**重 庆 大 学**

**学 生 实 验 报 告**

**实验课程名称 操作系统原理**

**开课实验室 重庆大学DS1501**

**学 院 软件工程 年级 20软件工程 专业班 1**

**学 生 姓 名 李易燔 学 号 20205644**

**开 课 时 间 2021 至 2022 学年第 二 学期**

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

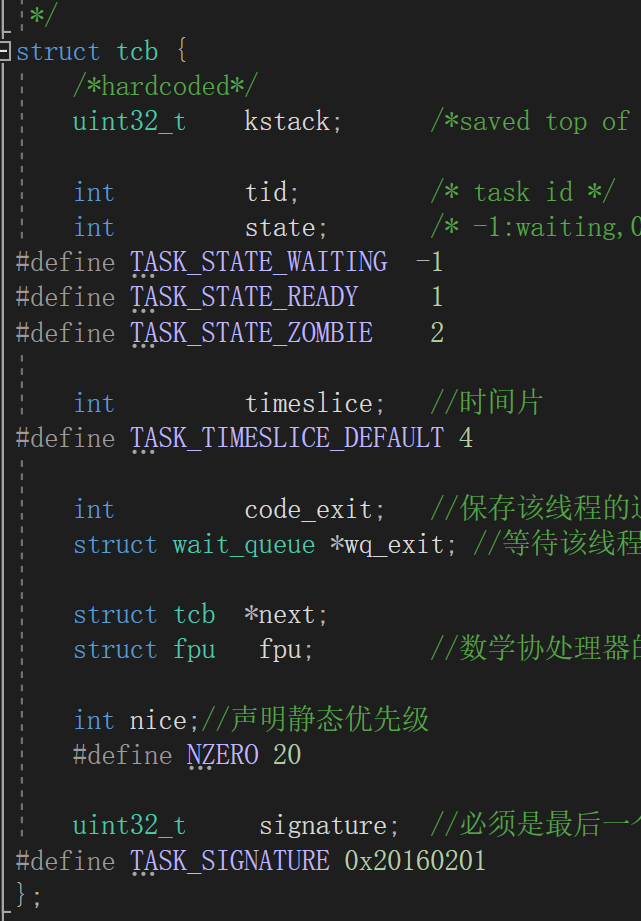
**重庆大学软件 学院制**

**《操作系统原理》实验报告**

**开课实验室：DS1501 2022 年 3 月30日**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 学院 | | 软件学院 | 年级、专业、班 | | 20软件工程1班 | 姓名 | 李易燔 | | 成绩 | |  |
| 课程  名称 | | 操作系统原理 | | 实验项目  名 称 | 动态优先级线程调度 | | | 指导教师 | | 刘寄 | |
| 教师评语 | 教师签名：  年 月 日 | | | | | | | | | | |
| * **实验目的（软件需求文档）**   掌握线程的创建  掌握线程的调度   * + 动态优先级调度   **二、实验原理（软件设计文档）**  系统调用接口，线程相关函数：   * **Step1：在“struct tcb”中增加线程的静态优先级nice** * **Step2：增加系统调用**   + **int getpriority(int tid)**   + **int setpriority(int tid, int prio)** * **Step3：在“struct tcb”中，再增加两个属性**   + **estcpu：表示线程最近使用了多少CPU时间**   + **priority：表示线程的动态优先级** * **Step4：增加一个全局变量** * **Step5：属性计算** * **Step6：修改task.c中的函数schedule，实现优先级调度算法** * **Step5：测试调度器**   + **创建两个冒泡排序的线程，记为A和B**   + **另外创建一个控制线程** | | | | | | | | | | | |

1. 静态优先级
   1. 在Kernel.c中添加静态优先级相关变量



* 1. 在task.c中的sys\_task\_create函数里初始化变量



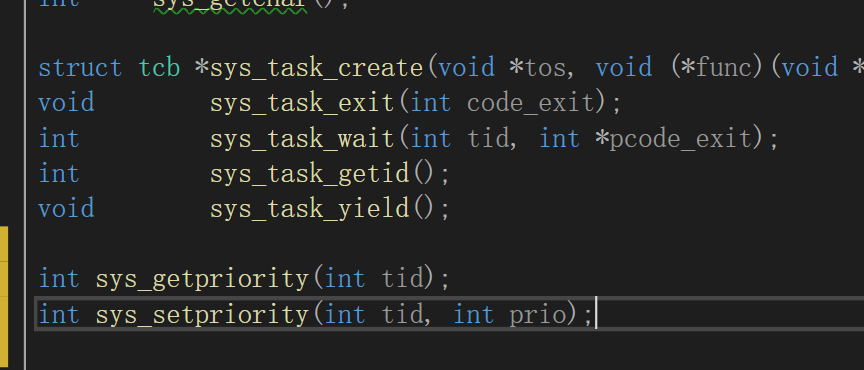
* 1. 在task.c中添加获取优先级函数

//获取当前线程优先级  
int sys\_getpriority(int tid) {  
    uint32\_t flags  
        struct tcb\* task;  
    save\_flags\_cli(flags);  
    //tid为零则获取当前线程  
    if (tid == 0) {  
        tsk = g\_task\_running;  
    }//不为零则获取指定线程  
    else {  
        tsk = get\_task(tid);  
    }  
    restore\_flags(flags);  
    if (tsk != NULL)return tsk->nice + NZERO;  
    else return -1;  
}

* 1. 在task.c中添加设置优先级函数

//设置优先级  
int sys\_setpriority(int tid, int prio) {  
    uint32\_t flags;  
    struct tcb\* tsk;  
    //prio超出范围则设置失败  
    if (prio < 0 || prio>2 \* NZERO - 1) {  
        return -1;  
    }  
    save\_flags\_cli(flags);  
    if (tid == 0) {  
        tsk = g\_task\_running;  
    }  
    else {  
        tsk = get\_task(tid);  
    }  
    restore\_flags(flags);  
    if (tsk != NULL) {  
        tsk->nice = prio - NZERO;  
        return 0;  
    }  
    else {  
        return -1;  
    }  
}

* 1. 在kernel.h中进行声明上述函数



* 1. 在syscall-nr.h中定义这两个函数的系统调用号

#define SYSCALL\_getpriority 11  
#define SYSCALL\_setpriority 12

* 1. 在machdep.c中的syscall(struct context \*ctx)函数中增加系统调用的分支

        //实验三  
    case SYSCALL\_getpriority: {  
        int tid = \*(int\*)(ctx->esp + 4);  
        ctx->eax = sys\_getpriority(tid);  
    }break;  
    case SYSCALL\_setpriority: {  
        int tid = \*(int\*)(ctx->esp + 4);  
        int prio = \*(int\*)(ctx->esp + 8);  
        ctx->eax = sys\_setpriority(tid, prio);  
    }break;

* 1. 用户态下的汇编接口，在syscall-wrapper.S文件中声明上述函数



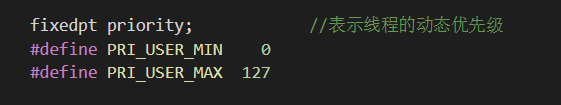
* 1. 最后在syscall.h中声明这两个系统调用函数

//实验三  
int getpriority(int tid);  
int setpriority(int tid, int prio);

1. 动态优先级
   1. 在Kernel.c中添加静态优先级相关变量
      1. Estcpu

fixedpt estcpu; //表示线程最近花了多少时间

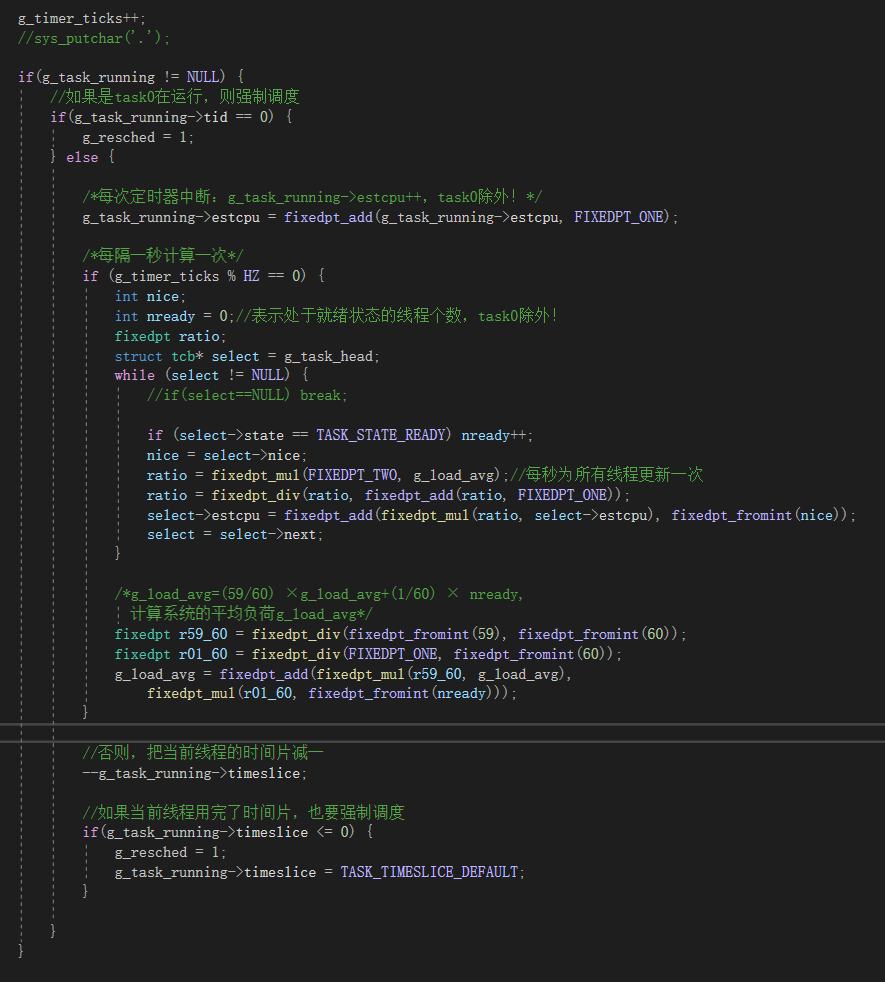
* + 1. Priority



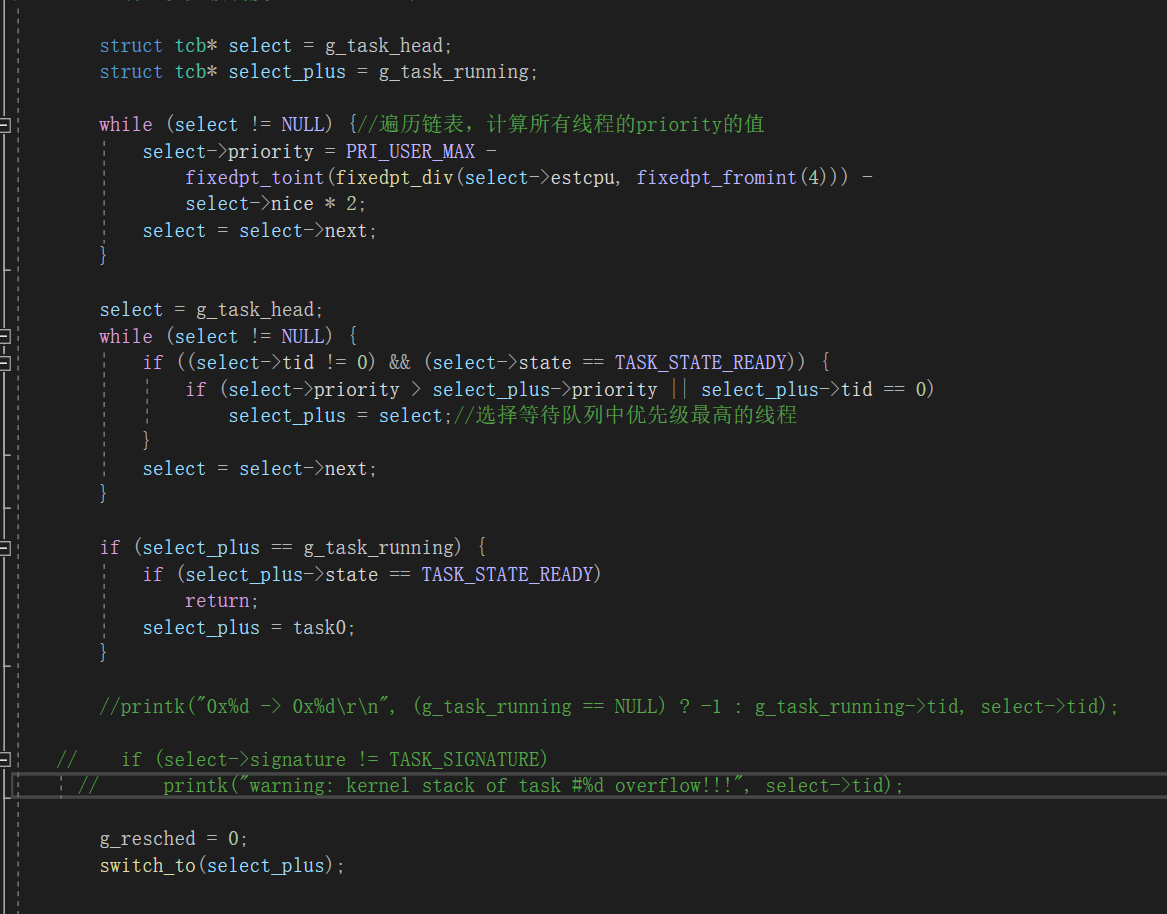
* 1. 在task.c中的sys\_task\_create函数里初始化变量

    new->estcpu = 0;  
    new->priority = 0;

* 1. 在timer.c中isr\_timer(uint32\_t irq, struct context \*ctx)函数编写estcpu以及priority处理代码



* 1. 修改task.c中的函数schedule，实现优先级调度算法



1. 验证静态优先级

/\*

\* 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);

}

//线程优先级

int tid\_foo1, tid\_foo2, tid\_foo3;

//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 = 1;

tim.tv\_nsec = 10000000;

nanosleep(&tim, &tim2);

}

//冒泡排序

void bubsort1(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(-a[j], j \* 6 , 0, j \* 6, 250, RGB(0, 0, 0));

drawLine(-a[j + 1], (j + 1) \* 6 ,0, (j + 1) \* 6 , 250, RGB(0, 0, 0));

temp = a[j];

a[j] = a[j + 1];

a[j + 1] = temp;

drawLine(-a[j], j \* 6 , 0, j \* 6 , 250, 0x4682B4);

drawLine(-a[j + 1], (j + 1) \* 6 , 0, (j + 1) \* 6 , 250, 0x4682B4);

//mySleep();

}

//mySleep();

}

mySleep();

}

}

void bubsort2(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 , a[j], j \* 6 , 250, RGB(0, 0, 0));

drawLine(0, (j + 1) \* 6 , a[j + 1], (j + 1) \* 6 , 250, RGB(0, 0, 0));

temp = a[j];

a[j] = a[j + 1];

a[j + 1] = temp;

drawLine(0, j \* 6 , a[j], j \* 6 , 250, 0xFFFF00);

drawLine(0, (j + 1) \* 6 , a[j + 1], (j + 1) \* 6 , 250, 0xFFFF00);

//mySleep();

}

//mySleep();

}

mySleep();

}

}

//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() % 200;

printf("%d\n", myCount\_1[i]);

}

//显示未排序的画面

for (k = 0; k < 150; k++)

{

drawLine(-myCount\_1[k], k \* 6 ,0, k \* 6 , 250, 0x4682B4);

}

mySleep();

bubsort1(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() % 200;

printf("%d\n", myCount\_2[i]);

}

//显示未排序的画面

for (k = 0; k < 150; k++)

{

drawLine(0, k \* 6 , myCount\_2[k], k \* 6 , 250, 0xFFFF00);

}

mySleep();

bubsort2(myCount\_2, 150);

task\_exit(0);

}

//优先级展示条

void show\_priority(int tid, int k) {

int length = 10 \* getpriority(tid);

int m = 0;

switch (k)

{

case 1:

{

for (m = 0; m < 20; m++)

drawLine(-250, 910+m, 0, 910+m, 250, RGB(0, 0, 0));

for (m = 0; m < 20; m++)

drawLine(-length, 910 + m, 0, 910 + m, 250, 0x4682B4);

}break;

case 2:

{

for (m = 0; m < 20; m++)

drawLine(0, 910 + m, 200, 910 + m, 250, RGB(0, 0, 0));

for (m = 0; m < 20; m++)

drawLine(0, 910 + m, length, 910 + m, 250, 0xFFFF00);

}break;

default:

break;

}

}

//控制线程

void mytask\_control(void\* pv) {

show\_priority(tid\_foo1,1);

show\_priority(tid\_foo2, 2);

int mykeypress;

while (1) {

mykeypress = getchar();

switch (mykeypress)

{

case 0x4800://(up)

{

setpriority(tid\_foo1, getpriority(tid\_foo1) + 2);

show\_priority(tid\_foo1, 1);

}

break;

case 0x5000://(down)

{

setpriority(tid\_foo1, getpriority(tid\_foo1)-2);

show\_priority(tid\_foo1, 1);

}

break;

//0x4d00(right)/0x4b00(left)

case 0x4d00:

{

setpriority(tid\_foo2, getpriority(tid\_foo2) + 2);

show\_priority(tid\_foo2, 2);

}

break;

case 0x4b00:

{

setpriority(tid\_foo2, getpriority(tid\_foo2) - 2);

show\_priority(tid\_foo2, 2);

}

break;

default:

break;

}

}

}

void main(void\* pv)

{

unsigned char\* stack\_foo\_1, \* stack\_foo\_2, \* stack\_foo\_3;

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);

init\_graphic(0x143);

tid\_foo1=task\_create(stack\_foo\_1 + stack\_size, &tsk\_foo1, (void\*)0);

tid\_foo2=task\_create(stack\_foo\_2 + stack\_size, &tsk\_foo2, (void\*)0);

tid\_foo3 = task\_create(stack\_foo\_3 + stack\_size, &mytask\_control, (void\*)0);

setpriority(tid\_foo3, 0);

setpriority(tid\_foo1, 10);

setpriority(tid\_foo2, 10);

free(stack\_foo\_1);

free(stack\_foo\_2);

free(stack\_foo\_3);

while (1)

;

task\_exit(0);

}

最后效果图：（显示条越长，则优先级越低，相关视频在文件夹中）

