武汉大学计算机学院

本科生实验报告

操作系统设计与实现

专 业 名 称 ：软件工程卓越工程师

课 程 名 称 ：操作系统设计

指 导 教 师 ：曾平

学 生 学 号 ：2019302110166

学 生 姓 名 ：陈子懿

二〇二一年六月

郑 重 声 明

本人呈交的实验报告，是在指导老师的指导下，独立进行实验工作所取得的成果，所有数据、图片资料真实可靠。尽我所知，除文中已经注明引用的内容外，本实验报告不包含他人享有著作权的内容。对本实验报告做出贡献的其他个人和集体，均已在文中以明确的方式标明。本实验报告的知识产权归属于培养单位。

本人签名：_____ 日期：_____

摘　要

操作系统设计与实现实验的实验目的是巩固《操作系统》课中所学到的关于 CPU 调度、内存分配、文件管理等等知识，并且运用这些理论知识来模拟操作系统内核所实现的功能。

关键词：处理机；主存储器；磁盘；同步

目录

1 处理机调度

1.1 实验内容及上机实验所用平台

选择一个调度算法，实现处理器调度。

1．1．1 实验内容

通过实习模拟在单处理器环境下的处理器调度，加深了解处理器调度的工作。

本实验采用高优先级优先调度算法。

1．1．2 上机平台

系统：Windows
IDE: vscode

1.2 数据结构及代码段分析

进程的状态分为就绪和完成：

```
enum PROCESS_STATUS{
    READY, ZOMBIE
};
```

每个进程包含一个 PCB 进程控制块，其中包含了

- 进程名——如 P1~P5。
- 指针——按优先数的大小把 5 个进程连成队列，用指针指出下一个进程 PCB 的首地址。
- 要求运行时间——假设进程需要运行的单位时间数。
- 优先数——赋予进程的优先数，调度时总是选取优先数大的进程先执行。
- 状态——假设两种状态：就绪和结束，用 R 表示就绪，用 E 表示结束。初始状态都为就绪状态。

```
struct PCB{
    bool operator<(const PCB& a) const{
        return priority < a.priority; //大顶堆
    }
```

```
    string processName;
    int priority;
   // PCB* nxtPCB;
    int demandTime;
    PROCESS_STATUS status;
}pool[5];
```

就绪进程队列是一个大根堆，在 PCB 结构体中已经实现了对于<的重载：

```
priority_queue<PCB> readyQueue;
```

打印就绪队列：

```
void printReadyQueue(){
    priority_queue<PCB> readyQueueClone = readyQueue;
    printf("\nreadyQueue:\n");
    printf("ProcessName\tPriority\tdemandTime\t\n");
    printf("-----------------------------------------------------\n");
    while(!readyQueueClone.empty()){
        PCB x = readyQueueClone.top();
        readyQueueClone.pop();
        printf("%s\t\t%d\t\t%d\t\n", x.processName.c_str(), x.priority, x.demandTime);
    }
    printf("\n");
}
```

开始执行进程：

```
void start(){
    while(!readyQueue.empty()){
        printReadyQueue();
        PCB x = readyQueue.top();
        readyQueue.pop();
        cout << "currently running: " << x.processName << endl;
        cout << "remaining demand time: " << --x.demandTime << endl;
        cout << "priority after running: " << --x.priority << endl;
        if(x.demandTime <= 0) x.status = ZOMBIE;
        else readyQueue.push(x);
        system("pause");
    }
}
```

Main 函数，初始化进程参数：

```cpp
int main(){
    //10 5freopen("h.in", "r", stdin);
    cout << "please input the priority and demand time of each process." << endl;
    for(int i = 0; i <= 4; i++){
        int pri, dt;
        cin >> pri >> dt;
        pool[i].priority = pri;
        pool[i].demandTime = dt;
        std::ostringstream oss;
        oss << "P " << i;
        std::string str = oss.str();
        pool[i].processName = str;

        pool[i].status = READY;
        cout<<pool[i].processName<<endl;
        readyQueue.push(pool[i]);

    }
    start();
    return 0;
}
```

1.3 调试过程

进入 Windows/Linux 系统，输入 g++ priority.cpp -o priority
然后运行./priority
运行结果如下：

```
PS D:\Study\大学资料\操作系统\实验\1\1> .\priority.exe
please input the priority and demand time of each process.
10 5
P 0
4 6
P 1
7 3
P 2
5 8
P 3
2 6
P 4

```

```
readyQueue:
ProcessName     Priority        demandTime
-----------------------------------------------------
P 0             10              5
P 2             7               3
P 3             5               8
P 1             4               6
P 4             2               6

currently running: P 0
remaining demand time: 4
priority after running: 9
请按任意键继续. . .

readyQueue:
ProcessName     Priority        demandTime
-----------------------------------------------------
P 0             9               4
P 2             7               3
P 3             5               8
P 1             4               6
P 4             2               6

currently running: P 0
remaining demand time: 3
priority after running: 8
请按任意键继续. . .

readyQueue:
ProcessName     Priority        demandTime
-----------------------------------------------------
P 0             8               3
P 2             7               3
P 3             5               8
P 1             4               6
P 4             2               6

currently running: P 0
remaining demand time: 2
priority after running: 7
请按任意键继续. . .

readyQueue:
ProcessName     Priority        demandTime
```

```
--------------------------------------------------
P 2             7               3
P 0             7               2
P 3             5               8
P 1             4               6
P 4             2               6

currently running: P 2
remaining demand time: 2
priority after running: 6
请按任意键继续. . .

readyQueue:
ProcessName     Priority        demandTime
--------------------------------------------------
P 0             7               2
P 2             6               2
P 3             5               8
P 1             4               6
P 4             2               6

currently running: P 0
remaining demand time: 1
priority after running: 6
请按任意键继续. . .

readyQueue:
ProcessName     Priority        demandTime
--------------------------------------------------
P 2             6               2
P 0             6               1
P 3             5               8
P 1             4               6
P 4             2               6

currently running: P 2
remaining demand time: 1
priority after running: 5
请按任意键继续. . .

readyQueue:
ProcessName     Priority        demandTime
--------------------------------------------------
P 0              6               1
```

```
P 3             5               8
P 2             5               1
P 1             4               6
P 4             2               6

currently running: P 0
remaining demand time: 0
priority after running: 5
请按任意键继续. . .

readyQueue:
ProcessName     Priority        demandTime
--------------------------------------------------
P 3             5               8
P 2             5               1
P 1             4               6
P 4             2               6

currently running: P 3
remaining demand time: 7
priority after running: 4
请按任意键继续. . .

readyQueue:
ProcessName     Priority        demandTime
--------------------------------------------------
P 2             5               1
P 1             4               6
P 3             4               7
P 4             2               6

currently running: P 2
remaining demand time: 0
priority after running: 4
请按任意键继续. . .

readyQueue:
ProcessName     Priority        demandTime
--------------------------------------------------
P 1             4               6
P 3             4               7
P 4             2               6

currently running: P 1
```

```
remaining demand time: 5
priority after running: 3
请按任意键继续. . .

readyQueue:
ProcessName     Priority        demandTime
-----------------------------------------------------
P 3             4               7
P 1             3               5
P 4             2               6

currently running: P 3
remaining demand time: 6
priority after running: 3
请按任意键继续. . .

readyQueue:
ProcessName     Priority        demandTime
-----------------------------------------------------
P 1             3               5
P 3             3               6
P 4             2               6

currently running: P 1
remaining demand time: 4
priority after running: 2
请按任意键继续. . .

readyQueue:
ProcessName     Priority        demandTime
-----------------------------------------------------
P 3             3               6
P 4             2               6
P 1             2               4

currently running: P 3
remaining demand time: 5
priority after running: 2
请按任意键继续. . .

readyQueue:
ProcessName     Priority        demandTime
-----------------------------------------------------
P 4             2               6
```

```
P 1             2               4
P 3             2               5

currently running: P 4
remaining demand time: 5
priority after running: 1
请按任意键继续. . .

readyQueue:
ProcessName     Priority        demandTime
-------------------------------------------------
P 1             2               4
P 3             2               5
P 4             1               5

currently running: P 1
remaining demand time: 3
priority after running: 1
请按任意键继续. . .

readyQueue:
ProcessName     Priority        demandTime
-------------------------------------------------
P 3             2               5
P 4             1               5
P 1             1               3

currently running: P 3
remaining demand time: 4
priority after running: 1
请按任意键继续. . .

readyQueue:
ProcessName     Priority        demandTime
-------------------------------------------------
P 4             1               5
P 1             1               3
P 3             1               4

currently running: P 4
remaining demand time: 4
priority after running: 0
请按任意键继续. . .

```

```
readyQueue:
ProcessName     Priority        demandTime
-------------------------------------------------
P 1             1               3
P 3             1               4
P 4             0               4

currently running: P 1
remaining demand time: 2
priority after running: 0
请按任意键继续. . .

readyQueue:
ProcessName     Priority        demandTime
-------------------------------------------------
P 3             1               4
P 4             0               4
P 1             0               2

currently running: P 3
remaining demand time: 3
priority after running: 0
请按任意键继续. . .

readyQueue:
ProcessName     Priority        demandTime
-------------------------------------------------
P 4             0               4
P 1             0               2
P 3             0               3

currently running: P 4
remaining demand time: 3
priority after running: -1
请按任意键继续. . .

readyQueue:
ProcessName     Priority        demandTime
-------------------------------------------------
P 1             0               2
P 3             0               3
P 4             -1              3

currently running: P 1
```

```
remaining demand time: 1
priority after running: -1
请按任意键继续. . .

readyQueue:
ProcessName     Priority        demandTime
-----------------------------------------------------
P 3             0               3
P 4             -1              3
P 1             -1              1

currently running: P 3
remaining demand time: 2
priority after running: -1
请按任意键继续. . .

readyQueue:
ProcessName     Priority        demandTime
-----------------------------------------------------
P 4             -1              3
P 1             -1              1
P 3             -1              2

currently running: P 4
remaining demand time: 2
priority after running: -2
请按任意键继续. . .

readyQueue:
ProcessName     Priority        demandTime
-----------------------------------------------------
P 1             -1              1
P 3             -1              2
P 4             -2              2

currently running: P 1
remaining demand time: 0
priority after running: -2
请按任意键继续. . .

readyQueue:
ProcessName     Priority        demandTime
-----------------------------------------------------
P 3             -1              2
```

```
P 4             -2              2

currently running: P 3
remaining demand time: 1
priority after running: -2
请按任意键继续. . .

readyQueue:
ProcessName     Priority        demandTime
--------------------------------------------------
P 4             -2              2
P 3             -2              1

currently running: P 4
remaining demand time: 1
priority after running: -3
请按任意键继续. . .

readyQueue:
ProcessName     Priority        demandTime
--------------------------------------------------
P 3             -2              1
P 4             -3              1

currently running: P 3
remaining demand time: 0
priority after running: -3
请按任意键继续. . .

readyQueue:
ProcessName     Priority        demandTime
--------------------------------------------------
P 4             -3              1

currently running: P 4
remaining demand time: 0
priority after running: -4
请按任意键继续. . .
```

1.4 实验总结

- 优先级调度（具有相同优先级的按 FCFS 调度）

- 在作业调度中，从后备作业队列中选择若干优先级高的作业调入内存。
- 在进程调度中，将处理机分配给就绪队列中优先级最高的进程。
- 调度方式分类
  - 非抢占式
  - 抢占式
- 优先级分类
- 静态：创建进程时确定
  - 特点：简单易行，系统开销小，但不精确
- 动态：进程运行过程中更改
  - 问题：饥饿，低优先级的进程可能永远得不到运行
  - 解决方案：老化，视进程等待时间的延长提高其优先级

2 主存空间的分配和回收

2.1 实验内容及上机实验所用平台

2．1．1 实验内容

主存储器空间的分配和回收。

本实验采用的时可变分区分配方式。

2．1．2 上机平台

系统：Windows
IDE: vscode

2.2 数据结构及代码段分析

空闲块结构体，包括了空闲块的 index，大小，始址：

```
1.  struct Item{
2.      int index;
3.      int szeInByte;
4.      int startAddr;
5.      bool operator < (int a) const{
6.          return startAddr < a;
7.      }
8.  };
```

所有的空闲块

```
9.  vector<Item> availTable;
```

所有的作业：

```
10. map<int, Item> jobs;
```

内存每一字节的占用情况，采用差分法，申请空间时，在始址的地方+1，始址+大小的地方+1，这样逐位求出前缀和就能获得内存每一位的占用情况：

```
11. const int MAXN = 128;
12. int memory[MAXN], memoryPrefixSum[MAXN];
```

初始化，先放入一块完整的空闲块。

```
void init(){
    availTable.clear();
    availTable.push_back(Item{1, MAXN, 0});
    memset(memory, 0, sizeof memory);
    memset(memoryPrefixSum, 0, sizeof memoryPrefixSum);
}
```

更新空闲块表，主要工作是合并邻接空闲块，并且将空闲块按照开始地址升序排列：

```
void updateAvailTable(){
    int curI = 1;
    bool suc = true;
    for(auto i = availTable.begin(); i != availTable.end(); i++){
        if(i->szeInByte == 0) availTable.erase(i);
        else{
            i->index = curI++;
            if(i!=availTable.begin()){
                if(i->startAddr == (i - 1)->startAddr+(i-1)->szeInByte){
                    int str = (i - 1)->startAddr, sze = (i-1)->szeInByte+i->szeInByte;
                    *i = Item{0, sze, str};
                    availTable.erase(i - 1);
                    suc = false;
                    break;
                }
            }
        }
    }
    if(!suc) updateAvailTable();
}
```

请求空间，内存标记方法已经在上面说明；此外还要记录该作业对应的是哪块：

```
bool askForSpace(int index, int byte){
    for(auto i = availTable.begin(); i != availTable.end(); i++){
        if(i->szeInByte >= byte){
            i->szeInByte -= byte;

            jobs[index] = Item{index, byte, i->startAddr};

            memory[i->startAddr]++;
            memory[i->startAddr + byte]--;

            i->startAddr += byte;
            if(!i->szeInByte) updateAvailTable();
```

```
51.
52.             return true;
53.         }
54.     }
55.     return false;
56. }
57.
```

打印出现在所有的空闲块信息：

```
58. void displayAvailTable(){
59.     cout << "-----------------------------------------------"<<endl;
60.     cout << "index\t|\tsze\t|\tstart\t|\t"<<endl;
61.     for(auto i = availTable.begin(); i != availTable.end(); i++){
62.         cout << i->index << "\t|\t" << i->szeInByte<<"\t|\t"<<i->startAddr<<
    "\t|\t\n";
63.     }
64.     cout << "-----------------------------------------------"<<endl;
65. }
```

展示出当前内存的占用情况：

```
66. void displayMemory(){
67.     memoryPrefixSum[0] = memory[0];
68.     if(memoryPrefixSum[0]) cout << "#" ;
69.     else cout << "=";
70.     for(int i = 1; i < MAXN; i++){
71.         memoryPrefixSum[i] =  memory[i] + memoryPrefixSum[i - 1];
72.         if(memoryPrefixSum[i]) cout << "#" ;
73.         else cout << "=";
74.     }
75.     cout << endl;
76. }
```

释放某作业占用的空间：

```
77. void releaseMemory(int index){
78.     int start = jobs[index].startAddr;
79.     int sze = jobs[index].szeInByte;
80.     memory[start]--;
81.     memory[start + sze]++;
82.     auto pos = lower_bound(availTable.begin(), availTable.end(), start);
83.     availTable.insert(pos, Item{0, sze, start});
84.     updateAvailTable();
85.    // return true;
```

```
}
```

Main 函数，接收来自键盘的命令，执行相应操作：

```
int n;
int main(){
    //freopen("h.in", "r", stdin);
    init();
    displayMemory();
    displayAvailTable();
    //return 0;
    cout << "how many opt?." << endl;
    cin >> n;
    for(int i = 1; i <= n; i++){
        cout <<endl<< "opt"<<i<<":"<<endl;
        char opt;
        cin >> opt;
         switch(opt){
            case 'A':{
                int sze;
                cin >> sze;
                if(askForSpace(i, sze)){
                    cout << "ok!";
                }
                else{
                    cout << "damn!";
                }
                break;
            }
            case 'R':{
                int index;
                cin >> index;
                releaseMemory(index);
                break;
            }
         }
         displayMemory();
         displayAvailTable();
    }
    return 0;
}
```

2.3 调试过程

进入 Windows/Linux 系统，输入 g++ dynamic.cpp -o dynamic
运行./dynamic
运行结果如下：

```
PS D:\Study\大学资料\操作系统\实验\2\1> .\dynamic.exe
================================================================================
------------------------------------------------
index   |       sze     |       start   |
1       |       128     |       0       |
------------------------------------------------
how many opt?.
5

opt1:
A 50
ok!#################################################=========================
------------------------------------------------
index   |       sze     |       start   |
1       |       78      |       50      |
------------------------------------------------

opt2:
A 30
ok!#############################################################################
------------------------------------------------
index   |       sze     |       start   |
1       |       48      |       80      |
------------------------------------------------

opt3:
A 10
ok!#########################################################################
------------------------------------------------
index   |       sze     |       start   |
1       |       38      |       90      |
------------------------------------------------

opt4:
```

```
R 2
################################################==========================
====##########=======================================
------------------------------------------------
index   |       sze     |       start   |
1       |       30      |       50      |
2       |       38      |       90      |
------------------------------------------------

opt5:
R 1
================================================================================
====##########=======================================
------------------------------------------------
index   |       sze     |       start   |
1       |       80      |       0       |
2       |       38      |       90      |
------------------------------------------------
```

2.4 实验总结

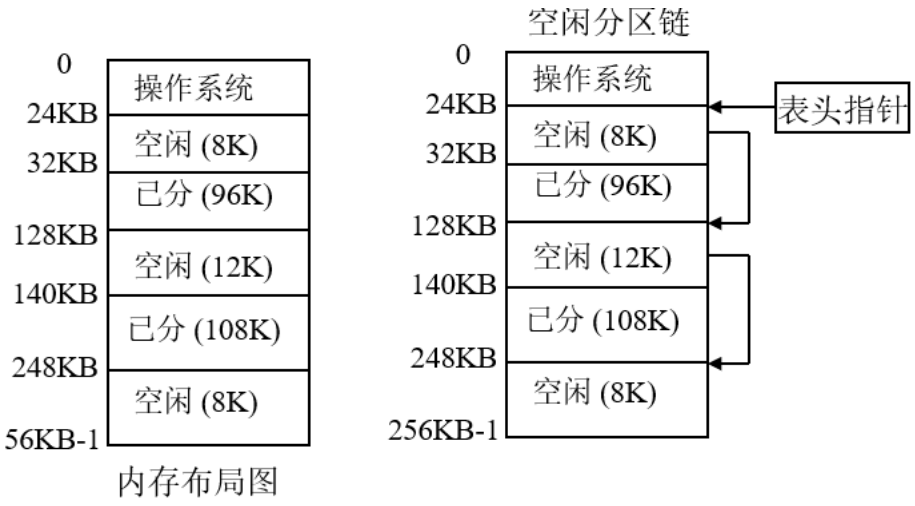
- 动态分区：根据需要动态建立分区
  - 数据结构
    - 空闲分区表。用一个空闲分区表来登记系统中的空闲分区。其表项类似于固定分区。

内存布局图

空闲分区表

分区号	大小	起始地址
1	8KB	24KB
2	12KB	128KB
3	8KB	248KB
4	…	…
5	…	…

    - 空闲分区链。将内存中的空闲分区以链表方式链接起来，构成空闲分区链。



- 首次适应算法：空闲分区按地址递增次序排列，找到第一个满足大小的空闲分区，划分一块分配，剩余空间留在空闲分区表中
  - 特点：优先利用内存低地址端，高地址端有大空闲区。但低地址端有许多小空闲分区时会增加查找开销。

3 磁盘存储空间的分配和回收

3.1 实验内容及上机实验所用平台

3．1．1 实验内容

模拟磁盘空闲空间的表示方法，以及模拟实现磁盘空间的分配和回收。

3．1．2 上机平台

系统：Windows
IDE: vscode

3.2 数据结构及代码段分析

一些常数：柱面数、磁道数、记录数：

```
const int zhumian = 8, cidao = 2, jilu = 4;
#define total (zhumian*cidao*jilu)
```

将十进制转换为二进制的字符串：

```
string intToBiStr(int x, int len){
    string ret = "";
    while(x){
        ret += x%2+'0';
        x/=2;
    }
    while(ret.length() < len) ret+="0";
    reverse(ret.begin(), ret.end());
    return ret;
}
```

位置图的设置与读取：

```
namespace BITMAP{
    int bitmap[65];
```

1） 将二进制字符串转换为十进制整数

```
    int biToInt(const char* x){
        int len = strlen(x);
```

```
17.         int ret = 0;
18.         for(int i = 0; i < len; i++){
19.             if(x[i] == '1') ret += pow(2, len - i - 1);
20.         }
21.         return ret;
22.     }
```

2） 获取位示图的某一位

```
23.     int getBit(int x){
24.         return bitmap[x];
25.     }
26.
27.     int getBit(const char* x){
28.         return bitmap[biToInt(x)];
29.     }
```

3） 设置位示图的某一位

```
30.     void setBit(int x, bool f){
31.         bitmap[x] = f;
32.     }
33.
34.     void setBit(char *s, bool f){
35.         bitmap[biToInt(s)] = f;
36.     }
```

4） 打印出整个位示图

```
37.     void displayBitMap(){
38.         cout<<"------------------------------------------------------------"<<e
    ndl;
39.
40.         for(int i = 0; i < zhumian; i++){
41.             cout<<i<<": ";
42.             for(int j = 0; j < cidao*jilu; j++)
43.                 cout<< getBit(i*cidao*jilu+j);
44.             cout<<endl;
45.         }
46.         cout<<"------------------------------------------------------------"<<
    endl;
47.     }
48. };
```

磁盘的相关操作：

```
namespace DISK{
    struct add{
        int zm,cd,jl;
    };
```

1） 申请空间

```
    using ret = pair<bool,add>;
    ret askForSpace(){
        int now = -1;
        for(int i = 0; i < total; i++){
            if(!BITMAP::getBit(i)){
                now = i;
                break;
            }
        }
        if(now >= 0){
            BITMAP::setBit(now, 1);

            return make_pair(true, add{now/(cidao*jilu), (now%(cidao*jilu))/4, (now%(cidao*jilu))%4});
        }
        else {

            return make_pair(false, add{0,0,0});
        }
    }
```

2） 释放空间

```
    using ret2 = pair<bool, int>;
    ret2 releaseSpace(add a){
        int biAdd = a.zm*(cidao*jilu) + a.cd*jilu + a.jl;
        if(!BITMAP::getBit(biAdd)){
            return make_pair(false, 0);
        }
        else {
            BITMAP::setBit(biAdd, 0);
            return make_pair(true, biAdd);
        }
    }
}

```

Main 函数，初始化位示图和磁盘，接收来自键盘的指令并执行相关操作：

```
int n;
int main(){
    //freopen("h.in", "r", stdin);
    //cout<<intToBiStr(4, 4)<<endl;
    BITMAP::displayBitMap();
   // displayAvailTable();
    //return 0;
    cout << "how many opt?." << endl;
    cin >> n;
    for(int i = 1; i <= n; i++){
        cout <<endl<< "opt"<<i<<":"<<endl;
        char opt;
        cin >> opt;
        switch(opt){
            case 'A':{
                DISK::ret ret = DISK::askForSpace();
                if(ret.first){
                    cout << "succeeded, zhumian:"<<ret.second.zm<<" cidao:"<<ret.second.cd<<" jilu:"<<ret.second.jl<<endl;
                }
                else cout << "failed, full." << endl;
                break;
             }
             case 'R':{
                int zm, cd, jl;
                cin>>zm>>cd>>jl;
                DISK::ret2 ret = DISK::releaseSpace(DISK::add{zm,cd,jl});
                if(ret.first){
                    cout << "succeeded, bit address:"<<intToBiStr(ret.second, 6)<<endl;
                }
                else cout<< "failed, it's available."<<endl;
                break;
             }
         }
         BITMAP::displayBitMap();
     }
     return 0;
}
```

3.3 调试过程

进入 Windows/Linux 系统，输入 g++ bitmap.cpp -o bitmap
运行./bitmap
运行结果如下：

```
PS D:\Study\大学资料\操作系统\实验\3\2> .\bitmap.exe
------------------------------------------------------------
0: 00000000
1: 00000000
2: 00000000
3: 00000000
4: 00000000
5: 00000000
6: 00000000
7: 00000000
------------------------------------------------------------
how many opt?.
10

opt1:
A
succeeded, zhumian:0 cidao:0 jilu:0
------------------------------------------------------------
0: 10000000
1: 00000000
2: 00000000
3: 00000000
4: 00000000
5: 00000000
6: 00000000
7: 00000000
------------------------------------------------------------

opt2:
A
succeeded, zhumian:0 cidao:0 jilu:1
------------------------------------------------------------
0: 11000000
1: 00000000
2: 00000000
3: 00000000
4: 00000000
5: 00000000
6: 00000000
7: 00000000
------------------------------------------------------------
```

```
42.
43. opt3:
44. A
45. succeeded, zhumian:0 cidao:0 jilu:2
46. ------------------------------------------------------------
47. 0: 11100000
48. 1: 00000000
49. 2: 00000000
50. 3: 00000000
51. 4: 00000000
52. 5: 00000000
53. 6: 00000000
54. 7: 00000000
55. ------------------------------------------------------------
56.
57. opt4:
58. A
59. succeeded, zhumian:0 cidao:0 jilu:3
60. ------------------------------------------------------------
61. 0: 11110000
62. 1: 00000000
63. 2: 00000000
64. 3: 00000000
65. 4: 00000000
66. 5: 00000000
67. 6: 00000000
68. 7: 00000000
69. ------------------------------------------------------------
70.
71. opt5:
72. A
73. succeeded, zhumian:0 cidao:1 jilu:0
74. ------------------------------------------------------------
75. 0: 11111000
76. 1: 00000000
77. 2: 00000000
78. 3: 00000000
79. 4: 00000000
80. 5: 00000000
81. 6: 00000000
82. 7: 00000000
83. ------------------------------------------------------------
84.
85. opt6:
```

```
A
succeeded, zhumian:0 cidao:1 jilu:1
------------------------------------------------------------
0: 11111100
1: 00000000
2: 00000000
3: 00000000
4: 00000000
5: 00000000
6: 00000000
7: 00000000
------------------------------------------------------------

opt7:
A
succeeded, zhumian:0 cidao:1 jilu:2
------------------------------------------------------------
0: 11111110
1: 00000000
2: 00000000
3: 00000000
4: 00000000
5: 00000000
6: 00000000
7: 00000000
------------------------------------------------------------

opt8:
A
succeeded, zhumian:0 cidao:1 jilu:3
------------------------------------------------------------
0: 11111111
1: 00000000
2: 00000000
3: 00000000
4: 00000000
5: 00000000
6: 00000000
7: 00000000
------------------------------------------------------------

opt9:
A
succeeded, zhumian:1 cidao:0 jilu:0
```

```
130. ----------------------------------------------------------
131. 0: 11111111
132. 1: 10000000
133. 2: 00000000
134. 3: 00000000
135. 4: 00000000
136. 5: 00000000
137. 6: 00000000
138. 7: 00000000
139. ----------------------------------------------------------
140.
141. opt10:
142. R 0 1 3
143. succeeded, bit address:000111
144. ----------------------------------------------------------
145. 0: 11111110
146. 1: 10000000
147. 2: 00000000
148. 3: 00000000
149. 4: 00000000
150. 5: 00000000
151. 6: 00000000
152. 7: 00000000
153. ----------------------------------------------------------
```

3.4 实验总结

- 磁盘上的一系列同心圆称为磁道（track），磁道沿径向又分成大小相等的多个扇区（sector），与盘片中心有一定距离的所有磁道组成一个柱面（cylinder）。
- 磁盘上的每个物理块可用柱面号，磁头号和扇区号表示。
- 逻辑扇区号=柱面号*每柱面扇区数+
  - 磁头号*每磁头扇区数+
  - 扇区号
- 位示图
  - 因位示图比较小，可以保存在主存中，因此空间的分配与回收较快；但需要进行位示图中二进制所在位置与盘块号之间的转换

4 进程创建

4.1 实验内容及上机实验所用平台

4．1．1 实验内容

利用 fork()系统调用创建进程。

4．1．2 上机平台

系统：Windows + Linux 虚拟机
IDE: vscode

4.2 数据结构及代码段分析

```
#include<unistd.h>
#include<stdio.h>
int main()
{
    pid_t pid;
    int n; //count of child processes to create;
    char outputContent;

    scanf("%d", &n);
    for(int i = 0; i < n; i++){
        pid=fork();
        if(pid==-1){
            printf("creation failed. error.\n");
            break;
        }
        else if(pid == 0){
            outputContent = i + 'A' ;
            printf("This is child process (id:%d) (parent process id:%d): %c\n", getpid(), getppid(), outputContent);
            break;
        }
        else {
            printf("This is parent process (id:%d), creating child Process %d\n", getpid(), pid);
        }
```

```
    }
    return 0;
}
```

fork()函数用于复制父进程以生成一个子进程，它在父进程和子进程中都会有返回值：

1） -1：创建子进程时出现错误。

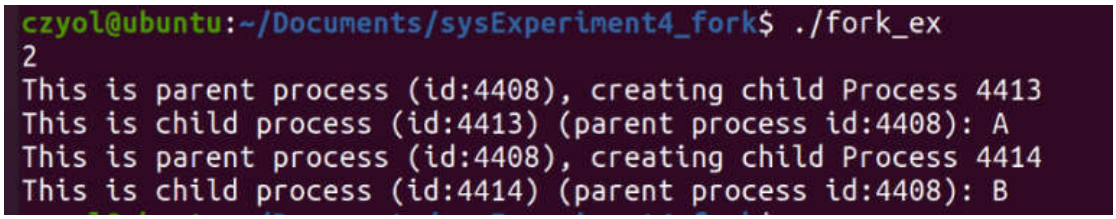
2） 0：创建子进程成功，当前进程为子进程。

3） >0：创建子进程成功，当前进程为父进程。

创建多个进程时，如果 fork()返回值为 0（即当前进程为子进程），那么就要退出循环，避免子进程继续创建子进程。

4.3 调试过程

进入 Ubuntu 虚拟机，输入 gcc fork_ex.cpp -o fork_ex
然后运行./fork_ex
运行结果如下：



4.4 实验总结

进程是资源分配的基本单位，也是独立运行的基本单位

4.4.1 引入进程的目的

1） 使多道程序能并发执行，以改善资源利用率及提高系统吞吐量

2） 描述程序动态执行过程的性质

4.4.2 进程和程序的区别

1）进程是动态概念，程序是静态概念；进程是程序在处理机上的一次执行过程，而程序是指令的集合。

2）进程是暂时的，程序是永久的。进程是一个状态变化的过程；程序可以长久保存。

3）进程与程序的组成不同。进程的组成包括程序、数据和进程控制块。

4）进程与程序是密切相关的。一个程序可以对应多个进程；一个进程可以包括多个程序。

5）进程可以创建新进程，而程序不能形成新程序。

5 进程同步

5.1 实验内容及上机实验所用平台

5．1．1 实验内容

模拟实现同步机构，以避免发生进程执行时可能出现的与时间有关的错误。

5．1．2 上机平台

系统：Windows + Linux 虚拟机
IDE: vscode

5.2 数据结构及代码段分析

进程的状态：执行、就绪、等待、完成：

```
enum PROCESS_STATUS{
    RUNNING, READY, WAITING, ZOMBIE
};
```

每个进程都有一个 PCB 进程控制块，包含内容

进程名
状态
等待原因
断点

```
struct PCB{
    std::string name;
    PROCESS_STATUS status;
    std::string reasonForWaiting;
    int breakpoint;
    int pc;
};
```

信号量 S1、S2：

```
int s1, s2;
```

就绪队列，其中保存的是进程控制块的指针：

```
std::vector<PCB*> readyQueue;
```

产品池，生产者的产品将放在里面，消费者也将从其中获取产品来消费：

```
char productPool[100];

```

生产者和消费者的进程控制块：

```
PCB consumer, producer;
```

当前进程指针，指向生产者或消费者：

```
PCB *currentPCB;
```

临界区的相关操作：

```
namespace Sync{
```

1） P 操作

```
    bool p(PCB &pcb, int &sem, std::string semName){
        sem--;
        if(sem < 0){
            pcb.status = WAITING;
            pcb.reasonForWaiting = semName;
            //pcb.reasonForWaiting.append(semName);
            printf("reason for waiting: %s\n", pcb.reasonForWaiting.c_str())
;
            pcb.breakpoint = pcb.pc;
            if(pcb.name == "consumer" && producer.status == ZOMBIE){
                currentPCB = nullptr;
                return false;
            }
            int randid = rand() % readyQueue.size();
            std::cout<<randid<<std::endl;
            currentPCB = readyQueue[randid];
            //std::cout<<randid<<" "<<currentPCB->name.c_str()<<" "<<current
PCB->pc<<" "<<std::endl;
            readyQueue.erase(readyQueue.begin() + randid);
            return false;

        }
        return true;
    }

```

2） V 操作

```
41.     bool v(PCB &pcb, int &sem, std::string semName){
42.         sem++;
43.         if(sem <= 0){
44.             //release a process
45.             std::string reasonForWaiting = semName;
46.             if(consumer.reasonForWaiting == semName) {
47.
48.                 consumer.status = READY;
49.                 consumer.reasonForWaiting = "";
50.                 consumer.pc = consumer.breakpoint;
51.                 readyQueue.push_back(&consumer);
52.                 printf("consumer is waken up\n");
53.             }
54.             else if(producer.reasonForWaiting == semName){
55.                 producer.status = READY;
56.                 producer.reasonForWaiting = "";
57.                 producer.pc = producer.breakpoint;
58.                 readyQueue.push_back(&producer);
59.                 printf("producer is waken up\n");
60.             }
61.
62.             return false;
63.         }
64.         return true;
65.     }
66. }
67.
```

生产者的相关操作：

```
68. namespace Producer{
69.     int id = 0;
70.     int in = 0;
71.     char C;
```

1） 生产者的指令集合，为了模拟 PC 跳转，将每一个 PC 值对应要执行的代码封装为一个函数；PA 中保存的是 pc 值到对应函数指针的映射：

```
72.     typedef void (*code)();
73.     std::map<int, code> PA;
```

2） 生产者执行完一条指令后，进入就绪状态，加入就绪队列。然后从就绪队列中随机选择一个进程作为当前进程：

```
74.     void ready(){
75.         producer.status = READY;
76.         
77.         readyQueue.push_back(&producer);
78.         int randid = rand() % readyQueue.size();
79.         std::cout<<randid<<std::endl;
80.         currentPCB = readyQueue[randid];
81.         readyQueue.erase(readyQueue.begin() + randid);
82.         printf("already put into ready queue\n");
83.     }
```

3） 生产者生产产品

```
84.     void produce(){
85.         C = 'A' + (id++);
86.         printf("producing a product %c\n", C);
87.         ready();
88.     }
```

4） 生产者将生产的产品放入产品池中

```
89.     void put(){
90.         productPool[in++] = C;
91.         in %= 10;
92.         printf("putting product %c into the pool\n", C);
93.         ready();
94.     }
```

5） 生产者执行 P(s1)操作

```
95.     void ps1(){
96.         printf("producer is excuting p(s1) %d\n", s1 - 1);
97.         if(Sync::p(producer,s1, "s1")){
98.             ready();
99.         }else{
100.             printf("producer is waiting\n");
101.         }
102.     }
```

6） 生产者执行 V(s2)操作

```
103.     void vs2(){
104.         printf("producer is excuting v(s2) %d\n", s2 + 1);
105.         if(Sync::v(producer,s2, "s2"))
106.             ready();
```

```
        else {
            //printf("producer is waken up\n");
        }
    }
```

7） 生产者执行 goto(0)操作，回到第一条语句：

```
    void goto0(){
        printf("producer is excuting goto(0)\n");
        producer.pc = 0;
        ready();
    }
}
```

消费者的相关操作，与生产者类似：

```
namespace Consumer{
    int outId = 0;
    char C;
    typedef void (*code)();
    std::map<int, code> SA;

    void ready(){
        consumer.status = READY;
        readyQueue.push_back(&consumer);
        int randid = rand() % readyQueue.size();
        std::cout<<randid<<std::endl;
        currentPCB = readyQueue[randid];
        readyQueue.erase(readyQueue.begin() + randid);
        printf("already put into ready queue\n");
    }

    void get(){
        C = productPool[outId++];
        printf("consumer got a product %c from the pool\n", C);
        outId %= 10;
        ready();
    }
    void consume(){
        printf("consumer is consuming product %c\n", C);
        int ip = ((outId - 1) + 10) % 10;
        productPool[ip] = '0';
        ready();
    }
    void ps2(){
```

```
146.         printf("consumer is excuting p(s2) %d\n", s2 - 1);
147.         if(Sync::p(consumer,s2, "s2"))
148.             ready();
149.         else printf("consumer is waiting\n");
150.     }
151.     void vs1(){
152.         printf("consumer is excuting v(s1) %d\n", s1 + 1);
153.         if(Sync::v(consumer,s1, "s1"))
154.             ready();
155.         else{
156.             //printf("consumere is waken up\n");
157.         }
158.     }
159.     void goto0(){
160.         printf("consumer is excuting goto(0)\n");
161.         consumer.pc = 0;
162.         ready();
163.     }
164. }
```

初始化消费者进程，包括名字、状态、等待原因、pc 值、每个 pc 值所对应的操作（函数指针）：

```
165. void initConsumer(){
166.     consumer.name = "consumer";
167.     consumer.status = READY;
168.     consumer.reasonForWaiting = "";
169.     consumer.breakpoint = 0;
170.     consumer.pc = 0;
171.     Consumer::SA[0] = Consumer::ps2;
172.     Consumer::SA[1] = Consumer::get;
173.     Consumer::SA[2] = Consumer::vs1;
174.     Consumer::SA[3] = Consumer::consume;
175.     Consumer::SA[4] = Consumer::goto0;
176. }
```

初始化生产者进程，包括名字、状态、等待原因、pc 值、每个 pc 值所对应的操作（函数指针）：

```
177. void initProducer(){
178.     producer.name = "producer";
179.     producer.status = READY;
180.     producer.reasonForWaiting = "";
181.     producer.breakpoint = 0;
182.     producer.pc = 0;
```

```
    Producer::PA[0] = Producer::produce;
    Producer::PA[1] = Producer::ps1;
    Producer::PA[2] = Producer::put;
    Producer::PA[3] = Producer::vs2;
    Producer::PA[4] = Producer::goto0;
}
```

Main 函数，接收来自键盘的命令并执行相应操作：

```
int main(){
    initConsumer();
    initProducer();

    s1 = 10;
    s2 = 0;

    srand(0);
    readyQueue.push_back(&consumer);
    currentPCB = &producer;
    std::cout<<"starting...\n"<<std::endl;
    while(currentPCB != nullptr){
        printf("------------------------------------\n");
        printf("\n\npool: ");
        for(int i=0; i < Producer::in; i++){
            printf("%c ", productPool[i]);
        }
        printf("\nreadyQueue:");
        for(auto x : readyQueue){
            printf("%s ", x->name.c_str());
        }
        printf("\n");
        printf("currently running: %s, pc:%d\n", currentPCB->name.c_str(), currentPCB->pc);
        //char x; std::cin>>x;
        int curPC = (*currentPCB).pc++;
        if(currentPCB->name == "producer"){
            (*Producer::PA[curPC])();
            char opt;
            printf("stop the producer from running? y/n ");
            std::cin >> opt;
            if(opt == 'y' || opt == 'Y'){
                producer.status = ZOMBIE;
                currentPCB = &consumer;
                readyQueue.clear();
            }
```

```
            else if(opt == 'n' || opt == 'N'){
                continue;
            }
        }
        else if(currentPCB->name == "consumer"){
            (*Consumer::SA[curPC])();
            char opt;
            std::cin >> opt;
        }
    }

}
```

5.3 调试过程

进入 Ubuntu 虚拟机，输入 gcc fork_ex.cpp -o fork_ex
然后运行./fork_ex
运行结果如下：

```
czyol@ubuntu:~/Documents/sysExperiment5_pv_sem$ ./pv
starting...

--------------------------------------


pool:
readyQueue:consumer
currently running: producer, pc:0
producing a product A
1
already put into ready queue
stop the producer from running? y/n n
--------------------------------------


pool:
readyQueue:consumer
currently running: producer, pc:1
producer is excuting p(s1) 9
0
already put into ready queue
stop the producer from running? y/n n
```

```
24. ------------------------------------
25.
26.
27. pool:
28. readyQueue:producer
29. currently running: consumer, pc:0
30. consumer is excuting p(s2) -1
31. reason for waiting: s2
32. 0
33. consumer is waiting
34.
35. .
36. ------------------------------------
37.
38.
39. pool:
40. readyQueue:
41. currently running: producer, pc:2
42. putting product A into the pool
43. 0
44. already put into ready queue
45. stop the producer from running? y/n n
46. ------------------------------------
47.
48.
49. pool: A
50. readyQueue:
51. currently running: producer, pc:3
52. producer is excuting v(s2) 0
53. consumer is waken up
54. stop the producer from running? y/n n
55. ------------------------------------
56.
57.
58. pool: A
59. readyQueue:consumer
60. currently running: producer, pc:4
61. producer is excuting goto(0)
62. 1
63. already put into ready queue
64. stop the producer from running? y/n n
65. ------------------------------------
66.
67.
```

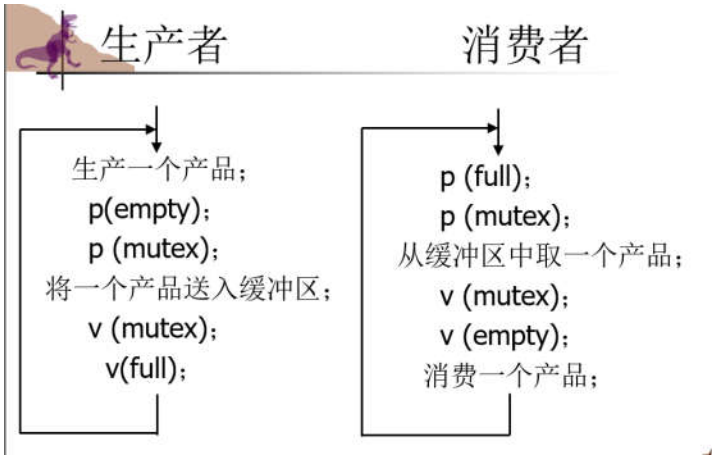
```
pool: A
readyQueue:consumer
currently running: producer, pc:0
producing a product B
1
already put into ready queue
stop the producer from running? y/n n
-------------------------------------


pool: A
readyQueue:consumer
currently running: producer, pc:1
producer is excuting p(s1) 8
0
already put into ready queue
stop the producer from running? y/n n
-------------------------------------


pool: A
readyQueue:producer
currently running: consumer, pc:1
consumer got a product A from the pool
0
already put into ready queue
.
-------------------------------------


pool: A
readyQueue:consumer
currently running: producer, pc:2
putting product B into the pool
1
already put into ready queue
stop the producer from running? y/n n
-------------------------------------


pool: A B
readyQueue:consumer
currently running: producer, pc:3
producer is excuting v(s2) 1
```

```
1
already put into ready queue
stop the producer from running? y/n y
-------------------------------------


pool: A B
readyQueue:
currently running: consumer, pc:2
consumer is excuting v(s1) 9
0
already put into ready queue
.
-------------------------------------


pool: A B
readyQueue:
currently running: consumer, pc:3
consumer is consuming product A
0
already put into ready queue
.
-------------------------------------


pool: 0 B
readyQueue:
currently running: consumer, pc:4
consumer is excuting goto(0)
0
already put into ready queue
.
-------------------------------------


pool: 0 B
readyQueue:
currently running: consumer, pc:0
consumer is excuting p(s2) 0
0
already put into ready queue
.
-------------------------------------
```

```
156.
157.
158. pool: 0 B
159. readyQueue:
160. currently running: consumer, pc:1
161. consumer got a product B from the pool
162. 0
163. already put into ready queue
164. .
165. ------------------------------------
166.
167.
168. pool: 0 B
169. readyQueue:
170. currently running: consumer, pc:2
171. consumer is excuting v(s1) 10
172. 0
173. already put into ready queue
174. .
175. ------------------------------------
176.
177.
178. pool: 0 B
179. readyQueue:
180. currently running: consumer, pc:3
181. consumer is consuming product B
182. 0
183. already put into ready queue
184. .
185. ------------------------------------
186.
187.
188. pool: 0 0
189. readyQueue:
190. currently running: consumer, pc:4
191. consumer is excuting goto(0)
192. 0
193. already put into ready queue
194. .
195. ------------------------------------
196.
197.
198. pool: 0 0
199. readyQueue:
```

```
currently running: consumer, pc:0
consumer is excuting p(s2) -1
reason for waiting: s2
consumer is waiting
.
```

5.4 实验总结

- 生产者-消费者：缓冲池中的每个缓冲区可以存放一个产品，生产者进程不断生产产品并将产品放入缓冲池中，消费者进程不断从缓冲池内取出产品并消费。
  - 设置两个同步信号量 empty、full，其初值分别为 n、0。
  - 有界缓冲池是一个临界资源，还需要设置一个互斥信号量 mutex，其初值为 1。
  - 

  - 两者的两个 p 操作均不能颠倒，否则可能造成死锁

参考文献

教师评语评分

评语：

评分：

评阅人：

年　　月　　日

（备注：对该实验报告给予优点和不足的评价，并给出百分之评分。）