

**实 验 报 告**

**（ 2024/2025 学年 第 一 学期 ）**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 课程名称 | 计算机操作系统 | | | | | |
| 实验名称 | 文件系统设计与模拟实现 | | | | | |
| 实验时间 | 2024 | 年 | 12 | 月 | 13 | 日 |
| 指导单位 | 物联网学院 | | | | | |
| 指导教师 |  | | | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| 学生姓名 |  | 班级学号 |  |
| 学院(系) | 物联网学院 | 专 业 | 网络工程 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **实验名称** | 文件系统设计与模拟实现 | | | | | | | | | **指导教师** | |  |
| **实验类型** | | 设计性实验 | | **实验学时** | | 2 | | | **实验时间** | | | 2024.12.13 |
| 1. **实验目的和要求** | | | | | | | | | | | | |
| **实验目的：**成组链接法是一种磁盘空闲空间管理方法。通过编程模拟实现成组链接法，加深对成组链接法设计思想的理解。  **实验要求：**  在 Linux 环境下，用 C 语言编程；理解成组链接法的数据结构和工作机制。 | | | | | | | | | | | | |
| **二、实验环境(实验设备)** | | | | | | | | | | | | |
| Linux | | | | | | | | | | | | |
| 1. **实验原理及内容**   1、 根据用户定义的盘块大小、文件区磁盘块总数和每组的块数，模拟Unix系统将磁盘空闲块进行分组，并初始化成组链。  2、 可查询成组链中空闲盘块总数、盘块分组情况。  3、 实现盘块的分配或回收操作，显示每次分配或回收盘块的结果。  4、 实现将空闲盘块的分组信息写入文件和从文件中读出的过程。  #include <stdio.h>  int MA[4]; /\*空闲块数组\*/  int A[9][4] = {  {3, 1, 2, 3}, {3, 4, 5, 6}, {0, 0, 0, 0},  {0, 0, 0, 0}, {3, 0, 7, 8}, {0, 0, 0, 0},  {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}  }; /\*磁盘空间\*/  int mark[9]; /\*存放已分配的块\*/  int No = 0; /\*已分配的块数\*/  void display1()  {  int i, j, temp, count;  No = 0;  if (MA[1] != 0)  {  i = MA[0];  printf("\ngroup1:");  for (j = 1; j <= i; j++)  {  printf("%d ", MA[j]);  mark[++No] = MA[j];  }  temp = MA[1];  count = 2;  while (A[temp][1] != 0)  {  printf("\ngroup%d:", count);  i = A[temp][0];  for (j = 1; j <= i; j++)  {  printf("%d ", A[temp][j]);  mark[++No] = A[temp][j];  }  count++;  temp = A[temp][1];  }  printf("\ngroup%d:", count);  i = A[temp][0];  for (j = 2; j <= i + 1; j++)  if (A[temp][j] > 0)  {  printf("%d ", A[temp][j]);  mark[++No] = A[temp][j];  }  }  else  {  i = MA[0];  if (i == 1)  printf("\nThe blocks are all assigned");  else  {  printf("\ngroup1:");  for (j = 2; j <= i; j++)  {  printf("%d ", MA[j]);  mark[++No] = MA[j];  }  }  }  }  void display() /\*显示分组情况\*/  {  int i, j;  if (MA[0] != 0)  display1();  else  {  i = MA[1];  for (j = 0; j <= 3; j++)  MA[j] = A[i][j];  display1();  }  }  void assign() /\*分配空闲块\*/  {  int s, i;  if (MA[0] > 1) /\*若该组不止一个空闲块\*/  {  i = MA[0];  s = MA[i];  MA[0]--;  printf("\nnumber of the block:%d", s);  }  else if (MA[0] == 1) /\*只剩一个空闲块\*/  {  if (MA[1] != 0) /\*还有其它空闲块组\*/  {  s = MA[1];  for (i = 0; i <= 3; i++)  A[0][i] = A[s][i];  MA[0]--;  printf("\nnumber of the block:%d", s);  }  else /\*没有其它空闲块组\*/  {  printf("\nThere isn't any space");  return;  }  }  else /\*当前组已分配完\*/  {  for (i = 0; i <= 3; i++)  MA[i] = A[0][i];  assign();  }  display(); /\*显示分组情况\*/  }  void callback() /\*回收空闲块\*/  {  int i, j, temp;  printf("\ninput the No. of the block you want to callback:");  scanf("%d", &j);  getchar(); /\*得到待回收的空闲块号\*/  for (temp = 1; temp <= No; temp++)  {  if (mark[temp] == j)  break;  }  if (temp < No + 1) /\*若该空闲块已在，退出\*/  {  printf("\nThe block is in the disk");  return;  }  if (MA[0] < 3) /\*当前组不满3块\*/  {  i = MA[0];  MA[i + 1] = j;  MA[0]++;  }  else /\*已有3块\*/  {  for (i = 0; i <= 3; i++)  A[j][i] = MA[i];  MA[0] = 1;  MA[1] = j;  }  display(); /\*显示\*/  }  void menu() /\*功能选择函数\*/  {  int choice;  char judge;  printf("\ninput your choice:(1--assign,2--callback):");  scanf("%d", &choice);  getchar();  if (choice == 1)  assign();  else if (choice == 2)  callback();  else  printf("\ninvalid command!");  printf("\ncontinue or not?(y--Yes,n--Not):");  scanf("%c", &judge);  getchar();  if (judge == 'y')  menu();  else  {  printf("\nNow the graph is:");  display();  printf("\npress any key to quit");  int getch(void);  }  }  int main(void)  {  int i;  for (i = 0; i <= 3; i++)  MA[i] = A[0][i];  display();  menu();  return 0;  } | | | | | | | | | | | | |
| **四、实验小结（包括问题和解决方法、心得体会、意见与建议等）** | | | | | | | | | | | | |
| **思考题：**   1. 超级块中的空闲盘块号栈是临界资源，如何实现对栈的互斥操作？   **使用互斥锁（Mutex）：**每次操作空闲盘块号栈时（如分配盘块或回收盘块），先获取互斥锁。操作完成后释放互斥锁。   **使用信号量（Semaphore）**：信号量用于限制同时访问空闲盘块号栈的线程数量为1。在进入操作时调用sem\_wait，操作结束后调用sem\_post。   1. 假定文件系统启用时，文件区磁盘块总数25块，编号从1至25。每10块划分一组，每组中最后一块登记下一组的所有盘块号和盘块数。请画出目前各组盘块的成组链接状态。   磁盘块总数为25，每10块划分一组，因此将有三组盘块：  第一组：块号1到10  第二组：块号11到20  第三组：块号21到25  **成组链接状态：**  **第一组（块号1到10）**  **块10**（最后一块）：记录第二组的盘块数和块号。内容：{盘块数=10, 块号=11, 12, 13, 14, 15, 16, 17, 18, 19, 20}  **第二组（块号11到20）**  **块20**（最后一块）：记录第三组的盘块数和块号。内容：{盘块数=5, 块号=21, 22, 23, 24, 25}  **第三组（块号21到25）**  **块25**（最后一块）：没有下一组，因此记录为空。内容：{盘块数=0, 块号=NULL}   |  |  |  |  | | --- | --- | --- | --- | | 组号 | 块范围 | 最后块号 | 内容 | | 1 | 1-10 | 10 | {10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20} | | 2 | 11-20 | 20 | {5, 21, 22, 23, 24, 25} | | 3 | 21-25 | 25 | {0, NULL} |   **实验总结：**  本次实验通过模拟实现Unix系统中的成组链接法，对磁盘空闲空间的管理方法进行了深入研究和实践。在实验过程中，我们按照实验要求，使用C语言在Linux环境中完成了成组链接法的设计与实现，主要功能包括磁盘空闲块的分组、查询、分配与回收，以及将分组信息写入文件和从文件中读出的操作。  在实验中遇到的主要问题包括对成组链接法数据结构的理解以及如何在代码中实现分组和链式结构。针对这些问题，我们通过对原理的详细分析和代码调试逐步解决。首先，设计了合适的二维数组来模拟磁盘空间，并使用单独的数组存储当前空闲块的分组信息。然后，通过递归和循环的结合，准确地实现了磁盘块的分配和回收，同时保证了每次操作后分组状态的正确性。  通过这次实验，我们深刻体会到成组链接法在管理大规模磁盘空闲空间中的高效性，同时也认识到细致的逻辑设计对实现复杂算法的重要性。在编程实践中，我们还学会了如何通过分步骤调试逐步排查问题，这提升了我们的代码调试能力和问题解决能力。此外，实验进一步加深了我们对操作系统中磁盘管理机制的理解，为后续学习相关内容打下了扎实的基础。 | | | | | | | | | | | | |
| **五、指导教师评语** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| **成 绩** | | |  | | **批阅人** | |  | **日 期** | | |  | |