**北京科技大学实验报告**

学院： 计算机与通信工程学院 专业： 计算机科学与技术 班级： 计184

姓名： 王丹琳 学号： 41824179 实验日期： 2020 年 11月 16日

**实验名称：操作系统实验7 扩展实验（线程调度算法改进）（2分）**

**实验目的：**实现多级反馈队列调度算法

**实验环境：**EOS操作系统及其实验环境。

**实验内容：**EOS 操作系统目前已经实现了基于优先级的抢先式调度，并且已经在实验 6 中实现了时间片的轮转调度算法，可以让同一优先级中的就绪线程轮转执行。但是，如果考虑到有一个运行时间较长的批处理作业优先级比较高，这样较低优先级的短作业就会在一段较长的时间内无法运行，甚至无法及时响应用户。使用多级反馈队列调度算法可以有效解决此问题。实现线程调度算法改进。

**实验步骤：**

1. 实现时间片轮转调度算法。

主要实现思路：

当一个新线程进入就绪态后，首先将它放入对应优先级队列的末尾，按 FCFS原则排队等待调度。当轮到该线程执行时，如能在已分配的时间片内完成，便可准备结束此线程；如果在分配的时间片用完时尚未完成，需要先降低该线程的优先级并增大该线程的时间片，然后调度程序将该线程转入下一个队列的末尾，再同样地按 FCFS 原则等待调度执行；如果它在该队列中运行时用完已分配的时间片后仍未完成，再依法将它转入下一队列。如此下去，当一个长作业从第一队列降到最后一个队列后，就采取按时间片轮转的方式运行，无法再继续降低优先级了。（注意：在优先级愈高的队列中，每个线程的初始时间片就规定得愈小）并且仅当第一队列空闲时，调度程序才调度第二队列中的线程运行；仅当第 1~（N- 1）队列均为空时，才会调度第 N 队列中的线程运行。

VOID

PspRoundRobin(

VOID

)

{ //

// 在此添加代码，实现时间片轮转调度算法。

//

int i,j;

PLIST\_ENTRY pReadyNode;

PTHREAD Thread;

//判断被中断线程是否仍处于运行状态

if (NULL != PspCurrentThread && Running == PspCurrentThread->State){

PspCurrentThread->RemainderTicks--;//剩余时间片--

PspCurrentThread->UseTicks++;//所用时间片++

if (PspCurrentThread->RemainderTicks == 0){//所分配的时间片用完了

if (PspCurrentThread->Priority != 0) {

//降低该线程的优先级

//并增大该线程的时间片，然后调度程序将该线程转入下一个队列的末尾，再同样地

//按 FCFS 原则等待调度执行；

PspCurrentThread->Priority--;

PspCurrentThread->BeginTicks = (8 - PspCurrentThread->Priority)\*10 + TICKS\_OF\_TIME\_SLICE;//// 线程初始时间片数量

PspCurrentThread->RemainderTicks = (25 - PspCurrentThread->Priority)\*10 + TICKS\_OF\_TIME\_SLICE;

PspReadyThread(PspCurrentThread);

} else{

//降到最后一个队列后，就采取按时间片轮转的方式运行（无法再继续降低优先级了）

PspCurrentThread->RemainderTicks = (31 - PspCurrentThread->Priority)\*10 + TICKS\_OF\_TIME\_SLICE;

//// 如果存在同优先级的就绪线程

if (BIT\_TEST(PspReadyBitmap, PspCurrentThread->Priority)){

PspReadyThread(PspCurrentThread);// 线程插入新优先级对应的就绪队列的队尾。

}

}

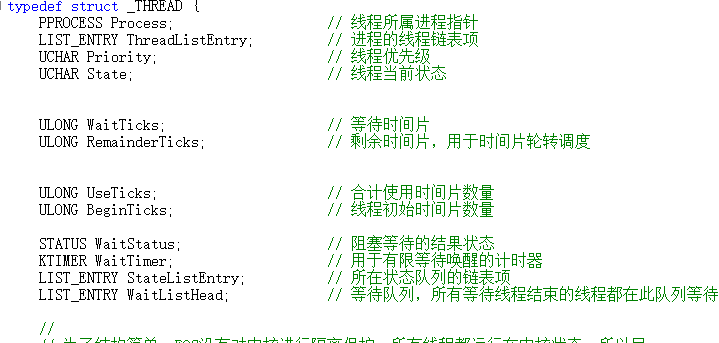
}

}

return;

}

1. 修改时间片的大小 TICKS\_OF\_TIME\_SLICE 为 100，方便观察执行后的效果。
2. 在控制台命令“rr”的处理函数中，将 Sleep 时间更改为 200\*1000，这样可以有充足的时间查看优先级降低后的效果。
3. 修改线程控制块（TCB）结构体，在其中新增两个成员，一个是线程整个生命周期中合计使用的时间片数量，另一个是线程的初始时间片数量。



1. 修改“rr”命令在控制台输出的内容和格式，不再显示线程计数，而是显示线程初始化时间片的大小，已使用时间片的合计数量，剩余时间片的数量。注意，在调用fprintf 函数格式化字符时，需要在字符串的末尾增加一个空格，否则会导致输出异常。



1. 在实现多级反馈队列调度算法后（注意：数字越大，优先级越高，反之，数字越小，优先级越低），使用实验 6 中提供的“rr”命令，查看各个线程的优先级逐步降低的过程。
2. 由于 EOS 没有提供鼠标，可以使用键盘事件或者控制台命令使线程优先级提升。由于键盘事件与线程之间没有建立一个明确的会话关系，所以还需要解决使用键盘事件提升哪个线程优先级的问题。一个简单的方式是，在键盘的中断处理程序中（在io/driver/keyboard.c 文件的 396 行的 KbdIsr 函数），如果当前线程（注意不能是 2 号线程）处于运行状态并且优先级大于 0 小于 8 的话（由于空闲线程的优先级为 0，不能更改该线程的优先级，如果当前线程的优先级为 8，没有必要再做提升线程优先级的操作），按下空格键，响应键盘事件后，就将其优先级提升为默认的优先级即可。关于键盘中断相关的内容可以参考实验 12。
3. 使用控制台命令提升线程优先级，在 EOS 操作系统中实现一个“up ThreadID”命令,通过输入的线程 ID 来提升对应线程的优先级。在实现命令的过程中需要做如下判断：需要提升线程的优先级应该大于 0 并且小于 8，如果是处于就绪状态的线程，需要先将该线程移出队列，然后设置该线程的优先级为默认值 8，并设置线程的初始时间片大小和剩余时间片大小，如果是处于运行状态或阻塞状态的线程，直接设置线程的优先级即可。测试提升线程优先级命令的方法：在控制台窗口 1 执行 rr时，可以按 Ctrl + F2 切换到控制台窗口 2，然后输入“up 24”命令，按回车执行该命令，按 Ctrl + F1 切换到控制台窗口 1，可以查看 ID 为 24 的线程优先级已更新为 8，并且初始时间片的大小恢复为基础时间片的大小。由于在使用控制台命令提升线程优先级时按下了空格键，此时查看已提升优先级的线程会有两个。

PRIVATE

VOID

ConsoleCmdUp(

IN HANDLE StdHandle,

IN PCSTR Arg

)

{

ULONG ThreadID;

HANDLE hThread;

STATUS Status;

BOOL IntState;

PTHREAD Thread;

//

// 从命令参数字符串中获得线程 ID。

//

ThreadID = atoi(Arg);

if(0 == ThreadID) {

fprintf(StdHandle, "Please input a valid thread ID.\n");

return;

}

//

// 由线程 ID 获得线程句柄

//

hThread = (HANDLE)OpenThread(ThreadID);

if (NULL == hThread) {

fprintf(StdHandle, "%d is an invalid thread ID.\n", ThreadID);

return;

}

Status = ObRefObjectByHandle(hThread, PspThreadType, (PVOID\*)&Thread);

if (!EOS\_SUCCESS(Status)) {

return ;

}

IntState = KeEnableInterrupts(FALSE);

if (Thread->Priority<=0 || Thread->Priority>=8) {

fprintf(StdHandle, "The Priority of the Thread should between 0 and 8.\n");

return;

}

//

// 升级线程

//

if (Ready == Thread->State) {

//

// 线程脱离当前优先级对应的就绪队列。

//

PspUnreadyThread(Thread);

Thread->Priority = 8;

Thread->RemainderTicks = (31 - Thread->Priority)\*10 + TICKS\_OF\_TIME\_SLICE;

Thread->BeginTicks = (31 - Thread->Priority)\*10 + TICKS\_OF\_TIME\_SLICE;

//

// 线程插入新优先级对应的就绪队列的队尾。

//

PspReadyThread(Thread);

} else {

Thread->Priority = 8;

}

//

// 优先级改变后需要执行线程调度。

//

PspThreadSchedule();

//

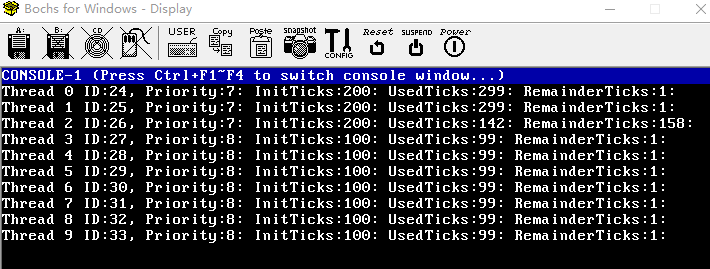
// 关闭线程句柄

//

CloseHandle(hThread);

}

**结果分析：**



从上图可以发现，对于同一个队列中的各个进程，按照FCFS分配时间片调度。对于不同队列的进程，优先级高的队列先进行调度，且随着优先级的降低，所分配的时间片变大。