**重 庆 大 学**

**学 生 实 验 报 告**

**实验课程名称 操作系统**

**开课实验室 DS1501**

**学 院 大数据与软件学院 年级 20 专业 软工 1班**

**学 生 姓 名 李易燔 学 号 20205644**

**开 课 时 间 2021 至 2022 学年第 2 学期**

|  |  |
| --- | --- |
| **成 绩** |  |
| **教师签名** |  |

**大数据与软件学院制**

**《操作系统》实验报告**

**开课实验室：DS15021 2021年3月28日**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 学院 | | 大数据与软件学院 | 年级、专业、班 | | 20级软工1班 | 姓名 | 李易燔 | | 成绩 |  | |
| 课程  名称 | | 操作系统 | | 实验项目  名 称 | 线程的创建 | | | 指导教师 | | | 刘寄 |
| 教师评 | 教师签名：  2022年 月 日 | | | | | | | | | | |
| * **实验目的**   掌握线程的创建  **二、实验内容**  随机生成N组非负整数列表，然后创建N个线程，分别用N种不同的排序算法对列表进行排序   * 创建   + int task\_create(void \*tos, void (\*func)(void \*pv), void \*pv)     - tos ：用户栈的栈顶指针     - func ：线程函数     - pv ：传递给线程函数func的参数     - 返回值 ：大于0，则表示新创建线程之ID * 退出   + int task\_exit(int code\_exit);     - code\_exit ：线程的退出代码 * 获取线程自己的ID   + int task\_getid(); * 等待线程退出   + int task\_wait(int tid, int \*pcode\_exit);     - tid ：要等待线程之ID     - pcode\_exit ：如果非NULL，用于保存线程tid的退出代码 * Step1：定义线程函数 * Step2：申请线程栈   + - * 线程退出后，才能把用户栈用free释放掉！ * Step3：创建线程   Main文件  /\*  \* vim: filetype=c:fenc=utf-8:ts=4:et:sw=4:sts=4  \*/  #include <inttypes.h>  #include <stddef.h>  #include <math.h>  #include <stdio.h>  #include <sys/mman.h>  #include <syscall.h>  #include <netinet/in.h>  #include <stdlib.h>  #include "graphics.h"  #include<time.h>  extern void \*tlsf\_create\_with\_pool(void\* mem, size\_t bytes);  extern void \*g\_heap;  /\*\*  \* GCC insists on \_\_main  \* http://gcc.gnu.org/onlinedocs/gccint/Collect2.html  \*/  void \_\_main()  {  size\_t heap\_size = 32\*1024\*1024;  void \*heap\_base = mmap(NULL, heap\_size, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANON, -1, 0);  g\_heap = tlsf\_create\_with\_pool(heap\_base, heap\_size);  }  //画线自定义函数  void drawLine(int x1, int y1, int x2, int y2, int extra\_x, COLORREF cr)  {  line(x1+extra\_x,(y1/5)\*3,x2+extra\_x,(y2/5)\*3,cr);  }  //睡眠函数  void mySleep()  {  struct timespec tim, tim2;  tim.tv\_sec = 0;  tim.tv\_nsec = 10000000;  nanosleep(&tim, &tim2);  }  //第一种排序算法：插入排序  void insertSort(int\* data,int n)  {  int i,j;    for(i=1;i<n;i++)  {  int temp=data[i];  for(j=i;j>0&&data[j-1]>temp;j--)  {  drawLine(0,j\*6+60,data[j],j\*6+60,0,RGB(0,0,0));//覆盖原来的线  data[j]=data[j-1];  drawLine(0, j \* 6 + 60, data[j], j \* 6 + 60, 0, 0x4682B4);  }  drawLine(0, j \* 6 + 60, data[j] , j \* 6 + 60,0,RGB(0,0,0));  data[j]=temp;  drawLine(0, j \* 6 + 60, data[j], j \* 6 + 60, 0, 0x4682B4);  mySleep();  }  }  //第二种 选择排序  void selsort(int \*a, int n)  {  int i, j,low,temp;  for (i = 0; i < n - 1; i++)  {  low = i;  for (j = i; j < n; j++)  {  if (a[low] > a[j])  low = j;  }  //进行交换，先抹掉原来线段  drawLine(0, low \* 6 + 60, a[low], low \* 6 + 60, 150, RGB(0, 0, 0));  drawLine(0, i \* 6 + 60, a[i], i \* 6 + 60, 150, RGB(0, 0, 0));  temp = a[low];  a[low] = a[i];  a[i] = temp;  drawLine(0, low \* 6 + 60, a[low], low \* 6 + 60, 150, 0x4169E1);  drawLine(0, i \* 6 + 60, a[i], i \* 6 + 60, 150, 0x4169E1);  mySleep();  }  }  //第三种排序，冒泡排序  void bubsort(int\*a, int n)  {  int i, j, temp;  for (i = 0; i < n; i++)  {  for (j = 0; j < n - i - 1; j++)  {  if (a[j] > a[j + 1])  {  //覆盖排序前的两条线段  drawLine(0, j \* 6 + 60, a[j], j \* 6 + 60, 300, RGB(0, 0, 0));  drawLine(0, (j+1) \* 6 + 60, a[j+1], (j+1) \* 6 + 60, 300, RGB(0, 0, 0));  temp = a[j];  a[j] = a[j + 1];  a[j + 1] = temp;  drawLine(0, j \* 6 + 60, a[j], j \* 6 + 60, 300, 0x556B2F);  drawLine(0, (j + 1) \* 6 + 60, a[j + 1], (j + 1) \* 6 + 60, 300, 0x556B2F);  mySleep();  }  }  }  }  //第四种排序,快速排序  void quicksort(int \*a, int left, int right)  {  if (left >= right)  return;  int j = right, l = left, r = right,temp;  while (l < r)  {  while (a[l] <= a[j] && l < r)l++;  while (l < r && a[r] >= a[j])r--;  //swap(a[l], a[r]);  drawLine(0, l \* 6 + 60, a[l], l \* 6 + 60, 450, RGB(0, 0, 0));  drawLine(0, r\* 6 + 60, a[r], r \* 6 + 60, 450, RGB(0, 0, 0));  temp = a[l];  a[l] = a[r];  a[r] = temp;  drawLine(0, l \* 6 + 60, a[l], l \* 6 + 60, 450, 0xFFFF00);  drawLine(0, r \* 6 + 60, a[r], r \* 6 + 60, 450, 0xFFFF00);  mySleep();  }  //swap(a[j], a[l]);  drawLine(0, l \* 6 + 60, a[l], l \* 6 + 60, 450, RGB(0, 0, 0));  drawLine(0, j \* 6 + 60, a[j], j \* 6 + 60, 450, RGB(0, 0, 0));  temp = a[l];  a[l] = a[j];  a[j] = temp;  drawLine(0, l \* 6 + 60, a[l], l \* 6 + 60, 450, 0xFFFF00);  drawLine(0, j \* 6 + 60, a[j], j \* 6 + 60, 450, 0xFFFF00);  mySleep();  quicksort(a, left, l - 1);  quicksort(a, l + 1, right);  }  //线程函数  void tsk\_foo1(void \*pv)  {  time\_t time(time\_t \*loc);  srand(time(NULL));  int myCount\_1[150];  int i,k;  for(i=0;i<150;i++)  {  myCount\_1[i]=rand()%150;  printf("%d\n",myCount\_1[i]);  }    //显示未排序的画面  for(k=0;k<150;k++)  {  drawLine(0,k\*6+60,myCount\_1[k],k\*6+60,0, 0x4682B4);  }  mySleep();  insertSort(myCount\_1, 150);  task\_exit(0);  }  void tsk\_foo2(void\* pv)  {  time\_t time(time\_t \* loc);  srand(time(NULL));  int myCount\_2[150];  int i, k;  for (i = 0; i < 150; i++)  {  myCount\_2[i] = rand() % 150;  printf("%d\n", myCount\_2[i]);  }  //显示未排序的画面  for (k = 0; k < 150; k++)  {  drawLine(0, k \* 6 + 60, myCount\_2[k], k \* 6 + 60,150, 0x4169E1);  }  mySleep();  selsort(myCount\_2, 150);  task\_exit(0);  }  void tsk\_foo3(void\* pv)  {  time\_t time(time\_t \* loc);  srand(time(NULL));  int myCount\_3[150];  int i, k;  for (i = 0; i < 150; i++)  {  myCount\_3[i] = rand() % 150;  printf("%d\n", myCount\_3[i]);  }  //显示未排序的画面  for (k = 0; k < 150; k++)  {  drawLine(0, k \* 6 + 60, myCount\_3[k], k \* 6 + 60, 300, 0x556B2F);  }  mySleep();  bubsort(myCount\_3, 150);  task\_exit(0);  }  void tsk\_foo4(void\* pv)  {  time\_t time(time\_t \* loc);  srand(time(NULL));  int myCount\_4[150];  int i, k;  for (i = 0; i < 150; i++)  {  myCount\_4[i] = rand() % 150;  printf("%d\n", myCount\_4[i]);  }  //显示未排序的画面  for (k = 0; k < 150; k++)  {  drawLine(0, k \* 6 + 60, myCount\_4[k], k \* 6 + 60, 450, 0xFFFF00);  }  mySleep();  quicksort(myCount\_4,0,149);  task\_exit(0);  }  /\*\*  \* 第一个运行在用户模式的线程所执行的函数  \*/  void main(void \*pv)  {  unsigned char \*stack\_foo\_1,\*stack\_foo\_2, \* stack\_foo\_3,\* stack\_foo\_4;  unsigned int stack\_size = 1024\*1024;  stack\_foo\_1 = (unsigned char \*)malloc(stack\_size );  stack\_foo\_2=(unsigned char\*)malloc(stack\_size);  stack\_foo\_3 = (unsigned char\*)malloc(stack\_size);  stack\_foo\_4 = (unsigned char\*)malloc(stack\_size);    init\_graphic(0x143);  task\_create(stack\_foo\_1+stack\_size,&tsk\_foo1,(void \*)0);  task\_create(stack\_foo\_2 + stack\_size, &tsk\_foo2, (void\*)0);  task\_create(stack\_foo\_3 + stack\_size, &tsk\_foo3, (void\*)0);  task\_create(stack\_foo\_4 + stack\_size, &tsk\_foo4, (void\*)0);  free(stack\_foo\_1);  free(stack\_foo\_2);  free(stack\_foo\_3);  free(stack\_foo\_4);  while(1)  ;  task\_exit(0);  }  实验结果： | | | | | | | | | | | |