**中央财经大学**

**实 验 报 告**

**实验项目名称**

**所属课程名称**

**实 验 类 型**

**实 验 日 期**

**班 级**

**学 号**

**姓 名**

**成 绩**

**实验室**

|  |
| --- |
| **实验概述：** |
| **【实验目的及要求】**  掌握Linux下动态异长分区分配算法的实现  **【实验原理】**  最佳适应算法：  算法思想：  最佳适应算法是一种按照内存块大小递增的顺序搜索可用空闲区域的算法。该算法在搜索到所有可用空闲区域后，选择最小的满足请求的空闲区域分配给进程。  算法步骤：  1. 初始化内存分区，将整个内存划分为一个个大小不同的分区。  2. 当进程请求内存时，遍历所有可用的空闲区域，找到大小最小且大于等于请求大小的空闲区域。  3. 如果找到合适的空闲区域，则将该区域分配给进程，并将该区域从空闲区域列表中删除。  4. 如果找不到合适的空闲区域，则等待内存中有空闲区域出现。  最坏适应算法：  算法思想：  最坏适应算法是一种按照内存块大小递减的顺序搜索可用空闲区域的算法。该算法在搜索到所有可用空闲区域后，选择最大的满足请求的空闲区域分配给进程。  算法步骤：  1. 初始化内存分区，将整个内存划分为一个个大小不同的分区。  2. 当进程请求内存时，遍历所有可用的空闲区域，找到大小最大且大于等于请求大小的空闲区域。  3. 如果找到合适的空闲区域，则将该区域分配给进程，并将该区域从空闲区域列表中删除。  4. 如果找不到合适的空闲区域，则等待内存中有空闲区域出现。  总结：  最佳适应算法和最坏适应算法都是基于内存块大小的递增或递减顺序来搜索可用空闲区域，以选择最小或最大的满足请求的空闲区域分配给进程。最佳适应算法选择最小的空闲区域，最坏适应算法选择最大的空闲区域。两种算法都有各自的优缺点，需要根据实际情况选择合适的算法。  **【实验环境】（使用的软件）**  **Linux虚拟机，gcc编译器** |
| **实验内容：** |
| **【实验方案设计】**  **实验一：**  #ifdef HAVE\_CONFIG\_H  #include<config.h>  #endif  #include<stdio.h>  #include<stdlib.h>  #define MAPSIZE 100  struct map //空闲区域表项结构  {  int m\_addr; //空闲区域首址  int m\_size; //空闲区域长度  };  struct map map[MAPSIZE]; //空闲区域表  //最佳分配算法，mp为空闲区域表的首地址，size为所申请空闲分区的长度  int BF\_malloc(struct map \*mp,int size) {  register int a,s; // a用来记录每个分区的首地址，s用来记录每个分区的大小  register struct map \*bp,\*bpp;  for(bp=mp;bp->m\_size;bp++)  {  if(bp->m\_size>=size) //先找到第一个可满足的空闲分区  {  a=bp->m\_addr;  s=bp->m\_size;  //遍历空闲区域表找到可满足的最小分区  for(bpp=bp;bpp->m\_size;bpp++){  if(bpp->m\_size>=size&&bpp->m\_size<s)  {  a=bpp->m\_addr;  s=bpp->m\_size;  bp=bpp;  }  }  bp->m\_addr+=size; //找到可满足的最小分区后，改变其首地址  if((bp->m\_size-=size)==0) //如果所找到的最小分区的大小等于所请求的大小  do //则删除该分区，并将其后的空闲分区依次向前移  {  bp++;  (bp-1)->m\_addr=bp->m\_addr;  }while((bp-1)->m\_size=bp->m\_size);  return(a); //返回所找到最适合的最小空闲分区的首地址  }//end if  }//end for  return(-1);  }  //最坏分配算法 mp为空闲区域表的首地址，size为所申请空闲分区的长度  int WF\_malloc(struct map \*mp,int size)  {  register int a,s;  register struct map \*bp,\*bpp;  for(bp=mp;bp->m\_size;bp++)  {  if(bp->m\_size>=size) //先找到第一个可满足的空闲分区  {  a=bp->m\_addr;  s=bp->m\_size;  //遍历空闲区域表找到可满足的最大分区  for(bpp=bp;bpp->m\_size;bpp++)  {  if(bpp->m\_size>s)  {  a=bpp->m\_addr;  s=bpp->m\_size;  bp=bpp;  }  }  bp->m\_addr+=size; //找到可满足的最大分区后，改变其首地址  if((bp->m\_size-=size)==0) //如果所找到的最大分区的大小等于所请求的大小  do {//则删除该分区，并将其后的空闲分区依次向前移  bp++;  (bp-1)->m\_addr=bp->m\_addr;  }while((bp-1)->m\_size=bp->m\_size);  return (a); //返回所找到的最大空闲分区的首地址  }//end if  }//end for  return (-1);  }  //分区的释放mp为空闲区域表的首地址, aa为被释放分区的首地址, size为被释放分区的  //长度  void mfree(struct map \*mp,int aa,int size)  {  register struct map \*bp;  register int t;  register int a;  a=aa;  //在空闲区域表中找到首地址大于并且最接近a的空闲分区首地址bp  for(bp=mp;bp->m\_addr<=a&&bp->m\_size!=0;bp++);  if(bp>mp&&(bp-1)->m\_addr+(bp-1)->m\_size==a)//如果bp-1首地址加上空间长度刚好等  //于a，即bp-1和a所代表的两空闲分区是相邻的  {  (bp-1)->m\_size+=size;//将bp-1和a所代表的两空闲分区合并  //如果a和bp所代表的两空闲分区相邻，则将bp-1,a和bp合并，并将bp后的分区依//次向前移  if(a+size==bp->m\_addr)  {  (bp-1)->m\_size+=bp->m\_size;  while(bp->m\_size)  {  bp++;  (bp-1)->m\_addr=bp->m\_addr;  (bp-1)->m\_size=bp->m\_size;  }  }  }  else{ //若bp-1和a所代表的两空闲分区不相邻  //若a和bp所代表的两空闲分区相邻，将a和bp合并  if(a+size==bp->m\_addr&&bp->m\_size){  bp->m\_addr-=size;  bp->m\_size+=size;  }  else if (size) //否则不合并，将a这一新的空闲分区插入  do{  t=bp->m\_addr;  bp->m\_addr=a;  a=t;  t=bp->m\_size;  bp->m\_size=size;  bp++;  }while(size=t);  }  }  void init(){ //空闲区域表的初始化  struct map \*bp;  int addr, size;  int i=0;  bp=map;  printf("please input starting addr and total size(addr,size):");  scanf("%d, %d", &addr, &size);  bp->m\_addr=addr;  bp->m\_size=size;  (++bp)->m\_size=0; //表尾  }  void show\_map() //打印空闲区域表  {  int i=0;  struct map \*bp;  bp=map;  printf("\nCurrent memory map...\n");  printf("Address \t\tSize");  while(bp->m\_size!=0) {  printf("<%d\t\t%d>\n",bp->m\_addr,bp->m\_size);  bp++;  }  printf("\n");  }  main()  {  int a,s;  char c;  int i;  init();    printf("Please input b for BF, w for WF:");  getchar();  scanf("%c",&c);    do  {  show\_map();  printf("Please input 1 for request, 2 for release, 0 for exit:");  scanf("%d",&i);  switch(i)  {  case 1:  printf("Please input size:");  scanf("%d",&s);  if(c=='b')  a=BF\_malloc(map,s);  else  a=WF\_malloc(map,s);  if(a==-1)  printf("request can't be satisfied\n");  else  printf("alloc memory at address:%d,size:%d\n",a,s);  break;  case 2:  printf("Please input addr and size(addr,size):");  scanf("%d,%d",&a,&s);  mfree(map,a,s);  break;  case 0:  exit(0);  }  } while(1);  }  **实验二：**  #ifdef HAVE\_CONFIG\_H  #include<config.h>  #endif  #include<stdio.h>  #include<stdlib.h>  #define MAPSIZE 100  struct map //空闲区域表项结构  {  int m\_addr; //空闲区域首址  int m\_size; //空闲区域长度  };  struct map map[MAPSIZE]; //空闲区域表  //最佳分配算法，mp为空闲区域表的首地址，size为所申请空闲分区的长度  int BF\_malloc(struct map \*mp,int size) {  register int a,s;  register struct map \*bp,\*bpp;  for(bp=mp;bp->m\_size;bp++) {  if(bp->m\_size>=size) {  a=bp->m\_addr;  s=bp->m\_size;  for(bpp=bp;bpp->m\_size;bpp++) {  if(bpp->m\_size>=size&&bpp->m\_size<s) {  a=bpp->m\_addr;  s=bpp->m\_size;  bp=bpp;  }  }  bp->m\_addr+=size;  if((bp->m\_size-=size)==0) {  do {  bp++;  (bp-1)->m\_addr=bp->m\_addr;  } while((bp-1)->m\_size=bp->m\_size);  }  // 新增代码，按空闲区域长度从小到大排序  struct map tmp;  for (struct map \*p = mp; p->m\_size; p++) {  for (struct map \*q = p+1; q->m\_size; q++) {  if (p->m\_size > q->m\_size) {  tmp = \*p;  \*p = \*q;  \*q = tmp;  }  }  }  return(a);  }  }  return(-1);  }  //最坏分配算法 mp为空闲区域表的首地址，size为所申请空闲分区的长度  int WF\_malloc(struct map \*mp,int size) {  register int a,s;  register struct map \*bp,\*bpp;  for(bp=mp;bp->m\_size;bp++) {  if(bp->m\_size>=size) {  a=bp->m\_addr;  s=bp->m\_size;  for(bpp=bp;bpp->m\_size;bpp++) {  if(bpp->m\_size>s) {  a=bpp->m\_addr;  s=bpp->m\_size;  bp=bpp;  }  }  bp->m\_addr+=size;  if((bp->m\_size-=size)==0) {  do {  bp++;  (bp-1)->m\_addr=bp->m\_addr;  } while((bp-1)->m\_size=bp->m\_size);  }  // 新增代码，按空闲区域长度从大到小排序  struct map tmp;  for (struct map \*p = mp; p->m\_size; p++) {  for (struct map \*q = p+1; q->m\_size; q++) {  if (p->m\_size < q->m\_size) {  tmp = \*p;  \*p = \*q;  \*q = tmp;  }  }  }  return(a);  }  }  return(-1);  }  //分区的释放mp为空闲区域表的首地址, aa为被释放分区的首地址, size为被释放分区的  //长度  void mfree(struct map \*mp,int aa,int size)  {  register struct map \*bp;  register int t;  register int a;  a=aa;  //在空闲区域表中找到首地址大于并且最接近a的空闲分区首地址bp  for(bp=mp;bp->m\_addr<=a&&bp->m\_size!=0;bp++);  if(bp>mp&&(bp-1)->m\_addr+(bp-1)->m\_size==a)//如果bp-1首地址加上空间长度刚好等  //于a，即bp-1和a所代表的两空闲分区是相邻的  {  (bp-1)->m\_size+=size;//将bp-1和a所代表的两空闲分区合并  //如果a和bp所代表的两空闲分区相邻，则将bp-1,a和bp合并，并将bp后的分区依//次向前移  if(a+size==bp->m\_addr)  {  (bp-1)->m\_size+=bp->m\_size;  while(bp->m\_size)  {  bp++;  (bp-1)->m\_addr=bp->m\_addr;  (bp-1)->m\_size=bp->m\_size;  }  }  }  else{ //若bp-1和a所代表的两空闲分区不相邻  //若a和bp所代表的两空闲分区相邻，将a和bp合并  if(a+size==bp->m\_addr&&bp->m\_size){  bp->m\_addr-=size;  bp->m\_size+=size;  }  else if (size) //否则不合并，将a这一新的空闲分区插入  do{  t=bp->m\_addr;  bp->m\_addr=a;  a=t;  t=bp->m\_size;  bp->m\_size=size;  bp++;  }while(size=t);  }  }  void init(){ //空闲区域表的初始化  struct map \*bp;  int addr, size;  int i=0;  bp=map;  printf("please input starting addr and total size(addr,size):");  scanf("%d, %d", &addr, &size);  bp->m\_addr=addr;  bp->m\_size=size;  (++bp)->m\_size=0; //表尾  }  void show\_map() //打印空闲区域表  {  int i=0;  struct map \*bp;  bp=map;  printf("\nCurrent memory map...\n");  printf("Address \t\tSize");  while(bp->m\_size!=0) {  printf("<%d\t\t%d>\n",bp->m\_addr,bp->m\_size);  bp++;  }  printf("\n");  }  main()  {  int a,s;  char c;  int i;  init();    printf("Please input b for BF, w for WF:");  getchar();  scanf("%c",&c);    do  {  show\_map();  printf("Please input 1 for request, 2 for release, 0 for exit:");  scanf("%d",&i);  switch(i)  {  case 1:  printf("Please input size:");  scanf("%d",&s);  if(c=='b')  a=BF\_malloc(map,s);  else  a=WF\_malloc(map,s);  if(a==-1)  printf("request can't be satisfied\n");  else  printf("alloc memory at address:%d,size:%d\n",a,s);  break;  case 2:  printf("Please input addr and size(addr,size):");  scanf("%d,%d",&a,&s);  mfree(map,a,s);  break;  case 0:  exit(0);  }  } while(1);  }  **【实验过程】（实验步骤、记录、数据、分析）**  实验一数据结构说明：  程序中使用了一个结构体map来表示空闲区域表项，空闲区域表项包含空闲区域的起始地址和长度。空闲区域表使用一个map数组来存储空闲区域表项，数组的最后一项的长度为0，表示空闲区域表的结尾。  实验二数据结构说明：  程序中使用了一个结构体map来表示空闲区域表项，空闲区域表项包含空闲区域的起始地址和长度。空闲区域表使用一个map数组来存储空闲区域表项，数组的最后一项的长度为0，表示空闲区域表的结尾。在最佳适应分配函数和最坏适应分配函数中，新增了一个排序的步骤，将空闲区域表按照空闲区域长度从小到大或从大到小进行排序。  **【结论】（结果）**    **图一**    **图二** |
| **【小结】**   1. 程序运行结果：   程序运行后，先输入起始地址和总大小，然后选择最佳适应算法或最坏适应算法。接着，按照提示输入1进行申请空间，输入2进行释放空间，输入0退出程序。程序会打印当前的空闲区域表。   1. 原因分析：   该程序实现了动态分区分配和释放算法，其中包括最佳适应算法和最坏适应算法。空闲区域表用结构体数组实现，每个元素包含空闲区域的起始地址和大小。申请空间时，根据算法选择最优的空闲区域进行分配，并按照空闲区域大小排序。释放空间时，会将空闲区域表中相邻的空闲区域进行合并，同时按照起始地址排序。   1. 实验体会及收获：   通过这个实验，我深刻理解了动态分区分配和释放算法的原理和实现方式。同时，我也学会了如何使用结构体数组实现空闲区域表，并且掌握了排序算法的实现方法。这个实验让我对内存管理有了更深入的了解，也提高了我的编程能力和算法实现能力。   1. 建议：   在实验中，程序只提供了最佳适应算法和最坏适应算法两种选择，可以考虑增加其他的算法，如首次适应算法和循环首次适应算法等。同时，可以进一步优化算法实现，提高程序的效率和稳定性。 |
| **指导教师评语及成绩：** |
| **评语：**  **成绩： 指导教师签名：**  **批阅日期：** |

实验报告说明

**1．实验项目名称：**要用最简练的语言反映实验的内容。要求与实验指导书中相一致。

**2．实验类型：**一般需说明是验证型实验还是设计型实验，是创新型实验还是综合型实验。

**3．实验目的与要求**：目的要明确，要抓住重点，符合实验指导书中的要求。

**4．实验原理：**简要说明本实验项目所涉及的理论知识。

**5．实验环境**：实验用的软硬件环境（配置）。

**6．实验方案设计（思路、步骤和方法等）**：这是实验报告极其重要的内容。概括整个实验过程。

对于**操作型实验**，要写明依据何种原理、操作方法进行实验，要写明需要经过哪几个步骤来实现其操作。

对于**设计型和综合型实验**，在上述内容基础上还应该画出流程图、设计思路和设计方法，再配以相应的文字说明。

对于**创新型实验**，还应注明其创新点、特色。

**7．实验过程（实验中涉及的记录、数据、分析）：**写明具体上述实验方案的具体实施，包括实验过程中的记录、数据和相应的分析。

**8．结论（结果）：**即根据实验过程中所见到的现象和测得的数据，做出结论。

**9．小结：**对本次实验的心得体会、思考和建议。

**10．指导教师评语及成绩：**指导教师依据学生的实际报告内容，用简练语言给出本次实验报告的评价和价值。

**注意：**

* 实验报告将记入实验成绩；
* 每次实验开始时，交上一次的实验报告，否则将扣除此次实验成绩。