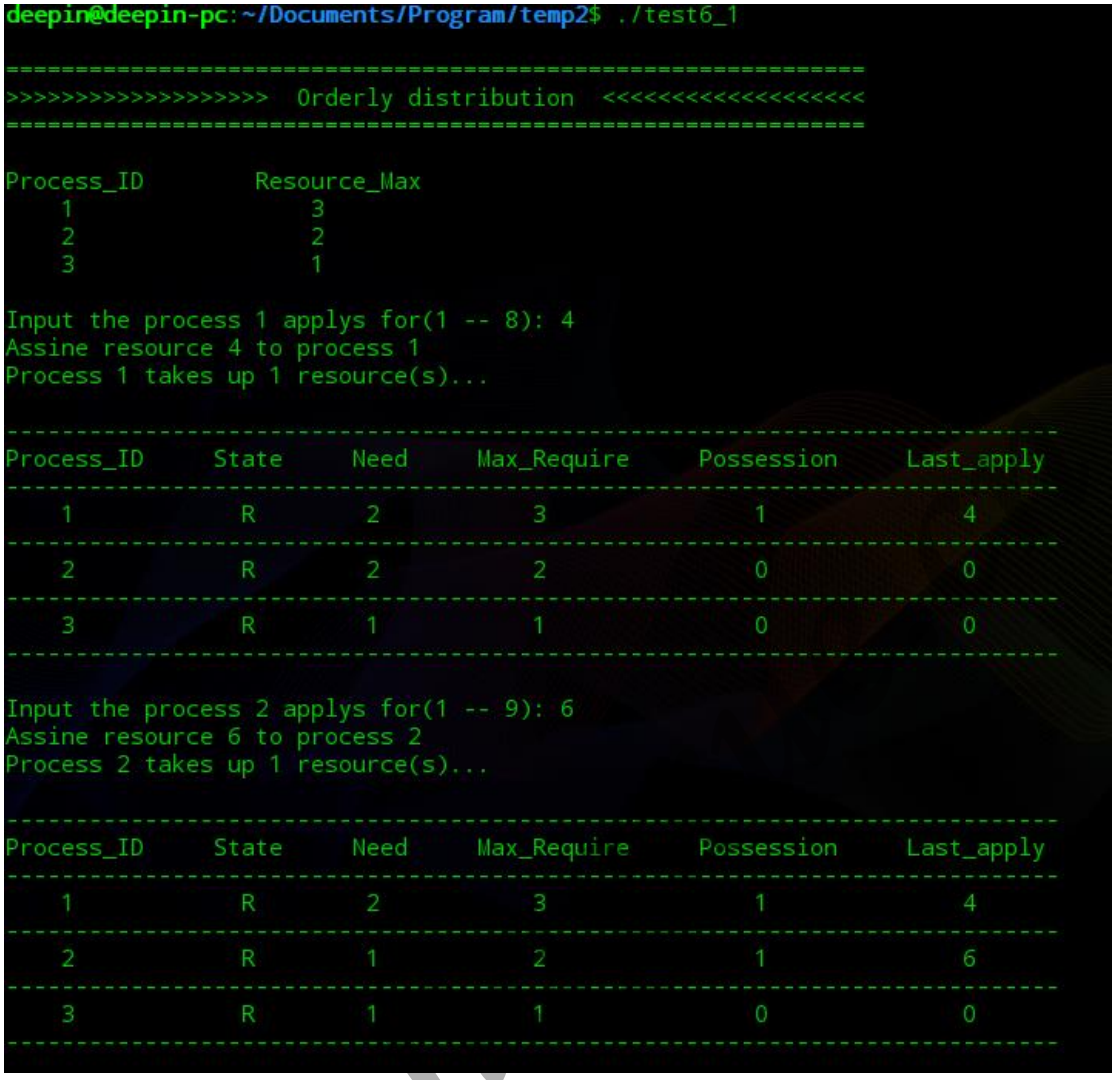
```
q->number);
                    printf("Process %d takes up %d resource(s)...\n\n", q->number,
q->allocation);
                    if (q->allocation == q->need)
                        q->state = end;
                    display();
                    if (q->allocation == q->need)//当前进程的占有量等于资源最大需求量
                    {
                        q->state = end;
                        for (k = R; k < R + 10; k++) //进程完成,释放资源
                        {
                            if (k->R_state == q->number)
                                k->R_state = 0;
                        }
                        for (h = P; h < P + 3; h++) //如果等待的资源被释放，将等待态
的进程改为就绪态
                        {
                            s = R + h->apply - 1;
                            if (s->R_state == 0)
                            {
                                if (h->state == waiting)
                                    h->state = ready;
                            }
                        }
                        printf("Process %d is successfuly assigned!\n", q->number);
                        h = P;
                        if (h->state == end && (h + 1)->state == end && (h + 2)->state
== end) {
                            printf("All of the processes are successfully
assigned!\n\n");
                            dis_info();
                            return;
                        }
                        else {
                            if (q == P + 2)
                                q = P - 1;
                            continue;
                        }
                    }
                    else {
                        if (q == P + 2)
                            q = P - 1;
                        continue;
                    }
```

```
                }
                else {
                    if (r->R_state == q->number) {
                        printf("Resource: %d has been assigned!\n", r->R_number);
                        q->apply = 0;
                    }
                    else {
                        printf("Resource: %d has been using by other process!\n",
r->R_number);
                        printf("Process: %d enters the wait state...\n");
                        q->state = waiting;
                        display();
                    }
                    if (q == P + 2)
                        q = P - 1;
                    continue;
                }
            }
            else //申请未按序
            {
                q->apply = 0;
                if (q == P + 2)
                    q = P - 1;
                continue;
            }
        }
        else//该进程不是就绪进程
        {
            t++;
            if (q == P + 2)
                q = P - 1;
            continue;//结束本次循环
        }
    }
}
void dis_info() {
    int ta, a;
    printf("----------------------------Apply
Information-----------------------------\n\n");
    printf("Process_ID          Process_1          Process_2          Process_3\n");
    for (ta = 0; ta < t - 1; t++) {
        printf("             ");
        for (a = 0; a < 3; a++) {
            printf("             ");
```

```
            if (NUM[ta] != 0) {
                printf("%d      ", NUM[ta]);
            }
            else
                printf("       ");
            ta++;

        }
        printf("\n");
    }
    printf("------------------------------------------------------------------------
-\n");
}

void display() {
    printf("------------------------------------------------------------------------
-\n");
    printf("Process_ID     State     Need     Max_Require     Possession
Last_apply\n");
    printf("------------------------------------------------------------------------
-\n");

    for (pr = P; pr < P + 3; pr++)
    {
        printf("    %d           ", pr->number);
        if (pr->state == 0)
            printf("R");
        if (pr->state == 1)
            printf("F");
        if (pr->state == 2)
            printf("W");
        printf("       %d          %d              %d             %d\n", pr->need -
pr->allocation, pr->need, pr->allocation, pr->lapply);


    printf("------------------------------------------------------------------------
-\n");
    }
}
void main() {
    printf("\n");
    printf("============================================================\n");
    printf(">>>>>>>>>>>>>>>>>>>  Orderly distribution  <<<<<<<<<<<<<<<<<<<\n");
    printf("============================================================\n");
```

```
    printf("\n");
    initial();
    order();
}
```

实验结果(见下页)：


deepin@deepin-pc:~/Documents/Program/temp2$ ./test6_1
>>>>>>>>>>>>>>>>>> Orderly distribution <<<<<<<<<<<<<<<<<<
Process_ID Resource_Max
1 3
2 2
3 1
Input the process 1 applys for(1 -- 8): 4
Assine resource 4 to process 1
Process 1 takes up 1 resource(s)...
Process_ID State Need Max_Require Possession Last_apply
1 R 2 3 1 4
2 R 2 2 0 0
3 R 1 1 0 0
Input the process 2 applys for(1 -- 9): 6
Assine resource 6 to process 2
Process 2 takes up 1 resource(s)...
Process_ID State Need Max_Require Possession Last_apply
1 R 2 3 1 4
2 R 1 2 1 6
3 R 1 1 0 0

```
Input the process 3 applys for(1 -- 10): 8
Assine resource 8 to process 3
Process 3 takes up 1 resource(s)...

-------------------------------------------------------------------------------
Process_ID     State     Need     Max_Require     Possession     Last_apply
-------------------------------------------------------------------------------
    1           R        2           3               1              4
-------------------------------------------------------------------------------
    2           R        1           2               1              6
-------------------------------------------------------------------------------
    3           F        0           1               1              8
-------------------------------------------------------------------------------
Process 3 is successfuly assigned!

Input the process 1 applys for(5 -- 9): 7
Assine resource 7 to process 1
Process 1 takes up 2 resource(s)...

-------------------------------------------------------------------------------
Process_ID     State     Need     Max_Require     Possession     Last_apply
-------------------------------------------------------------------------------
    1           R        1           3               2              7
-------------------------------------------------------------------------------
    2           R        1           2               1              6
-------------------------------------------------------------------------------
    3           F        0           1               1              8
-------------------------------------------------------------------------------
```

```
Input the process 2 applys for(7 -- 10): 9
Assine resource 9 to process 2
Process 2 takes up 2 resource(s)...

-------------------------------------------------------------------------
Process_ID     State     Need     Max_Require     Possession     Last_apply
-------------------------------------------------------------------------
    1           R        1           3               2              7
-------------------------------------------------------------------------
    2           F        0           2               2              9
-------------------------------------------------------------------------
    3           F        0           1               1              8
-------------------------------------------------------------------------
Process 2 is successfuly assigned!

Input the process 1 applys for(8 -- 10): 8
Assine resource 8 to process 1
Process 1 takes up 3 resource(s)...

-------------------------------------------------------------------------
Process_ID     State     Need     Max_Require     Possession     Last_apply
-------------------------------------------------------------------------
    1           F        0           3               3              8
-------------------------------------------------------------------------
    2           F        0           2               2              9
-------------------------------------------------------------------------
    3           F        0           1               1              8
-------------------------------------------------------------------------
Process 1 is successfuly assigned!
All of the processes are successfully assigned!

----------------------------Apply Information-----------------------------

Process_ID          Process_1          Process_2          Process_3
                        4                  6                  8
                        7                  9
                        8
-------------------------------------------------------------------------
```

五：思考题

(1) 死锁发生的条件？

1，互斥

2，占有并等待

3，非抢占

4，循环等待

(2) 避免死锁的方法有哪些？

资源分配法，银行家代码

(3) 你的算法采用什么思想预防或避免死锁的发生？

只要上述四个条件至少有一个不成立，就能预防死锁发生

(4) 效率如何？

资源分配图算法：每种资源类只有一个实例

银行家算法：每种资源类型有多个实例

六：实验总结

通过这两个小作业，我对银行家算法和按序分配策略实现资源分配有了更深刻的认识，通过编写和调试系统动态分配资源的简单模拟程序，观察死锁产生的条件，并采用适当的算法，有效地防止和避免死锁的发生，获益匪浅。

评价表格

考核标准	得分
（1）正确理解和掌握实验所涉及的概念和原理（20%）；	
（2）按实验要求合理设计数据结构和程序结构（20%）；	
（3）运行结果正确（20%）；	
（4）认真记录实验数据，原理及实验结果分析准确（20%）；	
(5)实验过程中,具有严谨的学习态度和认真、踏实、一丝不苟的科学作风(10%);	
（7）实验报告规范（10%）。	
合计	