****

**实 验 报 告**

课 程 名 称： 操作系统

实 验 项 目： 首次适应算法的模拟

专 业 班 级： 23级软件工程专升本3班

姓 名： 罗昕

学 号： 202316060328

日 期： 2023年12月15日

**实践教学中心制**

**实验报告说明**

1. 实验报告应包含实验目的、实验条件、实验方法、实验内容、结果分析、成绩评定等内容。
2. 学生应认真撰写实验内容，并对实验进行总结分析。
3. 指导教师应严格依照成绩评定标准对实验报告进行成绩评定并签字，评语围绕实验报告内容，具体、有针对性、有建设性。
4. 格式要求：A4幅面，双面打印。**封面：**汉字采用三号宋体、外文数字采用三号Times New Roman体，下划线居中填写，下划线整体长度不改变；**正文：**汉字采用小四号宋体、外文数字采用小四号Times New Roman体；左对齐填写，首行缩进2字符，行距固定值20磅，图或表的字体大小为五号字。

|  |  |  |  |
| --- | --- | --- | --- |
| **实验地点：** |  | **指导教师：** |  |
| **同组人员：** |  | | |
| **一、实验目标**  1.掌握动态分区分配方式使用的数据结构和分配算法（首次适应算法）。 2.进一步加深对动态分区分配管理方式及其实现过程的理解。 | | | |
| **二、实验条件（环境）**  1.VMware  2.CentOS7 | | | |
| **三、实验方法（原理）**  编写C语言程序，模拟实现首次适应算法的内存块分配和回收，要求每次分配和回收后显示出空闲分区和已分配分区的情况。假设初始状态下，可用的内存空间为640KB。  实验要求：  假设下列作业请求序列  （1）作业1 申请130 KB  （2）作业2 申请60 KB  （3）作业3 申请100 KB  （4）作业2 释放60 KB  （5）作业3 释放100 KB  （6）作业1 释放130 KB  实现方法：  1.内存分配：遍历空闲分区链表，查找满足申请大小的节点。若找到合适的节点，根据申请的大小进行分割，分配内存给进程使用。  2.内存回收：将回收的内存块按照地址和大小的合法性插入到空闲分区链表中。在插入过程中，对于相邻的空闲分区，如果它们的地址是连续的，则进行合并，以减少碎片。 | | | |
| **四、实验内容（步骤）**  #include <stdio.h> #include <stdbool.h> #define MAX\_SIZE 1000 typedef struct {  int process\_id;  int size; // 大小  bool allocated; } MemoryBlock; typedef struct {  int start\_address;  int size;  bool allocated; } Partition; MemoryBlock memory[MAX\_SIZE]; Partition partitions[MAX\_SIZE]; int num\_partitions = 0; void initialize() {  num\_partitions = 0; } void addPartition(int start\_address, int size) {  partitions[num\_partitions].start\_address = start\_address;  partitions[num\_partitions].size = size;  partitions[num\_partitions].allocated = false;  num\_partitions++; } void allocateMemory(int process\_id, int size) {  int i;  for (i = 0; i < num\_partitions; i++) {  if (!partitions[i].allocated && partitions[i].size >= size) {  memory[partitions[i].start\_address].process\_id = process\_id;  memory[partitions[i].start\_address].size = size;  memory[partitions[i].start\_address].allocated = true;  partitions[i].size -= size;  partitions[i].start\_address += size;  if (partitions[i].size == 0) {  // 如果分区已被用完，从分区表中删除  for (int j = i; j < num\_partitions - 1; j++) {  partitions[j] = partitions[j + 1];  }  num\_partitions--;  }  printf("分配成功！分区号：%d\n", partitions[i].start\_address);  return;  }  }  printf("分配失败！没有足够的空闲分区可供分配。\n"); } void releaseMemory(int process\_id) {  for (int i = 0; i < MAX\_SIZE; i++) {  if (memory[i].process\_id == process\_id && memory[i].allocated) {  int partition\_index = -1;   // 找到被释放的分区在分区表中的索引  for (int j = 0; j < num\_partitions; j++) {  if (partitions[j].start\_address + partitions[j].size == i) {  partition\_index = j;  break;  }  }  if (partition\_index >= 0) {  partitions[partition\_index].size += memory[i].size;  if (partition\_index < num\_partitions - 1 && partitions[partition\_index].start\_address + partitions[partition\_index].size == partitions[partition\_index + 1].start\_address) {  partitions[partition\_index].size += partitions[partition\_index + 1].size;  for (int j = partition\_index + 1; j < num\_partitions - 1; j++) {  partitions[j] = partitions[j + 1];  }  num\_partitions--;  }  } else {  addPartition(i, memory[i].size);  }  memory[i].allocated = false;  printf("释放成功！分区号：%d\n", partitions[partition\_index].start\_address);  return;  }  }  printf("释放失败！没有找到要释放的进程。\n"); } void printMemory() {  printf("内存状态：\n");  for (int i = 0; i < MAX\_SIZE; i++) {  if (memory[i].allocated) {  printf("分区号：%d\t进程ID：%d\t大小：%d KB\n", i, memory[i].process\_id, memory[i].size);  }  }  printf("\n"); } void printFreePartitions() {  printf("空闲分区表：\n");  for (int i = 0; i < num\_partitions; i++) {  printf("分区号：%d\t大小：%d KB\n", partitions[i].start\_address, partitions[i].size);  }  printf("\n"); } int main() {  int process\_id;  int size;  initialize();  addPartition(0, 1024);  allocateMemory(1, 130);  allocateMemory(2, 60);  allocateMemory(3, 100);  printMemory();  printFreePartitions();  releaseMemory(2);  releaseMemory(3);  releaseMemory(1);  printMemory();  printFreePartitions();  return 0; } | | | |
| **五、结果分析**  通过内存分配和内存回收完成实验  掌握了动态分区分配方式使用的数据结构和分配算法（首次适应算法）并进一步加深对动态分区分配管理方式及其实现过程的理解。算法要求空闲链已地址递增的次序连接。分配内存时，从链首开始顺序查找，直到找到第一个满足要求的空间并分配给进程，把分配后余下的空间仍然留在链表中。若从链首至链尾都不满足要求，则分配失败。该算法倾向于优先使用低地址的空间，在不断分割中会产生很多空间碎片，并且每次都是从链首开始查找，这无疑增加了开销。 | | | |
| **六、成绩评定**  **评 语：**  **评分：**  **指导教师签字：** | | | |