内存管理设计方案报告

项目要求

初始态下存在一请求序列，请分别用首次适应算法和最佳适应算法进行内存块的分配和回收，并显示出每次分配和回收后的空闲分区链的情况。

使用说明

1、运行程序，输入要模拟的内存大小，进入主界面

2、程序左上角为编辑进程调度队列模块，可采取随机生成调度队列或自定义调度队列的方式。生成随机队列会清空以前队列里的内容，可在生成随机队列后再添加要创建调入或释放的进程。

可点击主界面右侧清空进程队列按钮清空调度队列。

3、 调度算法可选首次适应算法或最佳适应算法。点击左侧执行单步或执行全部按钮开始执行。

4、 点击重置按钮，程序恢复至初始状态

本程序运行规则

1、调度队列长度最大为 30。

2、可创建的进程大小范围为：总内存容量的 2.5% 至 300K。

3、生成随机队列时先让内存充满至大于 70%，再进行随机操作

4、随机状态下创建新调入进程的概率为（剩余内存空间 / 总内存空间 * 1.2）

5、执行调入或释放操作后（并非指添加进调度队列）的进程在重置系统之前不会再被重新创建。

6、单步执行时可往进程队列添加或释放进程

7、执行完整个调度队列后，调度队列才会被清空。

8、内存占用表与空闲表的数据结构为双向链表

9、为更好地呈现空闲表与占用表的信息，两表内容均按内存地址顺序排列。

代码实现

本程序用 C#语言通过 winform 框架来实现。控件操作均来自 winform 框架。

主要源代码在 MainForm.cs，MainForm.Designer.cs，StartForm.cs，StartForm.Designer.cs 中。

类

StartForm 类：程序开始界面，主要包含内存大小输入框与开始按钮，获取输入的内存大小信息并创建程序主界面

MainForm 类：程序主界面，包含所有的内存调度的 UI 及交互，运行逻辑等，下方将详细介绍

Process 类：进程类，存储创建的进程的信息，如编号，大小，在内存中的位置及其 UI 模块

ProcessAction 类：进程操作类，存储进程的行为（调入/释放）及对应的进程

FreeMemList 类：内存空闲区表，存储内存空闲区表的表头

UsedMemList 类：内存占用区表，存储内存占用区的表的表头

MemBlock 类：内存块表，作为空闲区表占用区表两个双向链表的结点，存储在内存中的位置及其大小与他的上一个或下一个内存块的信息

MainForm 类主要属性

```
private static int maxProcessListSize = 30;
private static int memUISize = 1000;                                    //UI界面内存大小
private readonly int memSize = 1000;                                    //内存总容量
private int freeMem;                                                    //内存剩余容量
private int falseNum = 0;                                               //存储调度时调入进程失败次数
private int nextProcessNo = 1;                                          //下一个调入进程的编号
private int nextProcessNoInCreate = 1;                                  //编辑列表时创建下一个调入进程的编号
private bool algorithm = true;                                          //算法的选择，true为FF，False为BF
private List<Process> processes = new List<Process>();                  //已创建进程的列表
private List<ProcessAction> processList = new List<ProcessAction>();    //进程调度队列
private FreeMemList freeMemList;                                        //内存空闲区表
private UsedMemList usedMemList;                                        //内存占用区表
```

主要运行逻辑

生成随机调度列表

```
private void CreateRandomListButton_Click(object sender, EventArgs e)
{
    //检查数据规范性
    if (string.IsNullOrEmpty(getRandomNumTextBox.Text)) MessageBox.Show("数据未填写", "提示");
    else if (int.TryParse(getRandomNumTextBox.Text, out int randomNum) && randomNum >= 3 && randomNum <= 30)
    {
        //初始化（清除）调度进程列表
        ClearProcessList();
        //获取剩余空间大小
        int freeMemInRd = freeMem;
        //储存可释放的进程的编号
        List<int> rdCreatedProcess = new List<int>();
        //随机数引擎
        Random rd = new Random();
        bool isRdAction = false;
        //将目前在内存中的进程的编号添加进可释放的进程的编号的列表
        foreach (var process in processes)
        {
            if (process.processStarting != -1) rdCreatedProcess.Add(process.processNo);
        }
        for (int i = 0; i < randomNum; i++)
        {
            bool rdAction;
            //随机状态下创建新调入进程的概率为（剩余内存空间 / 总内存空间 * 1.2）
            //生成随机队列时先让内存充满至大于70%，再进行随机操作
            if (isRdAction) rdAction = rd.Next(0, memSize) < freeMemInRd * 1.2;
            else rdAction = true;
            if (rdAction)
            {
                //创建新进程，加入创建进程列表
                int rdProcessSize = rd.Next(25 * memSize / 10000 + 1, 30) * 10;

                Process process = new Process(nextProcessNoInCreate, rdProcessSize);
                ProcessAction processAction = new ProcessAction(process, rdAction);
                processes.Add(process);
                //添加进调度进程列表，添加进可释放进程列表，更新选择调出进程列表栏信息
                processList.Add(processAction);
                rdCreatedProcess.Add(nextProcessNoInCreate);
                chooseReleaseNoListBox.Items.Add(nextProcessNoInCreate);
                //更新下一个被创建的进程的编号，更新（随机创建状态下）剩余内存容量
                nextProcessNoInCreate++;
                freeMemInRd -= rdProcessSize;

                processListTextBox.AppendText("进程" + process.processNo + " 申请" + rdProcessSize + "K\r\n");

                if (!isRdAction) if (freeMemInRd < memSize * 0.3) isRdAction = true;
            }
            else
            {
                //从可释放的进程中随机选择一个进程进行释放操作
                int rdReleaseProcess = rd.Next(0, rdCreatedProcess.Count);

                Process process = processes[rdCreatedProcess[rdReleaseProcess] - 1];
                ProcessAction processAction = new ProcessAction(process, rdAction);

                //添加进调度进程列表，从可释放进程列表中剔除，更新选择调出进程列表栏信息，更新（随机创建状态下）剩余内存容量
                processList.Add(processAction);
                rdCreatedProcess.RemoveAt(rdReleaseProcess);
                chooseReleaseNoListBox.Items.Remove(process.processNo);
                freeMemInRd += process.size;

                processListTextBox.AppendText("进程" + process.processNo + " 释放\r\n");
            }
        }
    }
    else
    {
        MessageBox.Show("需要填入3~30之间的数字", "提示");
        getRandomNumTextBox.Text = string.Empty;
    }
```

自动执行函数

```
//自动执行函数
private async void AutoRun()
{
    //检查调度列表是否为空
    if (processList.Count == 0)
    {
        MessageBox.Show("进程列表为空", "提示");
        return;
    }
    //屏蔽按钮交互响应
    SetButton(false);
    //顺序执行
    foreach (var processAction in processList)
    {
        await Task.Delay(1000);
        if (processAction.op)
        {
            //记录下一个未调度的进程序号
            nextProcessNo++;
            AddProcess(processAction.process);
        }
        else ReleaseProcess(processAction.process);
    }
    MessageBox.Show("执行完成\r\n\r\n进程调入失败次数: " + falseNum, "提示");
    //清空调度队列
    ClearProcessList();
    //恢复按钮交互响应
    SetButton(true);
}
```

调入进程操作

```
//调入进程
private void AddProcess(Process process)
{
    //获取调入位置，如果成功则执行调入操作，不能便统计失败信息
    if (GetPositon(out int position, process.size))
    {
        //新建TextBoxUI控件
        process.memBlock = new TextBox();
        process.memBlock.Multiline = true;
        process.memBlock.ReadOnly = true;
        process.memBlock.BackColor = Color.LightSkyBlue;
        process.memBlock.Text = "\r\n进\r\n程\r\n" + process.processNo;
        process.memBlock.Font = new Font("微软雅黑", 9.75F, FontStyle.Regular, GraphicsUnit.Point, 134);
        process.memBlock.Size = new Size((process.size * memUISize / memSize), 100);
        process.memBlock.Location = new Point(10 + position * memUISize / memSize, 25);
        memPoolGroupBox.Controls.Add(process.memBlock);
        process.memBlock.BringToFront();
        process.memBlock.Refresh();

        //记录获取的进程位置，更新内存剩余空间容量，更新占用表
        process.processStarting = position;
        freeMem -= process.size;
        UpdateUsedMemList_AddProcess(position, process.size);
        messageTextBox.AppendText("进程" + process.processNo + " 调度成功，插入位置: " + position + "K\r\n");
    }
    else
    {
        falseNum++;
        messageTextBox.AppendText("无插入内存空间，进程" + process.processNo + "调度失败\r\n");
    }
}
```

获取调入进程的位置

```
//获取调入位置，若无调入空间，返回false
private bool GetPositon(out int p, int size)
{
    p = -1;
    MemBlock freeMemBlock = null;
    //首次适应算法
    if (algorithm)
    {
        freeMemBlock = freeMemList.first;
        while (freeMemBlock != null)
        {
            if (freeMemBlock.blockSize >= size)
            {
                p = freeMemBlock.starting;
                UpdateFreeMemList_AddProcess(freeMemBlock, size);
                break;
            }
            freeMemBlock = freeMemBlock.next;
        }
    }
    //最佳适应算法
    else
    {
        int maxSize = 0;
        MemBlock block = freeMemList.first;
        //寻找最大且足够大的块
        while(block != null)
        {
            if (block.blockSize >= size && block.blockSize > maxSize)
            {
                freeMemBlock = block;
                maxSize = block.blockSize;
            }
            block = block.next;
        }
        if (freeMemBlock != null)
        {
            p = freeMemBlock.starting;
            UpdateFreeMemList_AddProcess(freeMemBlock, size);
        }
    }
    return (p >= 0);
}
```

释放进程操作

```
//释放进程
private void ReleaseProcess(Process process)
{
    //若该进程已被调入内存，则进行释放操作，否则输出失败信息
    if (process.processStarting != -1)
    {
        //删除UI控件，更新占用表，空闲表信息，更新内存剩余空间大小
        process.memBlock.Dispose();
        UpdateFreeMemList_Release(process.processStarting, process.size);
        UpdateUsedMemList_Release(process.processStarting, process.size);
        freeMem += process.size;
        process.processStarting = -1;

        messageTextBox.AppendText("进程" + process.processNo + " 已释放\r\n");
    }
    else messageTextBox.AppendText("进程" + process.processNo + " 并未在内存中，释放失败\r\n");
}
```

调入操作后，更新两表信息

```
//调入进程后更新空闲区表信息
private void UpdateFreeMemList_AddProcess(MemBlock freeMemBlock, int size)
{
    //若无需删除整个空闲块结点，更改起点与大小
    if (freeMemBlock.blockSize != size)
    {
        freeMemBlock.starting += size;
        freeMemBlock.blockSize -= size;
    }
    //若要删除空闲区表头结点，将下一个结点当做头结点
    else if (freeMemBlock == freeMemList.first)
    {
        freeMemList.first = freeMemBlock.next;
        if (freeMemList.first != null) freeMemList.first.last = null;
    }
    //删除某个空闲块结点
    else
    {
        MemBlock lastBlock = freeMemBlock.last;
        lastBlock.next = freeMemBlock.next;
        if (lastBlock.next != null) lastBlock.next.last = lastBlock;
    }
    //更新空闲区表信息
    PrintFreeMemList();
}
```

```
//调入进程后更新占用区表信息
private void UpdateUsedMemList_AddProcess(int start, int size)
{
    //若占用表为空，创建头结点
    if (usedMemList.first == null) usedMemList.first = new MemBlock(start, size);
    else
    {
        MemBlock block = usedMemList.first;
        //若新表项在表头前，更新头结点
        if (start < usedMemList.first.starting)
        {
            MemBlock newBlock = new MemBlock(start, size);
            newBlock.next = block;
            block.last = newBlock;
            usedMemList.first = newBlock;
            block = newBlock;
        }
        //新表项紧挨某项后方（增大某项体积）
        else
        {
            while (block != null && block.starting + block.blockSize != start) block = block.next;
            block.blockSize += size;
        }

        //检测是否要与下一个结点项合并
        if (block.next != null && block.starting + block.blockSize == block.next.starting)
        {
            block.blockSize += block.next.blockSize;
            block.next = block.next.next;
            if (block.next != null) block.next.last = block;
        }
    }
    //更新占用表信息
    PrintUsedMemList();
}
```

释放操作后，更新两表信息

```
//释放进程后更新空闲表
private void UpdateFreeMemList_Release(int start, int size)
{
    //若空闲表为空，创建头结点
    if (freeMemList.first == null) freeMemList.first = new MemBlock(start, size);
    else
    {
        //储存已修改的结点及其上一项，下一项的结点信息，便于执行合并操作
        MemBlock block = new MemBlock(start, size);
        MemBlock lastBlock = freeMemList.first;
        MemBlock nextBlock = null;
        //若新表项添加在头结点前，更改头结点
        if (start < lastBlock.starting)
        {
            block.next = lastBlock;
            lastBlock.last = block;
            freeMemList.first = block;
            lastBlock = block;
            block = lastBlock.next;
            nextBlock = block.next;
        }
        else
        {
            //寻找插入的位置（插在哪一结点后方）
            while (lastBlock.next != null && lastBlock.next.starting < start) lastBlock = lastBlock.next;
            //若该结点不是尾节点，连接下一项
            if(lastBlock.next != null)
            {
                nextBlock = lastBlock.next;
                block.next = nextBlock;
                nextBlock.last = block;
            }
            //连接上一项
            lastBlock.next = block;
            block.last = lastBlock;
        }
```

```
        //合并操作
        //检查是否要与上一项合并
        if(lastBlock != null)
        {
            if(lastBlock.starting + lastBlock.blockSize == block.starting)
            {
                lastBlock.blockSize += block.blockSize;
                lastBlock.next = nextBlock;
                if (nextBlock != null) nextBlock.last = lastBlock;
                block = lastBlock;
            }
        }
        //检查是否要与下一项合并
        if(nextBlock != null)
        {
            if(block.starting + block.blockSize == nextBlock.starting)
            {
                block.blockSize += nextBlock.blockSize;
                block.next = nextBlock.next;
                if (block.next != null) block.next.last = block;
            }
        }
    }
    //更新空闲表信息
    PrintFreeMemList();
```

```
//释放进程后更新占用表
private void UpdateUsedMemList_Release(int start, int size)
{
    //若删去整个头结点，修改新的头结点
    if (usedMemList.first.starting == start && usedMemList.first.blockSize == size) usedMemList.first = usedMemList.first.next;
    else
    {
        MemBlock block = usedMemList.first;
        while (!(block.starting <= start && block.starting + block.blockSize > start)) block = block.next;
        //若要删去某个节点
        if(start == block.starting && size == block.blockSize)
        {
            block.last.next = block.next;
            if(block.next != null) block.next.last = block.last;
        }
        //若要删去某个节点前半部分，更新起址，大小
        else if (start == block.starting)
        {
            block.starting += size;
            block.blockSize -= size;
        }
        //若要删去某个节点后半部分，更新大小
        else if (start + size == block.starting + block.blockSize)
        {
            block.blockSize -= size;
        }
        //删去某个节点的中间部分，创建新的结点并连接
        else
        {
            MemBlock newBlock = new MemBlock(start + size, block.starting + block.blockSize - start - size);
            block.blockSize = start - block.starting;
            newBlock.next = block.next;
            if (newBlock.next != null) newBlock.next.last = newBlock;
            block.next = newBlock;
            newBlock.last = block;
        }
    }
    PrintUsedMemList();
}
```

清除调度进程列表操作

```
//清除调度进程列表
private void ClearProcessList()
{
    //清除已创建但未调度的进程
    processes.RemoveRange(nextProcessNo - 1, nextProcessNoInCreate - nextProcessNo);
    //清除调度进程列表
    processList.Clear();
    //清除调度进程列表信息
    processListTextBox.Clear();
    //更新选择释放进程列表栏信息
    //先清空
    //添加仍处在内存中的进程
    chooseReleaseNoListBox.Items.Clear();
    for(int i = 0; i < nextProcessNo - 1; i++)
    {
        if (processes[i].processStarting != -1) chooseReleaseNoListBox.Items.Add(processes[i].processNo);
    }
    //更新编辑列表时创建下一个调入进程的编号
    nextProcessNoInCreate = nextProcessNo;
    //重置记录失败次数为0
    falseNum = 0;
}
```

重置系统操作

```
//重置系统
private void Reset()
{
    //重置下一个要调入的进程编号
    nextProcessNo = 1;
    nextProcessNoInCreate = 1;
    //重置空闲区表占用区表
    freeMemList.first = new MemBlock(0, memSize);
    usedMemList.first = null;
    PrintFreeMemList();
    PrintUsedMemList();
    //清除内存
    freeMem = memSize;
    foreach (var process in processes)
    {
        if (process.memBlock != null) process.memBlock.Dispose();
    }
    //清除已创建的所有进程
    processes.Clear();
    //清除调度进程列表
    ClearProcessList();
    //清除运行信息
    messageTextBox.Clear();
}
```