**天津财经大学理工学院信息科学与技术系**

**《操作系统》实验报告**

**实验3 动态分区分配算法**

**班 级：软件2201**

**学 号：2022113179**

**姓 名：钟奇林**

**一、实验目的**

通过本实验模拟实现操作系统对存储器动态分区分配的管理方法，从而加深对动态分区分配算法的理解。

**二、实验内容**

编写实现动态分区分配算法。最佳适应法、最先适应法或最坏适应法的程序，任选其一。

根据某一存储分配算法编程，判断系统的空闲分区能否满足某一作业序列的请求。要求运行程序时，输入数据包括：空闲分区表、作业请求序列；程序运行结果是：如果能够满足作业序列的请求，则输出分配后的空闲分区表，否则输出拒绝分配的提示即可。

**三、实验结果**

最先适应法

核心代码说明如下：

**void** DPA::Amend(vector<Partion>**&** table, vector<Process>**&** p\_size\_a, **int** index, **int** num) {

*// 修改指定索引的表项的partition\_size，并更新相应的initial\_address*

    table[index].partion\_size -= p\_size\_a[num].p\_size;

    p\_size\_a[num].initial\_address = table[index].initial\_address;

    table[index].initial\_address += p\_size\_a[num].p\_size;

}

*// 对给定的partition\_size大小进行查找，返回可用的空闲分区的索引*

**int** DPA::Find(vector<Partion>**&** table, **int** p\_size) {

**int** index = -1;

    for (**int** j = 0; j < table.size(); ++j)

        if (table[j].partion\_size >= p\_size) {

            index = j;

            break;

        }

    return index;

}

*// 使用First Fit算法进行动态分区分配*

**bool** DPA::FirstFit(vector<Partion>**&** table, vector<Process>**&** p\_size\_a) {

*// 按照initial\_address对表进行升序排序*

    sort(table.begin(), table.end(), [](Partion x, Partion y)->**bool** {return x.initial\_address < y.initial\_address; });

**int** num = 0;

*// 对每个process使用First Fit算法进行分区分配*

    for (Process& p : p\_size\_a) {

**int** index = Find(table, p.p\_size);

        if (index == -1) return false;

*// 修改数据*

        this->Amend(table, p\_size\_a, index, num++);

    }

    return true;

}

*// 使用Best Fit算法进行动态分区分配*

**bool** DPA::BestFit(vector<Partion>**&** table, vector<Process>**&** p\_size\_a) {

*// 按照partion\_size对表进行升序排序*

    sort(table.begin(), table.end(), [](Partion x, Partion y)->**bool** {return x.partion\_size < y.partion\_size; });

**int** num = 0;

*// 对每个process使用Best Fit算法进行分区分配*

    for (Process& p : p\_size\_a) {

**int** index = Find(table, p.p\_size);

        if (index == -1) return false;

*// 修改数据*

        this->Amend(table, p\_size\_a, index, num++);

*// 重新排序*

        sort(table.begin(), table.end(), [](Partion x, Partion y)->**bool** {return x.partion\_size < y.partion\_size; });

    }

    return true;

}

*// 使用Worst Fit算法进行动态分区分配*

**bool** DPA::WorstFit(vector<Partion>**&** table, vector<Process>**&** p\_size\_a) {

*// 按照partion\_size对表进行降序排序*

    sort(table.begin(), table.end(), [](Partion x, Partion y)->**bool** {return x.partion\_size > y.partion\_size; });

**int** num = 0;

*// 对每个process使用Worst Fit算法进行分区分配*

    for (Process& p : p\_size\_a) {

**int** index = Find(table, p.p\_size);

        if (index == -1) return false;

*// 修改数据*

        this->Amend(table, p\_size\_a, index, num++);

*// 重新排序*

        sort(table.begin(), table.end(), [](Partion x, Partion y)->**bool** {return x.partion\_size > y.partion\_size; });

    }

    return true;

}

运行结果如下：

**四、实验总结**

**能力仍需提升，对需求的分析不够充分，算法实现简单。最终有一个问题，由于晚上写多了，没保存，最后从vscode的时间线功能找到历史版本，再完善，如果每次都提交到仓库里，也就不会发生代码遗失的问题。**