实现五态进程模型及多进程应用

个人信息

- 院系：数据科学与计算机学院
- 年纪：2018
- 姓名：王天龙
- 学号：18340168

实验题目

实现五态进程模型及多进程应用

实验目的

1. 理解5状态的进程模型
2. 掌握操作系统内核线程模型设计与实现方法
3. 掌握实现5状态的进程模型方法
4. 实现C库封装多线程服务的相关系统调用

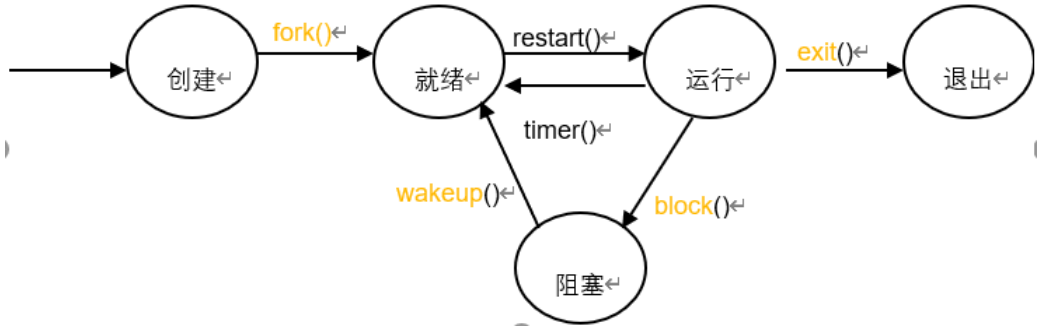
实验要求

1. 学习内核级线程模型理论，设计总体实现方案
2. 理解类unix的内核线程做法，明确全局数据、代码、局部变量的映像内容哪些共享
3. 扩展实验6的的内核程序，增加阻塞进程状态和阻塞过程、唤醒过程两个进程控制过程
4. 修改内核，提供创建线程、撤销线程和等待线程结束，实现你的线程方案
5. 4、增加创建线程、撤销线程和等待线程结束等系统调用。修改扩展C库，封装创建线程、撤销线程和等待线程结束等系统调用操作

6. 设计一个多线程应用的用户程序，展示你的多线程模型的应用效果
7. 编写实验报告，描述实验工作的过程和必要的细节，如截屏或录屏，以证实实验工作的真实性

实验内容

1. 修改内核代码，增加阻塞队列，实现5状态进程模型



2. 如果采用线程控制块，就要分离进程控制块的线程相关项目，组成线程控制块，重构进程表数据结构
3. 修改内核代码，增加阻塞block()、唤醒wakeup()、睡眠sleep()、创建线程do_fork()、撤销线程do_exit()和等待线程结束do_wait()等过程，实现你的线程控制
4. 修改扩展C库，封装创建线程**fork()**、撤销线程**exit(0)**、睡眠sleep()和等待线程结束**wait()**等系统调用操作
5. 设计一个多线程应用的用户程序，展示你的多线程模型的应用效果。示范：进程创建2个线程，分别统计全局数组中的字母和数字的出现次数。你也可以另选其他多进程应用

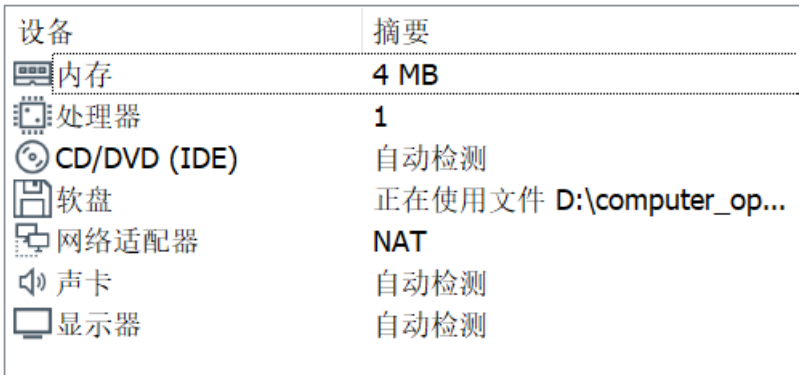
实验方案

所用工具

实验平台是windows10系统，使用的软件有：虚拟机软件vmware workstation pro、汇编语言编译器tasm、c语言编译器tcc、链接器tlink、可视化编译十六进制文件内容工具winhex、代码编辑器visual studio 2019

虚拟机配置

虚拟机为1cpu，内存为4MB，使用一个1.44MB大小的软盘，选择从软盘启动



代码概述

内核

文件名	函数名/数据结构	功能
kernel.asm	_start	内核入口
kfun.asm	_clrIF	把IF标志位设为0，关中断
	_setIF	把IF标志位设为1，开中断
	_LoadPSP	把PSP加载到程序所在段的前0x100个字节
	_LoadINT20H	设置20h号中断，功能是返回操作系统
	_LoadINT22H	设置22h号中断，功能是打印字符串INT22H
	_printc	利用BIOS调用，显示一个字符
	_readkeypress	利用BIOS调用，从键盘读一个按键
	_cleanscreen	清屏
	_loadblock	从软盘加载扇区到指定内存
	_LoadINT8H	设置8h号中断，功能是风火轮以及时间片轮转
	_save	把当前进程的寄存器保存到PCB块，保护现场
	_restart	利用PCB块恢复线程
	_LoadINT21H	设置21h号中断，提供系统调用
ckfun.c	struct cpuState	保存所有寄存器的值
	struct PCB	进程控制块
	PCB *currentPCB	指向当前正在运行的进程控制块
	PCB *kernelPCB	指向内核的进程控制块(把内核进程也视为一个普通进程)
	PCB pcbList[MaxProcessNum]	进程控制块队列
	char stackList[MaxProcessNum]	所有进程的栈段
	void strcpy(char* dest,char* source)	复制字符串
	void initPCBList()	初始化进程控制块
	void switchProcess()	切换进程控制块
	int creatProcess(int cs,int ip,int blockid,int numOfblock)	创建一个进程
	void endProcess()	系统调用，结束一个进程
	void printf(char* s)	打印一个字符串

文件名	函数名/数据结构	功能
	void scanf(char* s)	输入一个字符串
	void do_fork()	系统调用，从当前进程派生处一个子进程
	void do_exit()	系统调用，子进程退出
	void do_wait()	系统调用，父进程阻塞，等待子进程退出
	void kernelmain()	内核主函数

进程模块

文件名	函数	功能
process.asm	_fork	使用系统调用，从当前进程派生出子进程
	_exit	结束子进程，同时唤醒父进程
	_wait	父进程阻塞，等待子进程结束时唤醒

具体实现

这次实验的更新在于，有二状态进程模型升级为五状态进程模型，增加了对fork、wait和exit的支持。同时，把程序的栈段分离出来，都放在了内核的stackList里，所以，所有进程的栈的段ss相同，只有栈顶指针sp不同。其它部分就没有什么变化，在这里也就不一一赘述了。

fork的实现

这个函数定义在process.asm里，是利用内核提供的系统调用从当前进程中派生出一个子进程。返回值有-1(派生失败)、0(子进程)和1(父进程)。fork函数封装如下

```
;process.asm
_fork proc near
    mov ah,3h;使用3h号系统调用
    int 21h;从int 21h返回后，就有了两个进程开始平行，它们的ax寄存器的值不同，栈顶指针sp不同，其他的都相同。
    ret
_fork endp
```

内核中，3h号系统调用实现如下

```
;kfun.asm
SYCALL_3H proc near
    call _save;保护现场，这样在拷贝PCB块时，父子进程返回点相同

    call _do_fork;fork的具体过程用c语言实现

    jmp _restart
SYCALL_3H endp
```

do_fork函数

这个函数用c语言实现，作用时把父进程的PCB拷贝给子进程，同时把父进程的栈拷贝给子进程。这样，父子进程共享全局变量(在函数体之外声明的变量)，但是局部变量是私有的。由于子进程的PCB块是从父进程拷贝过来的，所以它和父进程有同样的返回地址，restart后，父子进程都是从int 21h后面开始执行。

```
/*kfun.c*/
char stackList[MaxProcessNum - 1][SizeOfStack];

void memcpy(void *src, void *dest, int len)/*len为要拷贝的字节数*/
{
    char *s = (char *)src;
    char *d = (char *)dest;
    int i;
    for (i = 0; i < len; i++)
    {
        d[i] = s[i];
    }
}

void do_fork()
{
    int i;
    for (i = (currentPCB->pid + 1) % MaxProcessNum; i != currentPCB->pid; i++, i
%= MaxProcessNum)/*寻找一个空闲的PCB块*/
    {
        if (pcbList[i].state == 0)/*PCB块的状态为0，说明时空闲的*/
        {
            memcpy(&pcbList[currentPCB->pid], &pcbList[i], sizeof(PCB));/*拷贝
PCB*/
            memcpy(stackList[currentPCB->pid], stackList[i], SizeOfStack);/*拷贝
栈段*/
            /*0:PCB空闲;1:进程运行;2:进程就绪;3:进程阻塞*/
            pcbList[i].state = 2;/*子进程状态先设为就绪，父进程的是运行*/
            pcbList[i].fid = currentPCB->pid;/*设置子进程的父进程id*/
            pcbList[i].pid = i;/*子进程id*/
            pcbList[i].cpu.sp += ((i - currentPCB->pid) * SizeOfStack);/*父子进程
所在的段相同，但是栈顶指针不同，两个之间相差n个SizeOfStack*/
            /*这时根据二维数组stackList[i]和stackList[j]的关系计算得来*/

            pcbList[i].cpu.ax = 0;/*子进程返回值*/

            currentPCB->cpu.ax = 1;/*父进程返回值*/
            numOfProcess++;/*进程数量增加*/
            break;
        }
    }
    if (i == currentPCB->pid)/*没有找到空闲的PCB块，创建失败*/
    {
        currentPCB->cpu.ax = -1;/*创建失败*/
    }
}
```

wait的实现

这个函数定义在process.asm中，利用系统调用，使得父进程阻塞，直到子进程退出(exit函数)时唤醒。

```
;process.asm
_exit proc near
    mov ah,5h;使用5h号系统调用
    int 21h
    ret
_exit endp
```

内核中，5h号系统调用实现如下

```
;kfun.asm
SYCALL_5H proc near
    call _save
    call _do_wait;把父进程设为阻塞，并且调用了switchProcess，切换了进程
    jmp _restart;_restart恢复的是另一个进程
SYCALL_5H endp
```

do_wait函数

```
/*ckfun.c*/
void do_wait()
{
    currentPCB->state = 3;/*把当前进程的状态设为阻塞*/
    switchProcess();/*进程调度器*/
}
```

exit的实现

这个函数定义在process.asm里，利用4h号系统调用。子进程退出，同时唤醒父进程。

```
;process.asm
_exit proc near
    mov ah,4h;使用4h号系统调用
    int 21h
    ret;其实调用4h号系统调用后，子进程就结束了，所以不会执行到这里
_exit endp
```

内核中，4h号系统调用实现如下

```
;kfun.asm
SYCALL_4H proc near
    ;这个进程是要结束的，所以不需要抱回现场
    mov ax,cs
    mov ss,ax
    mov ds,ax
    mov es,ax
    mov si,word ptr DGROUP:_kernelPCB
    mov sp,[si+20];切换到内核

    jmp _do_exit;do_exit是用c语言实现的，主要是用jmp而不用call，这样是避免在内核的栈里留下返回地址

SYCALL_4H endp
```

do_exit函数

```
void do_exit()
{
    pcbList[currentPCB->fid].state = 2;/*唤醒父进程*/
    currentPCB->state = 0;/*PCB块设置为空闲状态*/
    numOfProcess--;/*进程数量减少*/
    setIF();/*开中断，当时间中断来时，就会被切换走*/
    while (1);/*等待一个时间片结束，被切换到另一个就绪的进程*/
}
```

测试程序

这个程序主要是派生了一个子进程统计字符串中的数字的个数，父进程统计字母的个数。在命令行中输入8即可调用。具体代码如下

```
extern void scanf(char *s, ...);
extern void printf(char *s, ...);
extern int fork();
extern void wait();
extern void exit();

char str[80];/*字符串*/
int LetterNum = 0;/*记录字母个数*/
int DigitalNum = 0;/*记录数字个数*/

void countLetter()/*统计字母个数*/
{
    int i = 0;
    for (; str[i] != 0; i++)
    {
        if ((str[i] <= 'z' && str[i] >= 'a') || (str[i] <= 'Z' && str[i] >=
'A'))
        {
            LetterNum++;
        }
    }
}

void countDigital()/*统计数字的个数*/
{
    int i = 0;
    for (; str[i] != 0; i++)
    {
        if (str[i] <= '9' && str[i] >= '0')
        {
            DigitalNum++;
        }
    }
}

int main()
{
    int pid;
    printf("Input a string: ");
    scanf("%s", str);/*输入一个字符串*/
    pid = fork();/*派生进程*/
    if (pid == -1)/*派生失败*/
    {
```

```
        printf("error in fork!\r\n");
        exit();
    }
    if (pid)/*父进程*/
    {
        countLetter();
        wait();/*等待子进程*/
        printf("LetterNum=%d DigitalNum=%d\r\n", LetterNum, DigitalNum);
    }
    else/*子进程*/
    {
        countDigital();
        exit();
    }
}
```

实验过程

生成com文件

在DoxBox里使用命令

```
tasm kernel.asm
tasm kfun.asm
tcc -c ckfun.c
tlink /3 /t kernel.obj kfun.obj ckfun.obj,kernel.com
```

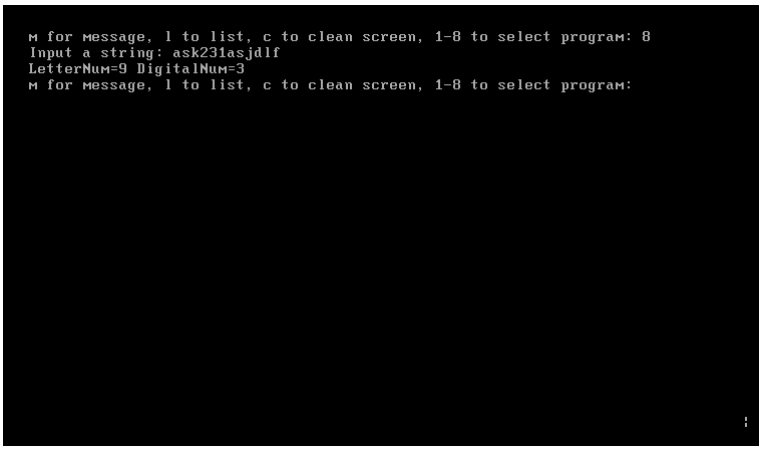
即可生成内核程序的com执行体。用户程序也类似。

写盘并且运行

写盘式，引导程序在首扇区，内核在2、3、4、5、6扇区，用户程序在7-17号扇区，可以用l指令查看用户程序所在的扇区

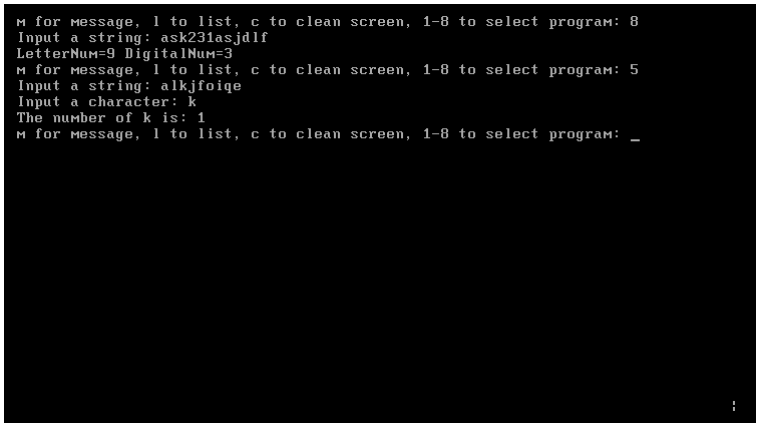
运行虚拟机

运行测试程序，输入8



结果正确

测试之前的程序，输入5



结果正常

实验总结

这次实验过程中，出了很多bug，主要是由于把栈段从程序中分离出来引起的。例如printf函数

```
void printf(char *s, ...)
{
    ……
}
```

如果这样使用

```
int main()
{
    char str[10];
    ……/*对字符串str进行初始化*/
    printf("%s\r\n",str);
}
```

会无法正常显示str里的内容，此时通过对查看printf的汇编代码就可以发现问题所在，问题在于printf访问字符串时，使用的是[bx]寻址，而[bx]默认的段寄存器是ds，但是str的内容却是在栈里，所以无法正确访问。归根到底，都是.com程序的”四段合一“结构不适合多线程并发。