**北京邮电大学计算机学院**

**2022-2023 学年第 2 学期项目总结报告**

**（每个项目小组一份）**

**课程名称：  *操作系统原理***

**项目名称： 实验四 进程调度**

**项目完成人：**

**姓名：\_万志恒 学号： 2021212413**

**姓名： 李伟泽 学号： 2021211462**

**姓名： 唐子潇 学号： 2021211460**

**指导教师： 赵方**

**日 期： 2023 年 5 月 27 日**

**一 . 项目目的和要求**

（1）. 理解 Linux 管理进程所用到的数据结构。

（2）. 理解 Linux 的进程调度算法的处理逻辑及其实现所用到的数据结构。

**二 .项目开发环境**

硬件：PC微型计算机、8核、16.0G内存、467G硬盘

软件：Vmware、Ubuntu操作系统、Vi、GCC、GDB

**三 .项目内容**

1. 通过查阅参考书或者上网找资料，熟悉/usr/src/linux（注意：这里最后一级目录名可能是个含有具体内核版本号和“linux”字符串的名字）下各子目录的内容，即所含 Linux源代码的情况。

2. 分析 Linux 进程调度有关函数的源代码，主要是 schedule()函数，并且要对它们引用的头文件等一并分析。

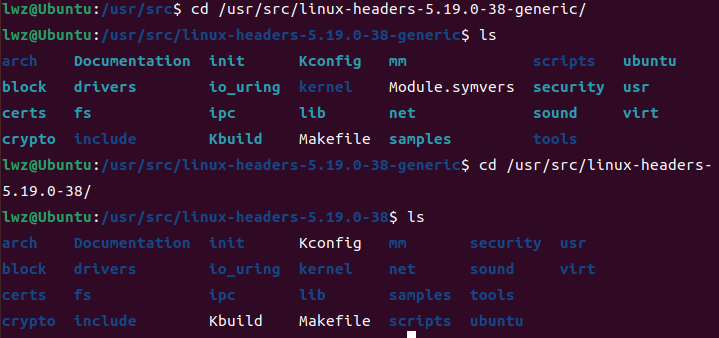
3. 实现 Linux 的进程调度算法及理解其实现所用的主要数据结构。

（可参考Linux0.11版本：https://github.com/Wangzhike/HIT-Linux-0.11

或者https://github.com/tinyclub）

**四 .项目结果及分析**

**任务一：**



arch：与所有体系结构相关结构的代码，arch目录下是好多个不同架构的CPU的子目录，如arm、X86

block：放一些linux存储体系中关于块设备管理的代码

certs：与认证和签名相关代码

crypto：目录下放了一些各种常见的加密算法的C语言代码实现

documentation：里面放了一些文档

drivers：驱动代码。这个目录是内核中最庞大的一个目录，显卡、网卡、SCSI适配器等其他任何LINUX支持的外设或总线的驱动程序都可以在这里找到。

fs：file system 虚拟文件系统的代码，和各个不同文件系统的代码都在这个目录中。

include：包含内核中大部分的头文件，各种CPU架构共用的头文件

init：内核的初始化代码。包括main.c、创建用户空间的代码及其他初始化代码。

io\_uring：高性能异步IO框架，适合io密集型应用

ipc：进程间通信实现，比如信号量、共享内存等等

Kbuild：kernel build system内核的编译

Kconfig：配置内核

kernel：内核中最核心的部分，包括进程的调度（sched.c）,以及进程的创建和撤销（fork.c和exit.c）和平台相关的另外一部分核心代码在arch/\*/kernel目录下。

lib：公用的库函数，与C语言的库函数不一样

Makefile：编译内核。将编译参数和编译所需的文件和必要的信息传给编译器

mm：memory management内存管理

net：核心的网络部分代码，实现了各种常见的网络协议

samples：一些内核编程的范例

scripts：该目录下没有内核代码，脚本，这个目录下全部是脚本文件，这些脚本文件不是linux内核工作时使用的，而是用来辅助对linux内核进行配置编译生产的

security：安全代码

sound：声卡驱动等

tools：包含和内核交互的工具

usr：实现用于打包和压缩的cpio等。这个文件夹中的代码在内核编译完成后创建这些文件

virt：包含了虚拟化代码，它允许用户依次运行多个操作系统。

**任务二：**

**Schedule()函数**

/\*

\* schedule() is the main scheduler function.

\* schedule()函数主要功能就是用另外一个进程来

\* 替换当前正在执行的进程

\*/

asmlinkage void \_\_sched schedule(void)

{

struct task\_struct \*prev, \*next;

unsigned long \*switch\_count;

struct rq \*rq;

int cpu;

need\_resched:/\* \* 禁用内核抢占\*/

preempt\_disable();

/\* \* 获取当前CPU核心ID\*/

cpu = smp\_processor\_id();

/\* \* 通过当前CPU核心ID获取正在运行队列数据结构\*/

rq = cpu\_rq(cpu);

/\* \* 标记不同的state度过quiescent state, \*/

rcu\_sched\_qs(cpu);

/\*\* 把当前进程赋给prev\*/

prev = rq->curr;

/\*\* 将截止目前的上下文切换次数赋给switch\_count\*/

switch\_count = &prev->nivcsw;

/\* \* 释放大内核锁，schedule（）必须要保证prev不能占中大内核锁\*/

release\_kernel\_lock(prev);

need\_resched\_nonpreemptible:

/\* \* 如果禁止内核抢占，而又调用了cond\_resched就会出错\* 这个函数就是用来捕获该错误的\*/

schedule\_debug(prev);

/\* \* 取消rq中hrtick\_timer\*/

if (sched\_feat(HRTICK))

hrtick\_clear(rq);

/\* \* 采用自旋锁，锁住rq，保护运行队列 \*/

spin\_lock\_irq(&rq->lock);

/\* \* 更新就绪队列的时钟\*/

update\_rq\_clock(rq);

/\*\* 清除prev需要调度标志TIF\_NEED\_RESCHED， \* 避免进入就绪队列中 \*/

clear\_tsk\_need\_resched(prev);

/\* \* 检查prev状态，如果不是可运行状态，

\* 且prev没有在内核态被抢占。

\*/

if (prev->state && !(preempt\_count() & PREEMPT\_ACTIVE)) {

/\*\* 检查prev是否是阻塞挂起，且状态为TASK\_INTERRUPTIBLE

\* 就把prev状态设置为TASK\_RUNNING。\*/

if (unlikely(signal\_pending\_state(prev->state, prev)))

/\*

\* 这里设置为TASK\_RUNNING，因为prev进程中有信号需要处理，

\* 不能从运行对列中删除，否则信号处理不了，会影响其余进程。

\*/

prev->state = TASK\_RUNNING;

else

/\*

\* 把prev从运行队列中删除

\*/

deactivate\_task(rq, prev, 1);

switch\_count = &prev->nvcsw;

}

/\*

\* 通知调度器，即将发生进程切换

\*/

pre\_schedule(rq, prev);

/\* \* 如果运行队列中没有可运行队列，

\* 则从另一个运行队列迁移可运行进程到本地队列中来。

\*/

if (unlikely(!rq->nr\_running))

idle\_balance(cpu, rq);

/\* \* 通知调度器，即将用另一个进程替换当前进程\*/

put\_prev\_task(rq, prev);

/\*\* 选择下一个进程\*/

next = pick\_next\_task(rq);

/\*\* 判断选择出的下一个进程是否是当前进程\*/

if (likely(prev != next)) {

/\* \* 计算prev和next进程运行时间等参数 \*/

sched\_info\_switch(prev, next);

/ \* 从调度程序调用以删除当前任务的事件，同时禁用中断

\* 停止每个事件并更新事件->计数中的事件值。\*/

perf\_event\_task\_sched\_out(prev, next, cpu);

/\* \* 队列切换次数更新 \*/

rq->nr\_switches++;

/\*\* 将next标记为队列的curr进程\*/

rq->curr = next;

/\* \* 进程切换次数更新 \*/

++\*switch\_count;

/\* \* 进程之间上下文切换，两个进程切换就在此处发生

\* 两个进程切换两大部分：1.prev到next虚拟地址空间的映射，

\* 由于内核虚拟地址空间是不许呀切换的,

\* 因此切换的主要是用户态的虚拟地址空间。

\* 2.保存、恢复栈信息和寄存器信息。

\*/

context\_switch(rq, prev, next); /\* unlocks the rq \*/

/\*

\* the context switch might have flipped the stack from under

\* us, hence refresh the local variables.进程切换了，刷新局部变量。

\*/

cpu = smp\_processor\_id();

rq = cpu\_rq(cpu);

} else

/\*

\* 释放rq锁

\*/

spin\_unlock\_irq(&rq->lock);

/\*

\* 通知调度器，完成了进程切换

\*/

post\_schedule(rq);

/\*

\* 重新获取大内核锁，如果获取不到则需要重新调度

\*/

if (unlikely(reacquire\_kernel\_lock(current) < 0))

goto need\_resched\_nonpreemptible;

/\*

\* 重新使能内核抢占

\*/

preempt\_enable\_no\_resched();

/\*

\* 检查其余进程已经设置当前进程的TIF\_NEED\_RESCHED标志，

\* 如果设置了需要进行重新调度。

\*/

if (need\_resched())

goto need\_resched;

}

**sleep\_on()函数**

void sleep\_on(struct task\_struct \*\*p)

{

struct task\_struct \*tmp;//临时指针

if (!p) //若指针无效，则退出

return;

if (current == &(init\_task.task)) //若当前任务是0号进程则死机

panic("task[0] trying to sleep");

tmp = \*p; //让tmp指向已经在等待队列上的任务,\*p表示等待队列头指针

\*p = current; //将睡眠队列头的等待指针指向当前任务

current->state = TASK\_UNINTERRUPTIBLE; //将当前任务置为不可中断的等待状态

schedule(); //重新调度

// 只有当这个等待任务被唤醒时，调度程序才又返回到这里，则表示进程已被明确地唤醒。

if (tmp)

tmp->state=0; // 若在其前还存在等待的任务，则也将其置为就绪状态（唤醒）。

}

在几个进程为等待同一资源而多次调用该函数时，程序就隐式地构筑出一个等待队列。

在插入等待队列后，sleep\_on()函数就会调用schedule()函数去执行别的进程。当进程被唤醒而重新

执行时就会执行后续的语句，把比它早进入等待队列的一个进程唤醒。

**wake\_up()函数**

void wake\_up(struct task\_struct \*\*p)

{

if (p && \*p) {

(\*\*p).state=0; // 置为TASK\_RUNNING就绪（可运行）状态。

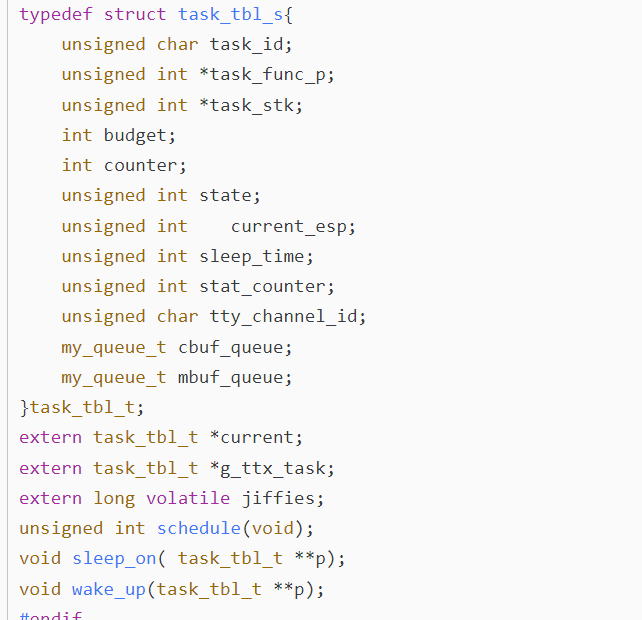
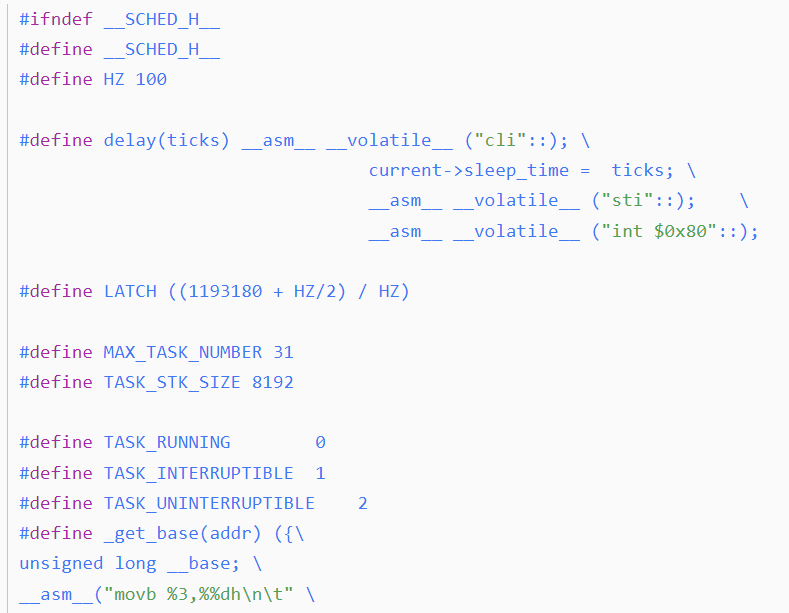
\*p=NULL;

}

}

**sched.h头文件**

sched.h主要定义了时钟频率、任务数据结构、和sleep宏



**任务三：**

**FIFO调度算法**

**源程序：**

#include<stdio.h>

#include<stdlib.h>

#include<sys/types.h>

#include<unistd.h>

#include<pthread.h>

#include<time.h>

#include<iostream>

#define Thread\_Num 20

**using** **namespace** std**;**

pthread\_mutex\_t Device\_mutex**;**

**struct** VirtualPCB

**{**

**int** tid**;**

**int** priority**;**

**int** waittime**;**

**int** runtime**;**

**int** arrivetime**;**

**int** visited**;**

**int** tempruntime**;**

**public:**

**int** gettid**()**

**{**

**return** tid**;**

**}**

**int** getwaittime**()**

**{**

**return** waittime**;**

**}**

**int** getpriority**()**

**{**

**return** priority**;**

**}**

**int** getruntime**()**

**{**

**return** runtime**;**

**}**

**int** getarrivetime**()**

**{**

**return** arrivetime**;**

**}**

**void** setvisit**(int** a**)**

**{**

visited**=**a**;**

**}**

**int** getvisit**()**

**{**

**return** visited**;**

**}**

**int** gettempruntime**()**

**{**

**return** tempruntime**;**

**}**

**void** setwaittime**(int** n**)**

**{**

waittime **=** n**;**

**}**

**void** settempruntime**(int** n**)**

**{**

tempruntime **=** tempruntime **-** n**;**

**}**

**}**TCB**[**Thread\_Num**];**

*//PCB初始化函数*

**void** t\_init**()**

**{**

**int** n**;**

srand**(**time**(**NULL**));**

**for(**n **=**0**;**n**<**Thread\_Num**;**n**++)**

**{**

TCB**[**n**].**tid **=** n **+** 1**;***//用线程创建序号作为虚拟id*

*//用随机数随机产生虚拟PCB的值*

TCB**[**n**].**priority **=** 1 **+** rand**()%**19**;**

TCB**[**n**].**runtime **=** 1 **+** rand**()%**19**;**

TCB**[**n**].**arrivetime **=** rand**()%**201**;***//假定所有线程在0~500时间内相继就绪*

TCB**[**n**].**waittime **=** 0**;**

TCB**[**n**].**visited **=**0**;**

TCB**[**n**].**tempruntime **=** TCB**[**n**].**runtime**;**

**}**

**}**

*//线程运行内容*

**void** **\***t\_print**(void** **\***arg**)**

**{**

**int** n **=** **\*(int** **\*)**arg**;***//获取当前线程id*

**while(**1**)**

**{**

pthread\_mutex\_lock**(&**Device\_mutex**);**

printf**("Thread\_%-2d: ",** n**);**

printf**("tid:%-2d priority:%-2d arrivetime:%-2d runtime:%-2d \n",**TCB**[**n**-**1**].**gettid**(),** TCB**[**n**-**1**].**priority**,** TCB**[**n**-**1**].**arrivetime**,** TCB**[**n**-**1**].**runtime**);**

pthread\_mutex\_unlock**(&**Device\_mutex**);**

sleep**(**1**);**

**break;**

**}**

pthread\_exit**(**0**);**

**}**

*//FIFO*

**void** FIFO**()**

**{**

cout**<<"-----------FIFO:"<<**endl**;**

**int** i**,**j**;**

**int** start **=** 0**;**

**float** waittime **=** 0**;**

**float** avwait **=** 0**;**

**for(**i**=**0**;**i**<**201**;**i**++)**

**{**

**for(**j**=**0**;**j**<**Thread\_Num**;**j**++){**

**if(**TCB**[**j**].**getarrivetime**()==**i **&&** TCB**[**j**].**getvisit**()==**0**){**

**if(**start**<**TCB**[**j**].**getarrivetime**())** start **=** TCB**[**j**].**getarrivetime**();***//空转情况*

printf**("Thread: %-2d Arrive: %-3d Start: %-3d Runtime: %-2d\n",**

TCB**[**j**].**gettid**(),**TCB**[**j**].**getarrivetime**(),**start**,**TCB**[**j**].**getruntime**());**

TCB**[**j**].**waittime**=(float)**start **-** TCB**[**j**].**getarrivetime**();**

waittime**+=**TCB**[**j**].**getwaittime**();**

start **=** start **+** TCB**[**j**].**getruntime**();**

TCB**[**j**].**setvisit**(**1**);**

**}**

**}**

**}**

avwait **=** waittime **/** **(float)**Thread\_Num**;**

printf**("Total waitting time : %f\n",**waittime**);**

printf**("Average waitting time : %f\n",**avwait**);**

**}**

*//创建线程函数*

**void** **\***Children**(void\*)**

**{**

**int** ret**[**Thread\_Num**];**

t\_init**();**

pthread\_t tid**[**Thread\_Num**];**

pthread\_mutex\_init**(&**Device\_mutex**,**NULL**);**

**int** i**,**j**;**

**for(**i**=**0**;**i**<**Thread\_Num**;**i**++)**

**{**

**int** k **=**i**+**1**;**

ret**[**i**]** **=** pthread\_create**(&**tid**[**i**],**NULL**,&**t\_print**,** **&**k**);**

**if(**ret**[**i**]** **==** 0**)** **{**

sleep**(**1**);**

**}**

**else{**

printf**("Thread\_%-2d failed!\n",**i**+**1**);**

**}**

**}**

**for(**j**=**0**;**j**<**Thread\_Num**;**j**++)**

pthread\_join **(**tid**[**i**],** NULL**);**

pthread\_mutex\_destroy**(&**Device\_mutex**);**

pthread\_exit**(**0**);**

**}**

**int** main**()**

**{**

**int** ret1**;**

pthread\_t tid1**;***//主线程*

ret1 **=** pthread\_create**(&**tid1**,**NULL**,&**Children**,**NULL**);**

**if(**ret1 **==** 0**)**

**{**

printf**("Main Thread done\n");**

sleep**(**20**);**

**}**

**else{**

printf**("Thread failed!\n");**

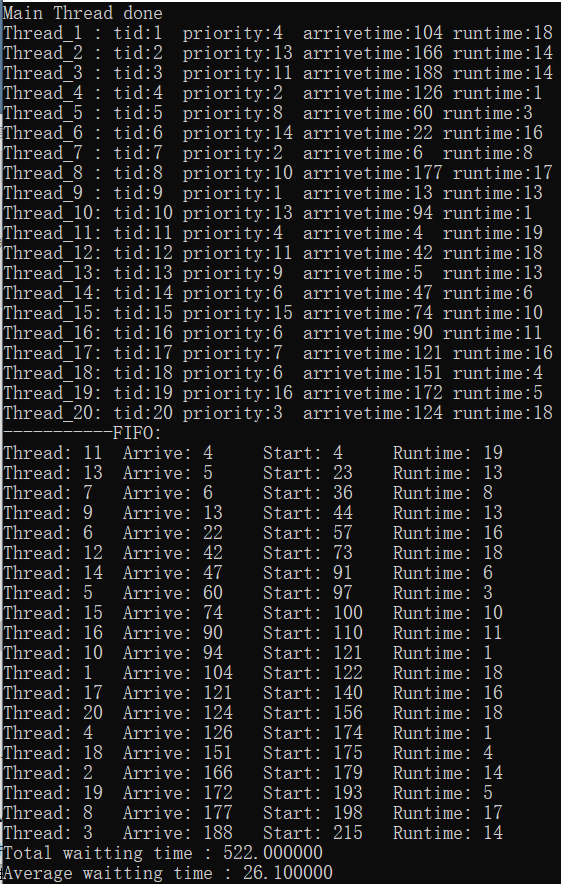
**}**

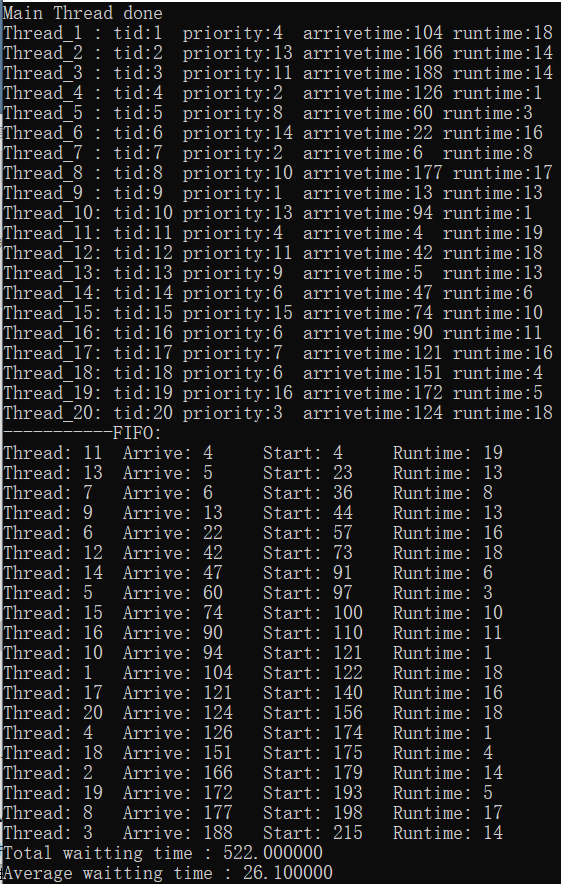
FIFO**();**

**return** 0**;**

**}**

**运行结果：**

****

****

**Priority调度算法**

**源程序：**

#include<stdio.h>

#include<stdlib.h>

#include<sys/types.h>

#include<unistd.h>

#include<pthread.h>

#include<time.h>

#include<iostream>

#define Thread\_Num 20

**using** **namespace** std**;**

pthread\_mutex\_t Device\_mutex**;**

**struct** VirtualPCB

**{**

**int** tid**;**

**int** priority**;**

**int** waittime**;**

**int** runtime**;**

**int** arrivetime**;**

**int** visited**;**

**int** tempruntime**;**

**public:**

**int** gettid**()**

**{**

**return** tid**;**

**}**

**int** getwaittime**()**

**{**

**return** waittime**;**

**}**

**int** getpriority**()**

**{**

**return** priority**;**

**}**

**int** getruntime**()**

**{**

**return** runtime**;**

**}**

**int** getarrivetime**()**

**{**

**return** arrivetime**;**

**}**

**void** setvisit**(int** a**)**

**{**

visited**=**a**;**

**}**

**int** getvisit**()**

**{**

**return** visited**;**

**}**

**int** gettempruntime**()**

**{**

**return** tempruntime**;**

**}**

**void** setwaittime**(int** n**)**

**{**

waittime **=** n**;**

**}**

**void** settempruntime**(int** n**)**

**{**

tempruntime **=** tempruntime **-** n**;**

**}**

**}**TCB**[**Thread\_Num**];**

*//PCB初始化函数*

**void** t\_init**()**

**{**

**int** n**;**

srand**(**time**(**NULL**));**

**for(**n **=**0**;**n**<**Thread\_Num**;**n**++)**

**{**

TCB**[**n**].**tid **=** n **+** 1**;***//用线程创建序号作为虚拟id*

*//用随机数随机产生虚拟PCB的值*

TCB**[**n**].**priority **=** 1 **+** rand**()%**19**;**

TCB**[**n**].**runtime **=** 1 **+** rand**()%**19**;**

TCB**[**n**].**arrivetime **=** 0**;***//假定所有线程在0时刻依次就绪*

TCB**[**n**].**waittime **=** 0**;**

TCB**[**n**].**visited **=**0**;**

TCB**[**n**].**tempruntime **=** TCB**[**n**].**runtime**;**

**}**

**}**

*//线程运行内容*

**void** **\***t\_print**(void** **\***arg**)**

**{**

**int** n **=** **\*(int** **\*)**arg**;***//获取当前线程id*

**while(**1**)**

**{**

pthread\_mutex\_lock**(&**Device\_mutex**);**

printf**("Thread\_%-2d: ",** n**);**

printf**("tid:%-2d priority:%-2d runtime:%-2d \n",**TCB**[**n**-**1**].**gettid**(),** TCB**[**n**-**1**].**priority**,** TCB**[**n**-**1**].**arrivetime**,** TCB**[**n**-**1**].**runtime**);**

pthread\_mutex\_unlock**(&**Device\_mutex**);**

sleep**(**1**);**

**break;**

**}**

pthread\_exit**(**0**);**

**}**

*//优先级调度*

**void** Priority**()**

**{**

cout**<<"-----------Priority:"<<**endl**;**

**int** i**,**j**;**

**int** start **=** 0**;**

**float** waittime **=** 0**;**

**float** avwait **=** 0**;**

**for(**i**=**1**;**i**<**Thread\_Num**;**i**++)**

**{**

**for(**j**=**0**;**j**<**Thread\_Num**;**j**++){**

**if(**TCB**[**j**].**getpriority**()==**i **&&** TCB**[**j**].**getvisit**()==**0**){**

printf**("Thread: %-2d Priority: %-2d Start: %-3d Runtime: %-2d\n",**

TCB**[**j**].**gettid**(),**TCB**[**j**].**priority**,** start**,** TCB**[**j**].**getruntime**());**

waittime **=** waittime **+** **(float)**start**;**

start **=** start **+** TCB**[**j**].**getruntime**();**

TCB**[**j**].**setvisit**(**1**);**

**}**

**}**

**}**

avwait **=** waittime **/** **(float)**Thread\_Num**;**

printf**("Total waitting time : %f\n",**waittime**);**

printf**("Average waitting time : %f\n",**avwait**);**

**}**

*//创建线程函数*

**void** **\***Children**(void\*)**

**{**

**int** ret**[**Thread\_Num**];**

t\_init**();**

pthread\_t tid**[**Thread\_Num**];**

pthread\_mutex\_init**(&**Device\_mutex**,**NULL**);**

**int** i**,**j**;**

**for(**i**=**0**;**i**<**Thread\_Num**;**i**++)**

**{**

**int** k **=**i**+**1**;**

ret**[**i**]** **=** pthread\_create**(&**tid**[**i**],**NULL**,&**t\_print**,** **&**k**);**

**if(**ret**[**i**]** **==** 0**)** **{**

sleep**(**1**);**

**}**

**else{**

printf**("Thread\_%-2d failed!\n",**i**+**1**);**

**}**

**}**

**for(**j**=**0**;**j**<**Thread\_Num**;**j**++)**

pthread\_join **(**tid**[**i**],** NULL**);**

pthread\_mutex\_destroy**(&**Device\_mutex**);**

pthread\_exit**(**0**);**

**}**

**int** main**()**

**{**

**int** ret1**;**

pthread\_t tid1**;***//主线程*

ret1 **=** pthread\_create**(&**tid1**,**NULL**,&**Children**,**NULL**);**

**if(**ret1 **==** 0**)**

**{**

printf**("Main Thread done\n");**

sleep**(**20**);**

**}**

**else{**

printf**("Thread failed!\n");**

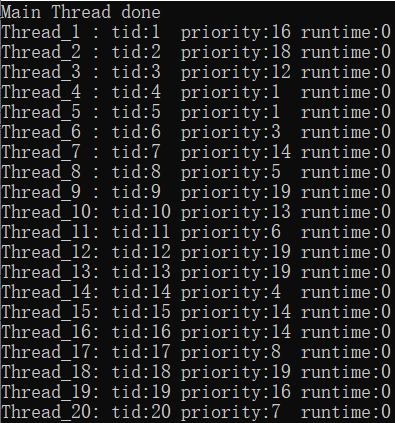
**}**

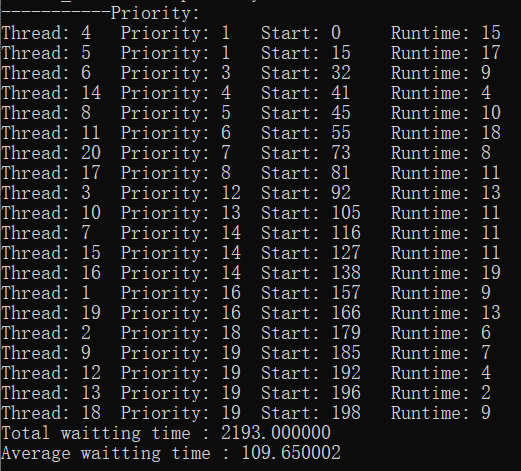
Priority**();**

**return** 0**;**

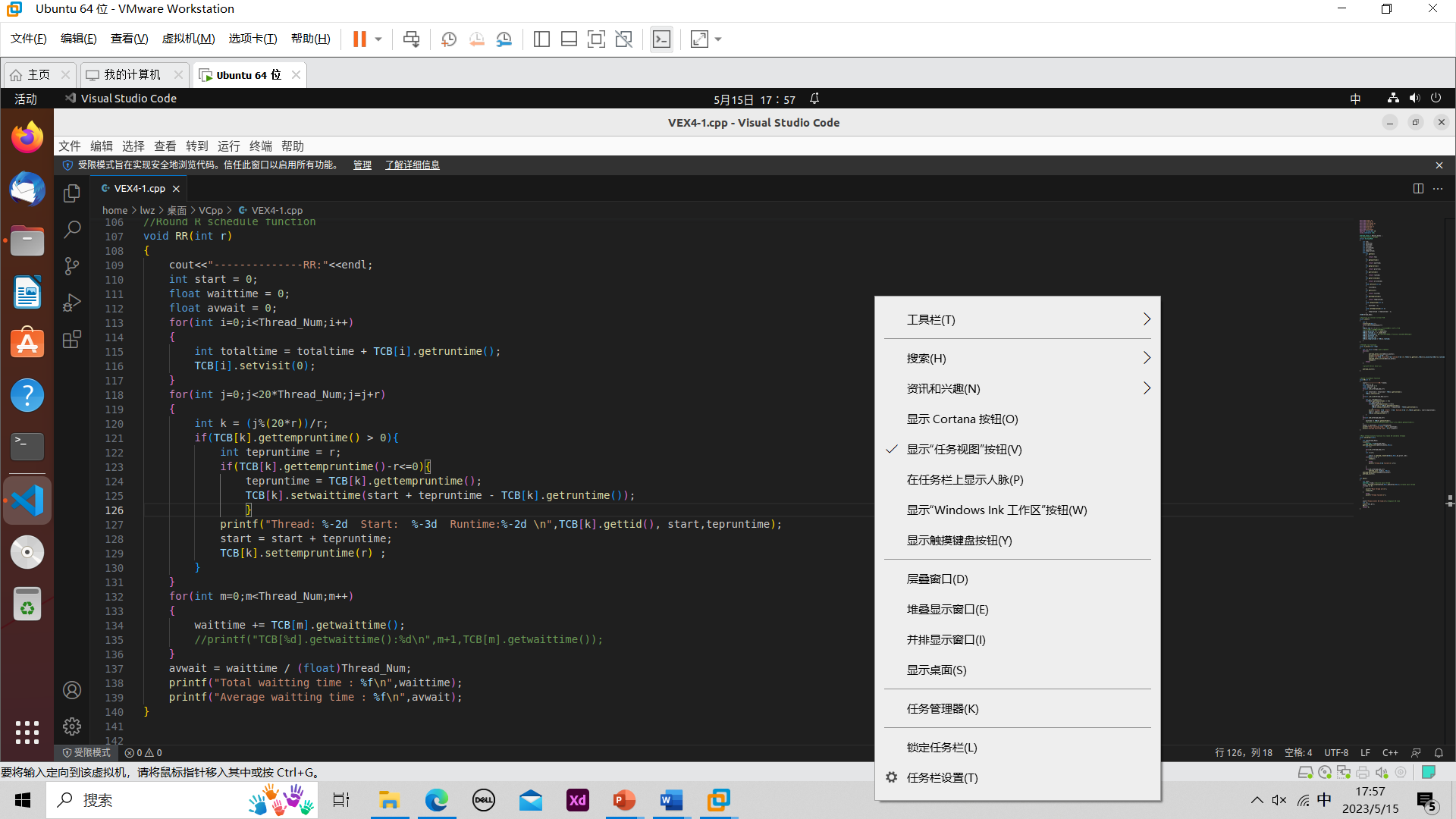
**}**

**实验结果：**

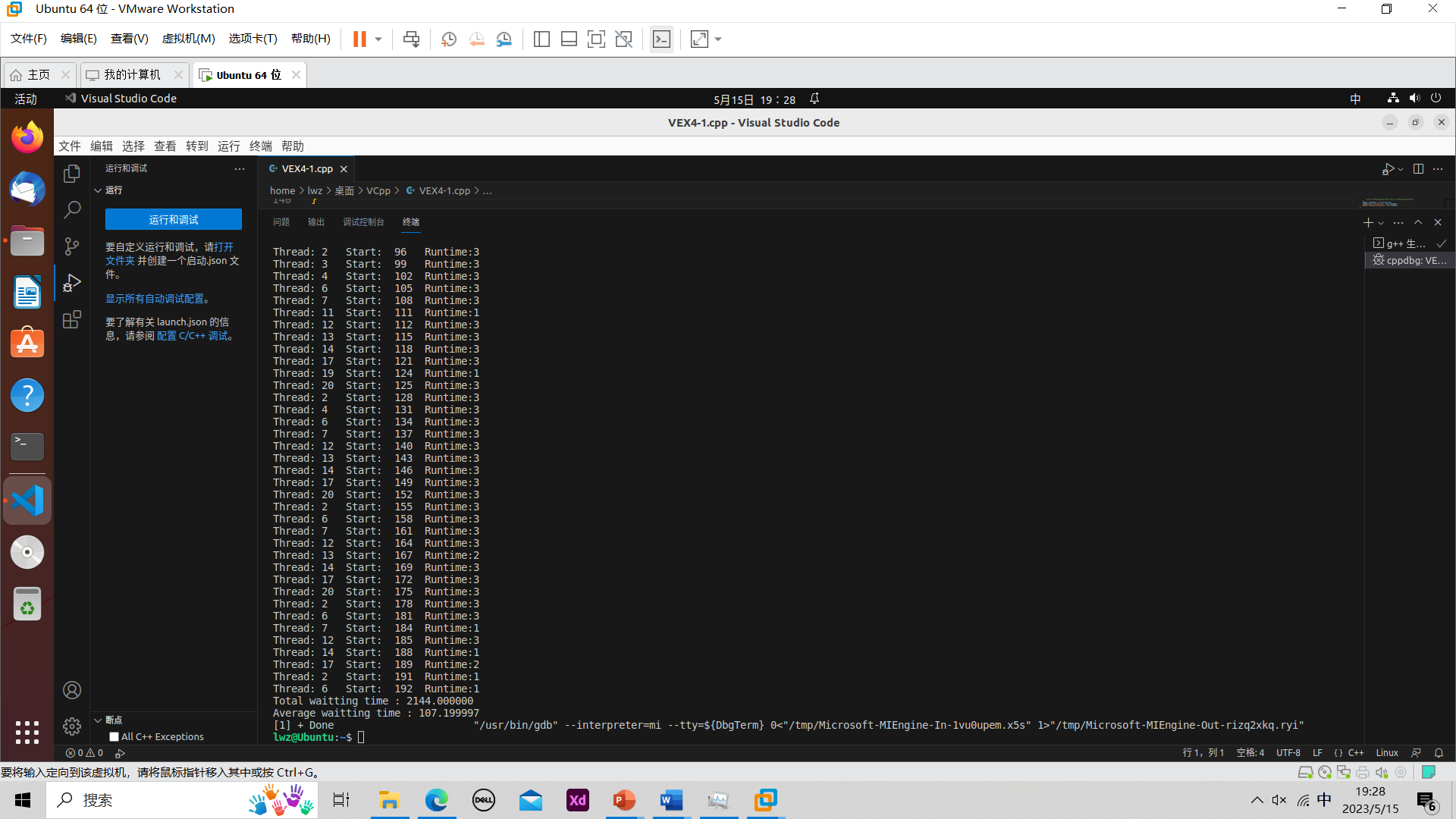
****

****

**RR调度算法**



**运行结果（时间片=3）：**



**五 .项目人员、进度安排及完成过程**

项目人员：万志恒、李伟泽、唐子潇

李伟泽负责任务一、任务二以及任务三的RR算法，万志恒负责任务三的FIFO算法和Priority算法，唐子潇负责任务二及任务三的源码分析

**六 . 项目心得及体会**

万志恒：学习了Linux操作系统中的三种调度，通过阅读源码，更加深刻地认识了这三种调度地实现方式。手动实现FIFO算法和Priority算法，锻炼了我的代码能力，同时加深了我对FIFO算法和Priority算法的印象以及对他们各自特点的认识。

李伟泽：通过查找网络资料，熟悉了/usr/src/linux下各子目录的内容；分析了schedule()、sleep\_on()和wake\_up()函数源码，对线程的调度有了更深层次的认知；手动实现RR算法，锻炼了代码编写能力，并加固了对RR算法原理的印象。

唐子潇：通过本次实验，我熟悉了Linux进程调度有关函数的源代码，初步了解了如何通过编程实现 Linux 的进程调度算法，尤其是其中的CFS公平调度算法。

**七 . 附录（附上项目其他文档，具体内容根据项目要求确定**