学号：2151299

姓名：苏家铭

学院：软件学院

专业：软件工程

班号：42036902

指导老师：张惠娟

完成日期：2023年5月19日

内存管理——调页管理

[Operation System Project 2nd]

[操作系统项目2]

[2023年5月19日]

目录

[1.项目介绍 2](#_Toc14067)

[1.1项目目的 2](#_Toc32461)

[1.2项目需求 3](#_Toc22302)

[2.开发环境 3](#_Toc27386)

[3.程序运行 3](#_Toc5035)

[3.1直接运行 3](#_Toc10306)

[4.功能实现 4](#_Toc238)

[4.1前端界面 4](#_Toc30448)

[4.2选择算法 10](#_Toc8136)

[4.3开始按钮被按下 11](#_Toc9175)

[4.4进度条 11](#_Toc32344)

[4.5拖动条 11](#_Toc21370)

[4.6内存块页显示 12](#_Toc7191)

[4.7结果展示 13](#_Toc18538)

[5.算法 14](#_Toc22111)

[5.1全局变量 14](#_Toc17248)

[5.2功能逻辑 14](#_Toc2223)

[5.2.1 延时输出 14](#_Toc24830)

[5.3.2执行模式 14](#_Toc14286)

[5.3.3置换和请求调页算法 15](#_Toc19155)

[5.3.4 产生随机指令 16](#_Toc26451)

# 1.项目介绍

## 1.1项目目的

1.1.1理解页面、页表、地址转换

1.1.2体会页面置换过程

1.1.3加深对请求调页系统的原理和实现过程的理解。

## 1.2项目需求

* 基本任务

假设每个页面可存放10条指令，分配给一个作业的内存块为4。模拟一个作业的执行过程，该作业有320条指令，即它的地址空间为32页，目前所有页还没有调入内存。

* 模拟过程

在模拟过程中，如果所访问指令在内存中，则显示其物理地址，并转到下一条指令；如果没有在内存中，则发生缺页，此时需要记录缺页次数，并将其调入内存。如果4个内存块中已装入作业，则需进行页面置换。

所有320条指令执行完成后，计算并显示作业执行过程中发生的缺页率。

置换算法可以选用FIFO或者LRU算法

作业中指令访问次序可以按照下面原则形成：

50%的指令是顺序执行的，25%是均匀分布在前地址部分，25％是均匀分布在后地址部分。

# 2.开发环境

* 开发环境：Windows 11
* 开发软件：Visual Studio code
* 开发语言：Html+JavaScript+css

# 3.程序运行

## 3.1编译运行

1. 进入项目目录
2. 通过浏览器方式打开html文件

# 4.功能实现

## 4.1前端界面



通过html+css代码实现：

<!DOCTYPE html>

<html lang="zh-CN">

  <head>

    <!-- 相关约束 -->

    <meta charset="utf-8">

    <meta name="author" content="苏家铭 215199">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <meta http-equiv='cache-control' content="no-cache, must-revalidate, post-check=0, pre-check=0">

    <meta http-equiv='expires' content='0'>

    <meta http-equiv='pragma' content='no-cache'>

    <meta http-equiv="cache-control" content="max-age=0" />

    <meta http-equiv="cache-control" content="no-cache" />

    <meta http-equiv="expires" content="0" />

    <meta http-equiv="expires" content="Tue, 01 Jan 1980 1:00:00 GMT" />

    <meta http-equiv="pragma" content="no-cache" />

    <title>内存管理</title>

    <!-- 引入样式 -->

    <link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css">

    <link href="https://cdn.bootcss.com/font-awesome/5.13.0/css/all.css" rel="stylesheet">

    <script src="http://ajax.aspnetcdn.com/ajax/jquery/jquery-1.9.0.js"></script>

    <script src="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/js/bootstrap.min.js"></script>

    <script src="http://ajax.aspnetcdn.com/ajax/knockout/knockout-2.2.1.js"></script>

    <!-- 引入CSS -->

    <link rel="stylesheet" type="text/css" href="css/main.css">

  </head>

  <body>

    <!-- 引入js -->

    <script src="js/jquery-3.5.1.js"></script>

    <script type="text/javascript" src="js/Mymain.js"></script>

    <div class="container">

        <!-- 页眉 -->

        <div class="w3-container" style="margin-top:20px;background-color: aliceblue;">

            <h3><b>操作系统内存管理——请求调页存储管理方式模拟</b></h3>

        </div>

        <div>

          <h3 style="margin-top:10px;background-color: rgb(159, 197, 232);">同济大学 软件学院</h3>

          <h3 style="margin-top:10px;background-color: rgb(159, 189, 216);">2151299 苏家铭</h3>

        </div>

        <div class="container2">

          <!--左边的框 参数信息-->

          <section class="box">

            <h2 style="font-weight: bold;">相关参数</h2><br>

            <h3>指令总数：<span id="overall\_instructions">320</span></h3>

            <h3>内存块数：<span id="overall\_memory\_blocks">4</span></h3>

            <h4>每页指令数：<span id="instruction\_per\_page">10</span></h4>

            <h4>顺序执行：50%</h4>

            <h4>向前跳转：25%</h4>

            <h4>向后跳转：25%</h4>

          </section>

         <!--中间的框 选择算法-->

          <section class="box\_algorithm">

            <h3 style="font-weight: bold;">选择算法</h3><br>

              <div class="container3">

                <input type="radio" name="algorithm" id="FIFORadio" value="FIFO" checked="checked">

              <button class="w3-button w3-pale-green" type="submit" style="border-radius:30px; width: 300px;height: 40px;">

                <label for="FIFORadio" style="font-weight:bold;font-size: large;">先进先出算法(FIFO)</label>

              </button>

              </div>

              <br><br><br>

              <div class="container3">

                <input type="radio" name="algorithm" id="LRURadio" value="LRU">

              <button class="w3-button w3-pale-green" type="submit" style="border-radius:30px; width: 300px;height: 40px;">

                <label for="LRURadio" style="font-weight:bold;font-size: large;">最近最久未使用算法(LRU)</label>

              </button>

              </div><br><br>

          </section>

          <!--实验结果动态更新-->

          <section class="box">

            <h2 style="font-weight: bold;">实验结果</h2><br>

            <h3><strong>①FIFO算法</strong></h3>

            <p>缺页数：<span id="mpage\_FIFO">NULL</span></P>

            <P>缺页率：<span id="mrate\_FIFO">NULL</span></p>

            <h3><strong>②LRU算法</strong></h3>

            <p>缺页数：<span id="mpage\_LRU">NULL</span></P>

            <P>缺页率：<span id="mrate\_LRU">NULL</span></p>

          </section>

        </div>

        <br>

        </br>

        <div class="container4">

          <!-- 开始按钮 -->

          <p><button class="w3-button w3-pale-red" id="start\_btn"><strong>开始！</strong> </button></p>

          <!-- 进度条 -->

          <div>

            <span style="font-size: large;">进度条：</span>

            <progress id="loading\_bar" value="0" max="320" min="0"></progress>

            <span id="now\_page">0</span><sapn>/320 页</span>

          </div>

          <!-- 拖动条 -->

          <div>

            <span style="font-size: large;">拖动条(指令间隔)：</span>

            <input type="range" min="0" max="1000" value="10" step="1" id="range\_bar">

            <span id="now\_rate">0</span><sapn>/1000 ms</span>

          </div>

          <div clsas="container3">

            <sapn>内存块01：</span>

            <span id="block1" style="font-size: large;">--</span>

            <sapn>内存块02：</span>

            <span id="block2" style="font-size: large;">--</span>

            <sapn>内存块03：</span>

            <span id="block3" style="font-size: large;">--</span>

            <sapn>内存块04：</span>

            <span id="block4" style="font-size: large;">--</span>

          </div>

        </div>

        <br></br>

        <!--信息动态展示-->

        <div class="show">

          <table class="presenting\_table" id="memory\_table" height="100px">

              <thead>

                <tr>

                  <th scope="col" >执行数</th>

                  <th scope="col" >指令序号</th>

                  <th scope="col" >内存块01</th>

                  <th scope="col" >内存块02</th>

                  <th scope="col" >内存块03</th>

                  <th scope="col" >内存块04</th>

                  <th scope="col" >是否缺页</th>

                  <th scope="col" >指令状态</th>

                </tr>

              </thead>

          </table>

        </div>

        <pre class="page\_footer">

          同济大学 软件学院 2151299 苏家铭

        </pre>

    </div>

  </body>

</html>

css:

@charset "UTF-8";

.container {

  margin-left: auto;

  margin-right: auto;

  text-align: center;

  width: 60%;

}

.container2{

  display:flex;

  /\* flex-wrap:wrap; \*/

  /\* 让区块位置不够就跳到下一行 \*/

  flex-direction:row;/\*变成横着排列\*/

  /\* height:300px; \*/

  border:10px solid rgb(192, 98, 167);

  border-radius: 10px;

  /\* justify-content:center; 居中（沿主轴）\*/

  /\* align-items:center;次轴置中 \*/

  /\* align-content:center; \*/

}

.container3{

  display:flex;

  flex-direction:row;/\*变成横着排列\*/

  border:2px none rgb(192, 98, 167);

  border-radius: 10px;

}

.container4{

  display:flex;

  flex-direction:column;/\*变成竖着排列\*/

  border:10px solid rgb(192, 98, 167);

  border-radius: 10px;

}

html {

  background: #c6cbc4;

  font-size: 13px;

  color: #040615;

}

.box\_algorithm{

  background: rgb(233, 165, 188);

  border-radius: 10px;

  display: inline-block;

  height: 340px;

  margin: 2rem;

  position: relative;

  width: 400px;

  box-shadow: 0 19px 38px rgba(0, 0, 0, 0.3), 0 15px 12px rgba(0, 0, 0, 0.22);

}

.box{

  background: #fff;

  border-radius: 10px;

  display: inline-block;

  height: 340px;

  margin: 2rem;

  position: relative;

  width: 200px;

  box-shadow: 0 19px 38px rgba(0, 0, 0, 0.3), 0 15px 12px rgba(0, 0, 0, 0.22);

}

table,

th,

td {

    border: 3px groove #4e2bcc;

    border-collapse: separate;

}

table {

  width: 80%;

  margin: 10px 10px 10px 10px;

  /\*height: 100px;\*/

}

th,

td {

    padding: 10px;

    width: 150px;

}

.show{

  height:650px;

  overflow-y: scroll;

  overflow-x: none;

}

#memory\_table{

  width: 100%;

  margin: auto;

  /\*height: 650px;\*/

  vertical-align: middle;

}

.page\_footer{

  display: inline-block;

  vertical-align: middle;

  font-size: medium;

}

button{

  text-align: center;

  font-size: large;

  font-weight:bold;

  position: relative;

  height: 50px;

  width: 100px;

  border: 1px solid #fff;

  background: #FDFCD6;

  border-radius: 6px;

  box-shadow: 0 19px 38px rgba(0, 0, 0, 0.3), 0 15px 12px rgba(0, 0, 0, 0.22);

}

progress {

  width: 300px;

  height: 30px;

  /\* border-radius: 10px; \*/

  background-color: #4caf50;

}

#range\_bar {

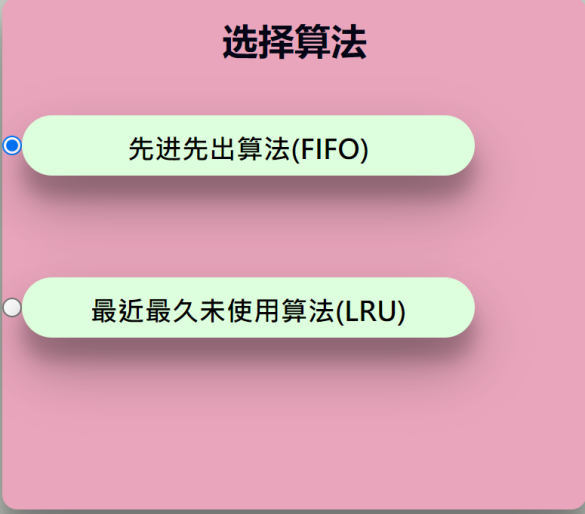
  width: 300px;

  height: 20px;

}

鉴于html的交互性强，并且易于用css美化和实现，本次项目一改往常的python+PyQt5，而是采用网页开发的方式来完成。

## 4.2选择算法



获取html中选择按钮组件被check的按钮的值，通过该值在JavaScript中调用对于的算法。

//算法的选择

    function algorithm() {

        var choose = document.querySelector("input:checked")

        if (choose.value === "FIFO") {

            algo\_FIFO()

        }

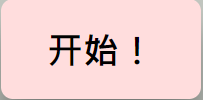
        else if (choose.value === "LRU") {

            algo\_LRU()

        }

    }

## 4.3开始按钮被按下



当按下开始按钮后会初始化相关变量，并且禁用开始按钮，进入算法选择的实现，最后展示到结果框中。

    function start() { // 点击按钮

        start\_btn.disabled = true//对开始按钮的保护

        init()//初始化

        $("#memory\_table  tr:not(:first)").hide()

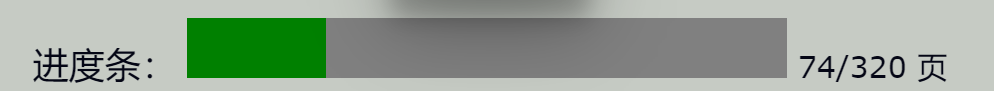
        algorithm()// 选择算法并执行

        start\_btn.disabled = false //重新启用开始按钮

    }

    $("#start\_btn").click(start) //监听点击按钮事件

## 4.4进度条

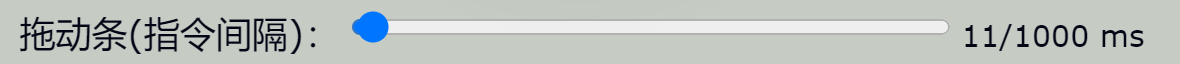


在指令调度的过程中可以通过进度条观察此时执行到第几条指令，并且可以根据指令间隔，在320条指令未执行完时动态地显示执行进度。在JavaScript中获取进度条信息，并在算法执行的过程中动态的更新，就可以显示在前端。

var loading\_bar = document.getElementById("loading\_bar")//进度条

now\_page.innerHTML = count\_instuctions//进度更新

## 4.5拖动条



通过一个监听事件，在JavaScript中动态获取拖动条的值，并将该值作为算法每层循环的睡眠时间，可以实现动态调整每步指令的跳转时间，也就是说，当320条还未执行完时，可以调整执行速度，在关键点可以放慢速度观察相关信息的变化。

    var now\_rate = document.getElementById("now\_rate")

    $('#range\_bar').on('input', function() {//添加对拖动条事件的监听

        var value = $(this).val();

        console.log('拖动条的值:', value);

        now\_rate.innerHTML = value;

      })

## 4.6内存块页显示



动态显示每个内存块的页号，实现原理是在JavaScript中动态获取每个内存块的页号并传给html中的相关控件进行展示。

    var block1 = document.getElementById("block1")//内存块1

    var block2 = document.getElementById("block2")//内存块1

    var block3 = document.getElementById("block3")//内存块1

var block4 = document.getElementById("block4")//内存块1

    //内存块的值导到html中

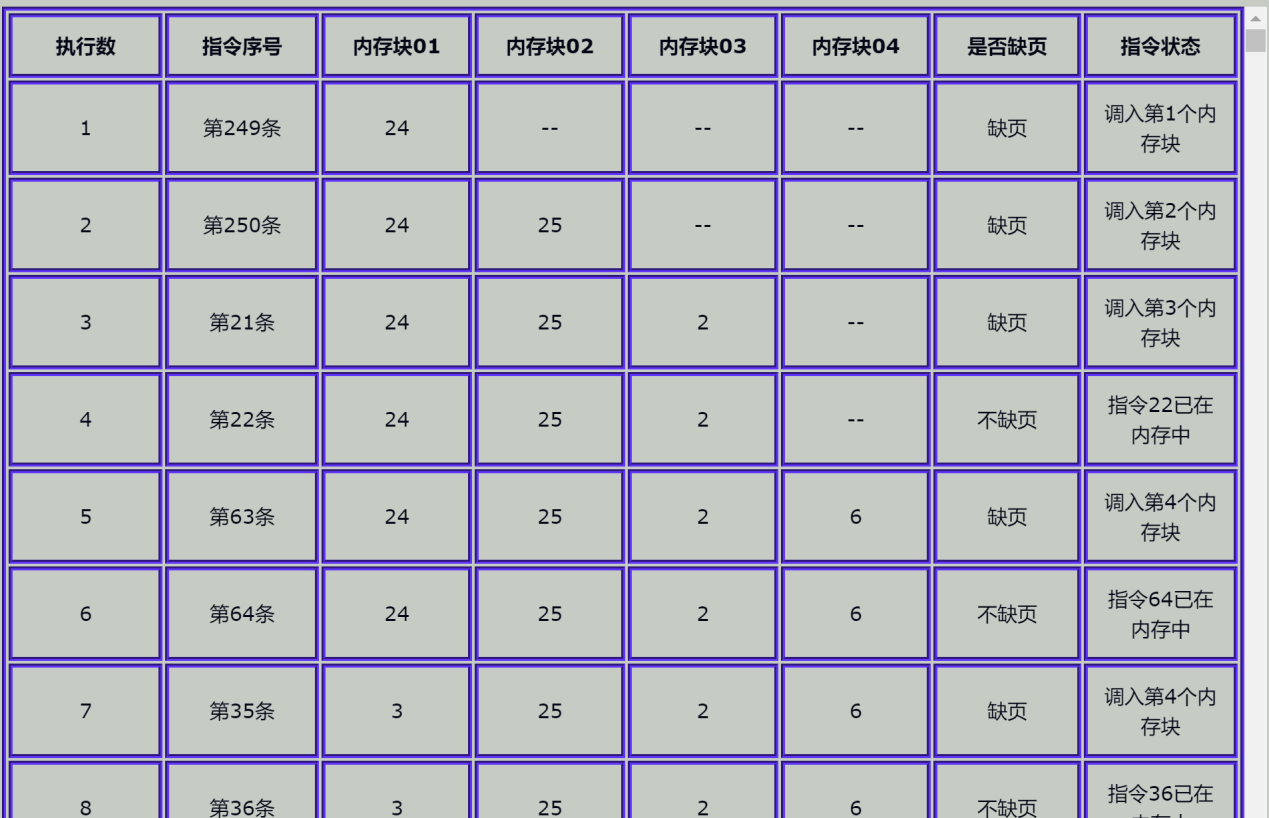
    block1.innerHTML = memory[0]

   block2.innerHTML = memory[1]

    block3.innerHTML = memory[2]

    block4.innerHTML = memory[3]

## 4.7结果展示



将程序执行的的结果动态输出到结果框进行记录，可以实现长时间保留信息，并查询每一步的信息。在html中建立表格，在JavaScript中把结果信息传输到html中的表格对应的位置，可以实现相关功能。

//结果输出

var row = document.getElementById("memory\_table").insertRow()

row.insertCell(0).innerHTML = count\_instuctions

row.insertCell(1).innerHTML ="第" + currentIns + "条"

for (let i = 0;i < 4;i++){

     row.insertCell(i+2).innerHTML =

     memory[i] == undefined ? "--" : memory[i]

}

//内存块的值导到html中

block1.innerHTML = memory[0]

block2.innerHTML = memory[1]

block3.innerHTML = memory[2]

block4.innerHTML = memory[3]

if(missing\_flag === 1){

    row.insertCell(6).innerHTML = "缺页 "

     row.insertCell(7).innerHTML = "调入第" + (pos + 1) + "个内存块"

}

else{

     row.insertCell(6).innerHTML = "不缺页"

     row.insertCell(7).innerHTML = "指令"+ currentIns + "已在内存中"

}

# 5.算法

## 5.1全局变量

|  |  |  |
| --- | --- | --- |
| 编号 | 变量名 | 作用 |
| 1 | MEMORY\_BLOCKS | 记录内存块数 |
| 2 | INSTRUCTIONS | 记录320条指令 |
| 3 | PAGE\_OF\_INSTRUCTION | 记录每页存放10条指令 |
| 4 | PAGES | 记录共32页 |
| 5 | memory | 记录内存块的页号 |
| 6 | count\_instuctions | 记录执行的指令个数 |
| 7 | mpage\_num\_of\_FIFO | 记录FIFO缺页数 |
| 8 | mpage\_num\_of\_LRU | 记录LRU缺页数 |
| 9 | record | 记录最后一次调入内存的时间（用于LRU） |

## 5.2功能逻辑

### 5.2.1 延时输出

为了实现可视化动态输出每个内存块的页号，要使用延时功能产生指令的跳转，JavaScript中没有sleep函数，可以使用以下方法实现：

//延时输出

function sleep(time){

    return new Promise((resolve) => setTimeout(resolve, time));

}

await sleep(now\_rate.innerHTML)//睡眠

### 5.3.2执行模式

320条指令产生方式如下：

* 在0－319条指令之间，随机选取一个起始执行指令，如序号为m
* 顺序执行下一条指令，即序号为m+1的指令
* 通过随机数，跳转到前地址部分0－m-1中的某个指令处，其序号为m1
* 顺序执行下一条指令，即序号为m1+1的指令
* 通过随机数，跳转到后地址部分m1+2~319中的某条指令处，其序号为m2
* 顺序执行下一条指令，即m2+1处的指令。

重复跳转到前地址部分、顺序执行、跳转到后地址部分、顺序执行的过程，直到执行完320条指令。

### 5.3.3置换和请求调页算法

一传统置换算法：

* 先进先出算法(FIFO)：选择建立最早的页面被置换。
* 最佳算法(OPT)：选择“未来不再使用的”或“在离当前最远位置上出现的”页面被置换。
* 最近最久未使用算法(LRU)：使用离过去最近作为不远将来的近似，置换最长时间没有使用的页。
* 最不常用算法(LFU)：选择到当前时间为止被访问次数最少的页面被置换。每页设置访问计数器，每当页面被访问时，该页面的访问计数器加1。发生缺页中断时，淘汰计数值最小的页面，并将所有计数清零。
* 轮转算法(clock)：也称最近未使用算法(NRU)，是LRU和FIFO的折衷。每页标志位(use)，若该页被访问则置user =1。置换时采用一个指针，从当前指针位置开始按地址先后检查各页，寻找use=0的页面作为被置换页，并将指针经过的页修改为user=0 ，最后指针停留在被置换页的下一个页。

本项目的置换算法基于FIFO和LRU算法实现。

FIFO：

使用一个pointer指针指向将要被置换的内存块序号，每次置换时，pointer的值更新为(++pointer)%4

if(in\_memory(currentIns) === -1){//缺页

     missing\_flag = 1

     if(mem\_not\_full() !== -1){//内存未满

     not\_full\_num = mem\_not\_full()

     memory[not\_full\_num] = Math.floor(currentIns / PAGE\_OF\_INSTRUCTION)

}

     else{//内存已满

memory[pointer] = Math.floor(currentIns / PAGE\_OF\_INSTRUCTION)

     pointer = ( ++pointer ) % 4

     mpage\_FIFO.textContent = mpage\_num\_of\_FIFO

     mrate\_FIFO.textContent = mpage\_num\_of\_FIFO / INSTRUCTIONS

}

     mpage\_num\_of\_FIFO++

}

LRU：

使用一个record数组记录内存块中的页号调入内存块中的时间，找到最先调入内存块的页号，把它所在的内存块作为将要置换的内存块。

    //找放进哪个内存块

    function find\_block(){

        var r = []

        for(let i =0;i<4;i++){

            r[i] = record[memory[i]]

        }

        var min\_pior = Math.min(r[0],r[1],r[2],r[3])

        for(let j = 0; j < MEMORY\_BLOCKS; j++){//找到是哪块应该被置换

            if(record[memory[j]] === min\_pior){

                return j //第j块

            }

        }

    }

### 5.3.4 产生随机指令

Math.random() 返回一个介于 0（包含）和 1（不包含）之间的随机浮点数。

这意味着它可以生成像 0、0.234、0.753 或 0.999 这样的值，但绝不会是完全等于 1 的值。

(INSTRUCTIONS - 1) 从 INSTRUCTIONS 的值中减去 1。得到的数确定了随机数的上限范围。

将Math.random() 乘以 (INSTRUCTIONS - 1)，

将Math.random() 生成的随机数按比例缩放到 0 到 (INSTRUCTIONS - 1) 的范围内。

结果将是一个介于 0 和 (INSTRUCTIONS - 1) 之间的随机浮点数。

Math.round() 函数会将结果四舍五入为最接近的整数。

cur\_ins = Math.round( Math.random() \* (INSTRUCTIONS-1))

# 程序运行截图

