DYNAMIC PARTITION SIMULATOR

课程名： 操作系统

授课老师：王冬青

设计者： 杜书杨

学号：1452981

设计时间：2017年5月18日 – 2017年5月22日

1. 项目描述及需求

基本任务：

假设初始态下，可用内存空间为640K，并有下列请求序列，请分别用首次适应算法和最佳适应算法进程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来。

具体目标：

分别实现首次适应算法和最佳适应算法对进程内存块的分配和回收，并通过可视化图形界面将内存空间的实时情况展现出来。在这个模拟系统内，用户可通过输入进程所占空间的大小来添加新进程，通过输入现有进程的序号来删除现有进程。当内存不满足添加条件时（内存溢出），系统会弹窗提示无法添加。

2. 解决方案

开发环境：

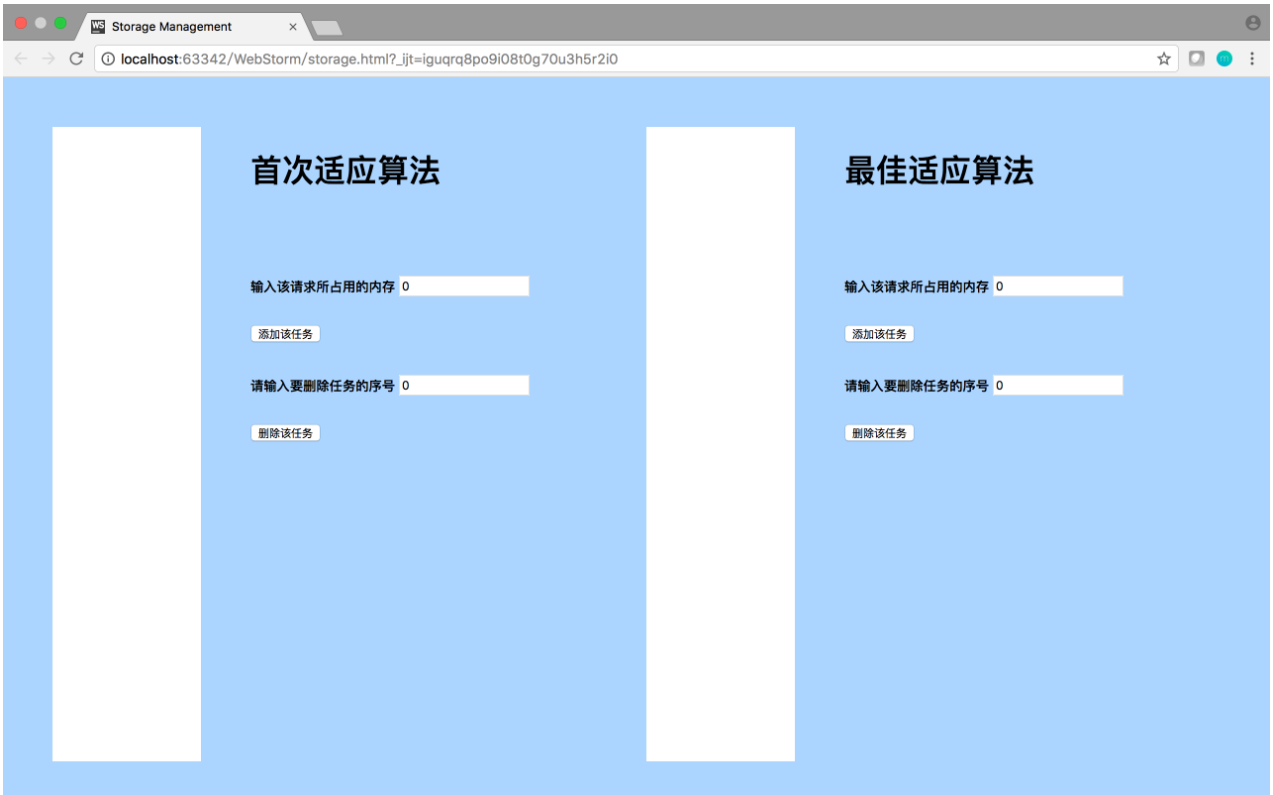
开发工具：WebStorm

开发语言：HTML5 & JavaScript

辅助工具：iWork Pages 文档

成品：点击html文件运行

项目大纲：



基本界面如上图所示，左侧为首次适应算法的操作及显示部分，右侧是最佳适应算法的操作及显示部分。两侧的操作互不干扰，可同时进行。内存用长为640px的矩形来表示，每1px长度代表1K内存。

界面部分使用HTML5编写，白色的内存空间为自己编写的div块，其余文本和按钮均为自带属性。

两种适配算法介绍：

首次适应算法从空闲分区表的第一个表目起查找该表，把最先能够满足要求的空闲区分配给作业，这种方法目的在于减少查找时间。为适应这种算法，空闲分区表(空闲区链)中的空闲分区要按地址由低到高进行排序。该算法优先使用低址部分空闲区，在低址空间造成许多小的空闲区，在高地址空间保留大的空闲区。

最佳适配算法从全部空闲区中找出能满足作业要求的、且大小最小的空闲分区，这种方法能使碎片尽量小。为适应此算法，空闲分区表（空闲区链）中的空闲分区要按从小到大进行排序，自表头开始查找到第一个满足要求的自由分区分配。该算法保留大的空闲区，但造成许多小的空闲区。

添加进程：

```
// 首次适配算法
current_1 = 0;       // 目前已经添加的进程数
var space_1 = new Array();  // 当前空余空间的列表,数组中的值为各空余空间的大小
space_1[0] = 640;
var start_1 = new Array();  // 当前各个空余的空间的起始位置
start_1[0] = 0;
var list_1 = new Array();  // 当前的进程列表,添加的第一个进程为list_1[1]
list_1[0] = 0;

// 最佳适配算法
current_2 = 0;       // 目前已经添加的进程数
var space_2 = new Array();  // 当前空余空间的列表,数组中的值为各空余空间的大小
space_2[0] = 640;
var start_2 = new Array();  // 当前各个空余的空间的起始位置
start_2[0] = 0;
var list_2 = new Array();  // 当前的进程列表,添加的第一个进程为list_2[1]
list_2[0] = 0;

// 首次适配算法
var s_1 = document.getElementById("size_1");       // 输入添加进程的大小
var n_1 = document.getElementById("free_number_1");    // 输入删除进程的序号

// 最佳适配算法
var s_2 = document.getElementById("size_2");       // 输入添加进程的大小
var n_2 = document.getElementById("free_number_2");    // 输入删除进程的序号
```

首先针对两种算法模式建立三个数组，分别代表各个进程（list），当前内存中各空余空间的起始地址（start）和当前各个空余空间的大小（space）。

两种算法均通过输入要添加的进程所占的内存空间大小，然后点击按钮来添加进程。之后程序将根据不同的适配算法来确定应该添加的位置，通过创立一个div块来代表进程，

并显示进程的序号和所占内存，并更改相应数组中的内容下图是代码展示。注：两种算法只有select函数不一样，所以这里只展示第二种算法的select函数。考虑过合并函数的问题，但涉及不同数组的更改数据很难协调遂放弃。

```
// 首次适配算法添加新进程
function add_1()
{
    var size= parseInt(s_1.value);  // 获取新进程所需内存空间
    if(size <= 0)                   // 提示输入的值不符合要求
    {
        alert("请输入正确的值!");
        return;
    }
    var start = select_1(size);  // 调用select函数计算应该插入哪个空余空间
    if(start === -1)             // 如果没有可以添加的空间,提示内存溢出
    {
        alert("内存溢出,无法添加!");
        return;
    }
    var t = start + 50;                    // 添加div块,表示新进程
    var mission = document.createElement("div");
    if(size < 20)
        mission.style.fontSize="5px";
    mission.style.textAlign="center";
    mission.style.color="white";
    mission.style.position="absolute";
    mission.style.border="1px solid white";
    mission.style.background="#0000FF";
    mission.style.width="148px";
    mission.style.height=size-2+"px";
    mission.style.left="50px";
    mission.style.top=t+"px";
    document.body.appendChild(mission);
    list_1[++current_1] = mission;                // 新进程加入列表中
    mission.innerHTML=current_1+" - "+size+"K";          // 显示进程序号和所占空间
}
```

```
// 首次适配,选择空间插入
function select_1(size)
{
    var is_ok=false;             // 是否有合适的空间
    var number=0;                // 第一个合适的空间在列表中的序号
    for(var i=0; i<space_1.length; i++)     // 搜寻合适的空间
    {
        if (size <= space_1[i])       // 得到合适的空间,因为是首次适配,所以记下空间序号后直接跳出循环
        {
            is_ok = true;
            number = i;
            break;
        }
    }
    if(is_ok === false)          // 没有合适的空间,返回-1
        return -1;
    else if(size === space_1[number])      // 空间大小正好等于新进程大小,调用delete函数去除这个空间,并返回该空间起始位置值
    {
        var re = start_1[number];
        delete_space_1(number);
        return re;
    }
    else                         // 空间大于进程大小,则修改该空间的起始位置和大小,并返回原先的起始位置值
    {
        space_1[number]=space_1[number]-size;
        var r = start_1[number];
        start_1[number] = start_1[number] + size;
        return r;
```

```
// 最佳适配,选择空间插入
function select_2(size)
{
    var content = 0;
    var is_ok=false;            // 是否有合适的空间
    var number=0;               // 最佳空间在列表中的序号
    for(var i=0; i<space_2.length; i++)       // 搜寻最佳合适空间
    {
        if (size <= space_2[i])           // 因为是最佳适配,所以要搜寻所有的合适空间
        {
            if(is_ok === false)           // 找到第一个合适空间
            {
                is_ok = true;
                number = i;
            }
            else if(space_2[i] < space_2[number])       // 若后面找到更加合适的空间,则改变最佳空间的序号
                number = i;
        }
    }
    if(is_ok === false)         // 没有合适的空间,返回-1
        return -1;
    else if(size === space_2[number])       // 空间大小正好等于新进程大小,调用delete函数去除这个空间,并返回该空间起始位置值
    {
        var re = start_2[number];
        delete_space_2(number);
        return re;
    }
    else                        // 空间大于进程大小,则修改该空间的起始位置和大小,并返回原先的起始位置值
    {
        space_2[number]=space_2[number]-size;
        var r = start_2[number];
        start_2[number] = start_2[number] + size;
        return r;
    }
}
```

注意：在选择空间插入时有一种情况是进程的内存大小和所选择的空间的内存大小正好相同，此时要调用一个delete_space函数来删除掉这个空间，代码如下，两种算法都一样，所以仅展示其一。

```
// 去除一个空余空间
function delete_space_1(number)
{
    if(space_1.length === 1)      // 如果只有一个空间
    {
        space_1[0] = 0;
        start_1[0] = -1;
    }
    else
    {
        for(var i=number; i<space_1.length-1; i++)    // 去除这个空间后,后面的空间序号改变
        {
            space_1[i] = space_1[i+1];
            start_1[i] = start_1[i+1];
        }
        space_1.splice(space_1.length-1, 1);
        start_1.splice(start_1.length-1, 1);
    }
}
```

删除进程：

两种算法删除进程的函数逻辑均一样，再删除进程时，获取其占用空间大小，在内存中的起始和终止位置，然后调用create_space函数来判断新生成的空间是否要与现有空间

合并，并针对不同情况对空间的起始和大小数组做出相应的操作，同时也要考虑极端情况，例如删除后内存中没有进程的情况。下图为代码展示，因为两种算法逻辑相同，所以仅展示其一。

```
// 删除内存中现有的进程
function remove_mission(type)
{
    if(type === 1)          // 删除首次适配中的进程
    {
        var number = parseInt(n_1.value);          // 获取要删除的进程的序号
        var x = list_1[number].offsetTop-50;        // 获取要删除的进程在内存中的起始位置
        var newSpace = list_1[number].offsetHeight;   // 获取要删除的进程的大小
        var y = x + newSpace;                        // 获取要删除的进程在内存中的终止位置
        create_space_1(x, y, newSpace);              // 删除该进程以创造新的空间
        document.body.removeChild(list_1[number]);      // 在页面上删除进程的显示
    }
    else                    // 删除最佳适配中的进程
    {
        var number = parseInt(n_2.value);          // 获取要删除的进程的序号
        var x = list_2[number].offsetTop-50;        // 获取要删除的进程在内存中的起始位置
        var newSpace = list_2[number].offsetHeight;   // 获取要删除的进程的大小
        var y = x + newSpace;                        // 获取要删除的进程在内存中的终止位置
        create_space_2(x, y, newSpace);              // 删除该进程以创造新的空间
        document.body.removeChild(list_2[number]);      // 在页面上删除进程的显示
    }
}
```

```
// 在首次适配的内存中删除的进程处产生新的空间
function create_space_1(x, y, newSpace)
{
    var is_merge = false;                   // 新空间是否需要和现有空间合并
    for(var i=0; i<start_1.length; i++)
    {
        if(start_1[i]+space_1[i] === x)          // 如果新空间要与前面的进程合并
        {
            is_merge = true;
            space_1[i] = space_1[i] + newSpace;
        }
        else if(y === start_1[i])          // 如果新空间要与后面的进程合并
        {
            if(is_merge === true)          // 如果新空间已经与前面的进程合并,又要与后面的进程合并
            {
                space_1[i-1] = space_1[i-1] + space_1[i];
                delete_space_1(i);
            }
            else                           //  如果新空间只与后面的进程合并
            {
                is_merge = true;
                space_1[i] = space_1[i] + newSpace;
                start_1[i] = x;
            }
        }
    }
    if(is_merge === false)              // 如果新空间不需与现有空间合并,新建一个空间
    {
        var is_last = true;             // 新空间是否为内存上的最后一个空间
        for(var i=0; i<start_1.length; i++)
        {
            if(start_1[i] > y)           // 新空间不是最后一个空间,新建空间后后面原有的空间序号+1
            {
                is_last = false;
                for(var j=start_1.length; j>i; j--)
                {
                    start_1[j] = start_1[j-1];
                    space_1[j] = space_1[j-1];
                }
```

```
                start_1[i] = x;
                space_1[i] = newSpace;
                break;
            }
        }
        if(is_last === true)            // 新空间为内存上的最后一个空间
        {
            if(start_1[0]===-1)         // 删除前内存已满的情况
            {
                start_1[0] = x;
                space_1[0] = newSpace;
            }
            else                        // 删除前内存未满的情况
            {
                var old_length = start_1.length;
                start_1[old_length] = x;
                space_1[old_length] = newSpace;
            }
        }
    }
}
```

3. 工作流程

5.18-5.19　HTML写界面，JS写基本的外部回调函数

5.19-5.20　首次适配算法的添加函数完成

5.21-5.22　首次适配的删除函数完成，两种算法的相同内容进行移植，最佳适配完成

5.23-5.25　项目文档

5. 备注&小问题

1. 备注：有了上次的经验，这次我在写代码时仔细考虑了冗余的问题，但最后自己还是觉得有很多冗余无法处理，主要还是自己在开始定义的全局变量作为实参和函数的形参的问题，因为难以协调而只能将逻辑相同的一个函数写成两个函数。因此觉得自己对JS的使用还是不熟练，只能说勉强入门，还任重道远。

注：这次主要是chrome中进程的内存块的显示问题，如果添加的进程的内存大小小于20，显示出的进程的序号和内存大小的字就会显示不全。我试图加入判断来缩小特定情况的字体的大小，但最后发现好像chrome是有默认最小值的，改过后也只能满足大于15的情况，内存小于15的进程依然不能把字显示全。但safari中就没有这种情况，改过之后可以满足6以上的情况。还有因为我的进程内存div的两边框加起来是2px，所以如果加入2K大小的进程，就只会显示白边，而没有蓝色的实体，所以颜色上不可见。而如果加入1K大小的则会显示出大概20px的空间，但占位依然是1px，这也是我有点不太理解的地方。不过这些问题都不影响测试结果，只是显示的时候会造成困惑。

最后说一句老师助教辛苦了！

有问题可以邮箱联系我：dushuyang@126.com