西南石油大学实验报告

**院/系：计算机科学学院 课程名称：操作系统**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **班号** | 计算机类202008 | **姓名** | 夏卓昭 | **学号** | 202031061299 | **实验室号** | 明理楼B411 |
| **日期** | **2021年11月11日** | **组号** | 无 | **计算机号** | 25 |
| **实验名称** | 存储管理的模拟程序设计 | | | | | **成绩评定** |  |
| **所用软件** | Linux version 5.10.60-amd64-desktop (gcc (Uos 8.3.0.6-1+dde) 8.3.0, GNU ld (GNU Binutils for Uos) 2.31.1), VS CODE | | | | | **老师签名** |  |
| **实 验 目 的**  **或**  **要**  **求** | 实验目的：实验要求从基于连续分配思想的**动态分区分配**方案和基于离散分配思想的**分页**方案中选择一种方案进行内存空间的模拟管理。   1. 动态分区分配 2. 要求：根据实际需要，动态地创建分区为之分配内存空间。 3. 数据结构：1. 空闲分区表；2. 空闲分区链表。用于记录空闲块的使用状况。 4. 分配过程：若分配需求内存后剩余空间小于等于1K，则全部分配；否则，一分为二。 5. 分配算法：首次适应法、最佳适应法、最差适应法。 6. 内存回收：根据释放区域地址，判断是否需要合并链表结点。 7. 分页技术实现 8. 要求：开发一个C语言程序实现内存管理的分页技术。 9. 获取物理内存的总大小和逻辑页面大小； 10. 对物理内存进行分块； 11. 初始化物理内存的数据结构，注意内存中的块有些应当为已分配，有些为空闲，可通过随机生成； 12. 构建空闲块链表或者位示图； 13. 获取进程的内存需求，将进程的程序地址空间进行划分，n个页面； 14. 如果有n个物理块可用，则给进程分配并更新进程页表； 15. 能完成一些逻辑地址到相应物理地址的转换； 16. 完成进程的相应空间回收； 17. 能重复完成多个进程的空间管理。 | | | | | | |
| **实**  **验**    **步**  **骤**  **、 心**  **得**    **体**    **会**  **总结** | **基于C++的动态分区内存分配方案的设计**   1. 实验简介   本人使用C++ with Classes实现了一套模拟动态分区分配方案，模拟页式内存管理，总内存大小为1M。使用按块分配的方案，块大小取4K，总共256个块用于分配，用2位16进制数字表示块地址，忽略块内地址。内存空间使用空闲分区链表表示，链表使用C++ STL List模板方案实现。项目使用交互式方案设计。本人还构建了一套图形界面用于生动地展示每一个块的使用情况，未分配的块用空心方块表示，已分配的块用实心方块表示。   1. 功能与界面展示 2. 初始界面      1. 操作命令 2. firstadapt size：使用首次适应法分配size个内存块，size为16进制数字； 3. bestadapt size：使用最佳适应法分配size个内存块，size为16进制数字； 4. worstadapt size：使用最差适应法分配size个内存块，size为16进制数字； 5. free start size：释放掉以start为首块编号、size大小的内存块，参数均为16进制数字； 6. exit：退出。 7. 内存分配   以下全部内容均基于如图所示的基本状态：     1. 首次适应法     使用首次适应法分配4个块的内存，被分配在了块编号为0x10-0x13的地方   1. 最佳适应法     使用最佳适应法分配4个块的内存，被分配在了块编号为0x48-0x4B的地方   1. 最差适应法     使用最差适应法分配4个块的内存，被分配在了块编号为0x10-0x13的地方   1. 分配规则   按照题意，当分配后剩余空间不足1K时，应当把此块全部分配。但在本人的方案中，分配的最小单位是1个块（4K），因此本实验中将当分配后剩余空间不足4K的全部分配。例如，在前述情况下，尝试以最佳适配法分配31（0x1F）个块的空间，程序会将0x80-0x9F的块全部分配，实际分配32个块。     1. 内存释放     要求程序从0x70开始释放0x10个连续的内存块，程序将0x70-0x7F释放   1. 打印链表   SpaceAllocater类的showlinklist()方法可用于打印链表的状况，用于调试上述过程中链表的合并、拆分状态。     1. 类的设置   实验设计有两个类。   1. FreeSpace   FreeSpace类用于表示一个连续的内存块地址，并重载了==运算符。  class FreeSpace  {  public:  int startBlock; //起始块地址  int endBlock; //终点块地址  int size(); //计算并返回块的大小  FreeSpace(int startBlock, int endBlock); //构造函数  bool operator ==(const FreeSpace& s); //运算符'=='的重载  };   1. SpaceAllocater   SpaceAllocater类用于实现内存的分配算法，使用类编写划分了命名空间，符合程序的设计美学。  class SpaceAllocater  {  private:  void allocate(FreeSpace f, int size); //本来想用此函数实现具体的分配操作，而下方的几个xxxAdapt函数仅仅用来查找一个合适的空间，以便进一步提高程序的内聚性；结果写着写着不小心忘了它的存在，留下一个小小的残念  std::list<FreeSpace>freeSpaceList; //使用C++ STL模板构建的空闲分区链表，装载有FreeSpace对象  public:  SpaceAllocater(); //类的构造函数  int firstAdapt(int size); //首次适应法  int bestAdapt(int size); //最佳适应法  int worstAdapt(int size); //最差适应法  int free(int start, int size); //释放内存  void showMemoryUseage(); //显示内存使用情况  void showLinkList(); //打印空闲分区链表，用于调试和学习  };   1. 分配算法的设计   以首次适应法的设计为例，代码实现如下：  int SpaceAllocater::firstAdapt(int size)  {  int flag = ERROR; //存放分配结果，默认为失败  if (size <= 0 || size > 256) //对分配命令进行检错  {  printf("ERROR: Invalid size %d.\n", size);  return ERROR;  }  list<FreeSpace>::iterator ListIterator = freeSpaceList.begin(); //使用迭代器遍历链表  while (ListIterator != freeSpaceList.end())  {  if ((\*ListIterator).size() >= size) //如果首次遇到合适的内存空间  {  if ((\*ListIterator).size() - size > 1) //如果分配后剩余空间大于一个块，则一分为二再分配  {  printf("Block %X to %X will be allocated.\n", (\*ListIterator).startBlock, (\*ListIterator).startBlock + size - 1);  (\*ListIterator).startBlock += size;  }  else //如果分配后剩余空间小于等于一个块，则全部分配  {  printf("Block %X to %X will be allocated.\n", (\*ListIterator).startBlock, (\*ListIterator).endBlock);  ListIterator = this->freeSpaceList.erase(ListIterator);  }  flag = SUCCEED; // 分配成功  break;  }  ListIterator++;  }  if (!!flag)  printf("ERROR: No free space.\n");  return flag;  }   1. 释放算法的设计   释放算法是本实验中最复杂的一个部分。以下是对算法的简介，篇幅所限，具体代码细节不表，在本报告随后会将代码附上。   1. 释放算法的指令检查部分，本人考虑了以下3种情况。 2. 遍历整个链表，如果待释放的起始地址和终止地址不被任何一个空闲分区链表中的结点包含，那么代表释放指令是安全的。      1. 如果释放的起始地址或终止地址被任何一个空闲分区链表中的结点所包围，那么指令是不安全的      1. 对于一个空闲分区链表中的结点，如果它的首地址比请求起始地址大，末地址比请求终止地址小，那么指令是不安全的。      1. 内存释放的代码实现，根据实际情况，本人考虑了以下6种情况，并根据具体情况分别编写了相应的代码。 2. 待释放地址在初始地址附近，前端没有空闲分区链表中的任何一个结点，且末端不与任何一个结点相邻，则在第一个结点前新建一个结点。      1. 待释放地址在初始地址附近，末端与空闲分区链表中的第一个结点相邻，则向前延长第一个结点的起始地址。      1. 待释放地址前端与空闲分区链表中的结点相邻，则延长这个结点。      1. 待释放地址前端有空闲分区链表中的结点但不与之相邻，且后端不与任何一个结点相邻，则新建一个结点      1. 待释放地址前端有空闲分区链表中的结点但不与之相邻，但后端与下一个结点相邻，则新建一个结点      1. 待释放地址两端与结点相邻，则删除前一个结点，并将后一个结点延长至前一个结点的头部      1. SpaceAllocater类运算符的重载   这个实验中非常有意思的一部分是运算符的重载过程，我第一次知道C++居然连运算符都能重载。重载运算符的目的是为了在后文中进行链表的.remove()或者.erase()操作，然而在List STL的.tcc文件里进行移除操作只能对C语言基本数据类型进行比对，因此会报错。我不能修改STL的tcc文件，因此只能重写==运算符。  bool FreeSpace::operator==(const FreeSpace &s)  {  if (s.startBlock == this->startBlock && s.endBlock == this->endBlock)  return true;  return false;  }   1. 界面的程序设计   void SpaceAllocater::showMemoryUseage()  {  int spaceUsage[NUM\_BLOCK];  int unUsedBlocks = 0;  float useage;  for (int i = 0; i < NUM\_BLOCK; i++)  {  spaceUsage[i] = 1;  }  list<FreeSpace>::iterator ListIterator = freeSpaceList.begin();  while (ListIterator != freeSpaceList.end())  {  FreeSpace f = \*ListIterator;  for (int i = f.startBlock; i <= f.endBlock; i++) //生成空闲分区表  {  spaceUsage[i] = 0;  unUsedBlocks++;  //printf("%d unused \n", i);  }  ListIterator++;  }  useage = 1.0 - (1.0 \* unUsedBlocks / NUM\_BLOCK); //计算内存占用比  printf("Useage:\n ");  for (int i = 0; i < 16; i++)  {  printf("%s", i < 16 \* useage ? "■ " : "□ "); //形象地展示内存使用率  }  printf(" %.2lf%% Used\n", useage \* 100);  printf("Useage Map:\n "); //形象地展示内存使用状况  for (int i = 0; i < 16; i++)  {  printf("%X ", i);  }  printf("\n");  for (int i = 0; i < 16; i++)  {  printf("%X ", i);  for (int j = 0; j < 16; j++)  {  printf("%s", spaceUsage[i \* 16 + j] == 1 ? "■ " : "□ ");  }  printf("\n", i);  }  return;  }   1. 检错机制   拥有良好的检错机制是良好程序的特征之一。本实验中，在以下部分存在充分的检错机制。   1. 命令操作界面中，会对每次的输入内容进行检错。如果输入错误，则提示“INVALID OPERATION”并要求重新输入。每次输入结束后，都会使用cin.clear()清空输入缓冲区（实际上，提到清空输入缓冲，我的第一个念头是fflush(stdin)，然而这并没有奏效。查阅资料后发现，这只是微软对C++的一个扩充，而我现在正在使用Linux平台，在GCC下这是一个未定义行为）； 2. 分配内存过程中，会对输入的数据检错； 3. 释放内存的过程中，会通过严格的算法检测释放区域是否包括未分配区域。 | | | | | | |

备注：本实验报告用于各学科与计算机应用相关课程的实验，务必按时完成。不交此报告者，本次实验为“不合格”。