第 0004 讲 2 进程优先级与调度策略-实战分析

一、基础知识

1、Linux 内核当中有 3 种调度策略：

SCHED_OTHER 分时调度策略；

SCHED_FIFO 实时调度策略，先到先服务；

SCHED_RR 实时调度策略，时间片轮转。

备注：如果有相同优先级的实时进程（根据优先级计算的调度权值是一样的）已经准备好，FIFO 时必须等待该进程主动放弃之后才可以运行这个优先级相同的任务。而 RR 可以每个任务都执行一段时间。

2、获取线程设置的最高和最低优先级函数如下：

int sched_get_priority_max(int policy); // 获取实时优先级的最大值

int sched_get_priority_min(int policy); // 获取实时优先级的最小值

SCHED_OTHER 它不支持优先级使用，而 SCHED_RR/SCHED_FIFO 支持优先级使用，它们分析为 1-99，数值越大优先级越高。

实时调度策略（SCHED_FIFO/SCHED_RR）优先级最大值为99；

普通调度策略（SCHED_NORMAL/SCHED_BATCH/SCHED_IDLE），始终返回0，即普通任务调度的函数。

3、设置和获取优先级 2 个主要核心函数：

```
int pthread_attr_setschedparamg(pthread_attr_t *attr,const struct sched_param *param); // 创建线程优先级
int pthread_attr_getschedparam(pthread_attr_t *attr,const struct sched_param *param); // 获取线程优先级
```

```
struct sched_param
{
int sched_priority; // 所有设定的线程优先级
};
param.sched_priority=11; // 设置优先级
```

当操作系统创建线程时，默认线程是 SCHED_OTHER，我们也可以通过改变调度策略，使用如下函数：

```
int pthread_attr_setschedpolicy(pthread_attr_t *attr,int policy); // 设置线程调度策略
```

二、基础案例分析

1、操作系统所支持优先级测试程序分析，具体源码如下：

```
#include <stdio.h>
#include <pthread.h>
#include <sched.h>
#include <assert.h>
static int GetThreadPolicyFunc(pthread_attr_t *pAttr)
{
int iPlicy;
int igp=pthread_attr_getschedpolicy(pAttr,&iPlicy);
assert(igp==0);

switch (iPlicy)
{
case SCHED_FIFO:
   printf("Policy is --> SCHED_FIFO.\n");
   break;

case SCHED_RR:
   printf("Policy is --> SCHED_RR.\n");
   break;
```

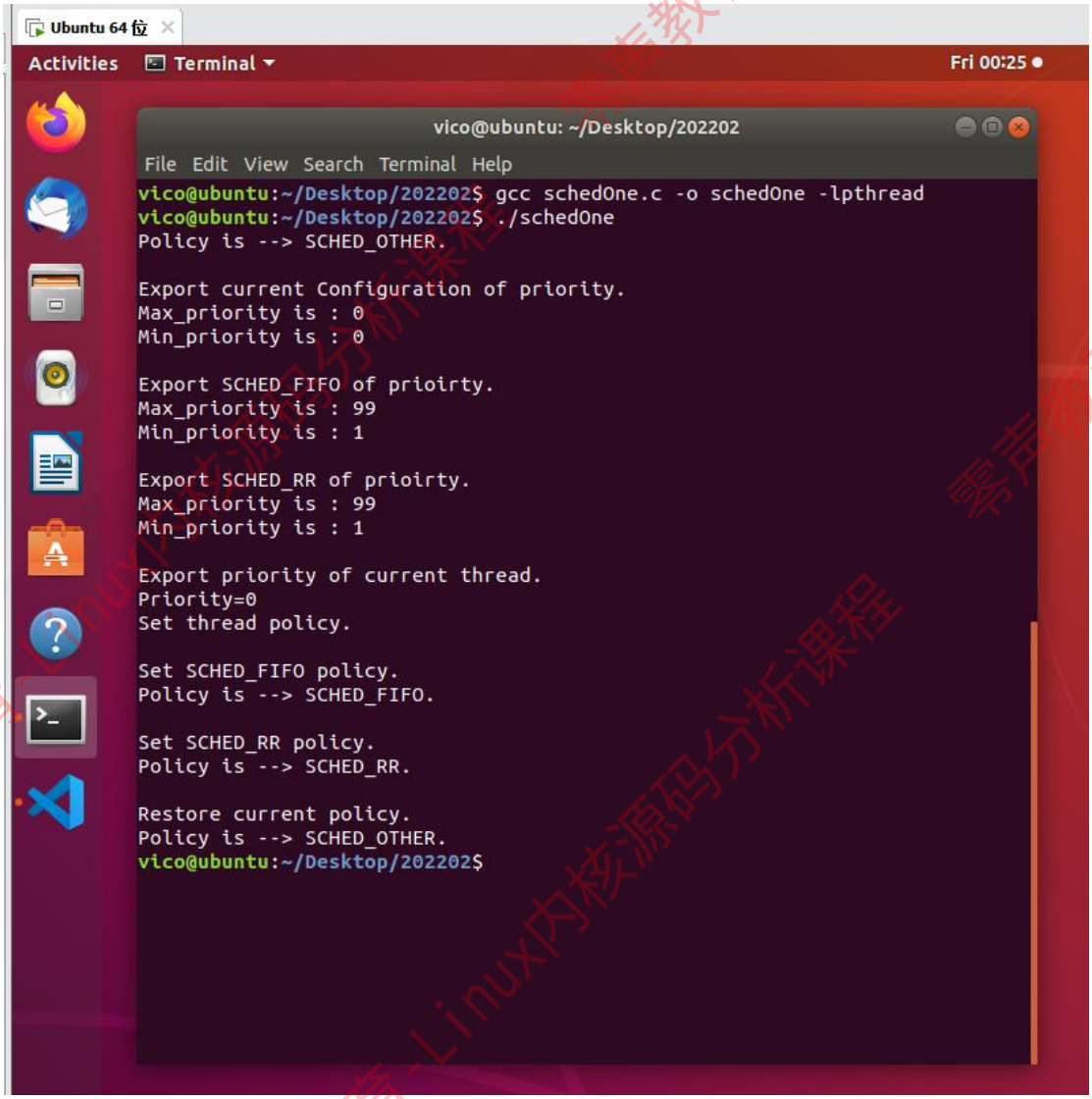
```
case SCHED_OTHER:
    printf("Policy is --> SCHED_OTHER.\n");
    break;

default:
printf("Policy is --> Unknown.\n");
    break;
}

return iPlicy;
}
static void PrintThreadPriorityFunc(pthread_attr_t *pAttr,int
iPolicy)
{
int iPriority=sched_get_priority_max(iPolicy);
assert(iPriority!=-1);
printf("Max_priority is : %d\n",iPriority);
iPriority=sched_get_priority_min(iPolicy);
assert(iPriority!=-1);
printf("Min_priority is : %d\n",iPriority);
}
```

```
static int GetThreadPriorityFunc(pthread_attr_t *pAttr)
{
struct sched_param sParam;
int irs=pthread_attr_getschedparam(pAttr,&sParam);
assert(irs==0);

printf("Priority=%d\n",sParam.__sched_priority);

return sParam.__sched_priority;
}
static void SetThreadPolicyFunc(pthread_attr_t *pAttr,int
iPolicy)
{
int irs=pthread_attr_setschedpolicy(pAttr,iPolicy);
assert(irs==0);

GetThreadPolicyFunc(pAttr);
}
int main(int argc,char *argv[])
{
pthread_attr_t pAttr;
struct sched_param sched;
```

```
int irs=pthread_attr_init(&pAttr);
assert(irs==0);

int iPlicy=GetThreadPolicyFunc(&pAttr);

printf("\nExport current Configuration of priority.\n");
PrintThreadPriorityFunc(&pAttr,iPlicy);

printf("\nExport SCHED_FIFO of prioirty.\n");
PrintThreadPriorityFunc(&pAttr,SCHED_FIFO);

printf("\nExport SCHED_RR of prioirty.\n");
PrintThreadPriorityFunc(&pAttr,SCHED_RR);


printf("\nExport priority of current thread.\n");
int iPriority=GetThreadPriorityFunc(&pAttr);
printf("Set thread policy.\n");

printf("\nSet SCHED_FIFO policy.\n");
SetThreadPolicyFunc(&pAttr,SCHED_FIFO);
```

```
printf("\nSet SCHED_RR policy.\n");
SetThreadPolicyFunc(&pAttr,SCHED_RR);

printf("\nRestore current policy.\n");
SetThreadPolicyFunc(&pAttr,iPlicy);

irs=pthread_attr_destroy(&pAttr);
assert(irs==0);

return 0;
}
```

运行结果如下：



2、简单线程调度策略，我们创建三个线程，默认创建的线程它的调度策略为 SCHED_OTHER，另外两个线程调度策略为 SCHED_RR/FIFO。具体源码如下：

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <pthread.h>
void TestThread1Func()
```

```
{
sleep(1);
int i,j;
int iPolicy;
struct sched_param sParam;
pthread_getschedparam(pthread_self(),&iPolicy,&sParam);

if(iPolicy==SCHED_OTHER)
   printf("SCHED_OTHER.\n");

if(iPolicy==SCHED_FIFO)
   printf("SCHED_FIFO.\n");

if(iPolicy==SCHED_RR)
   printf("SCHED_RR TEST001.\n");

for(i=1;i<=5;i++)
{
   for(j=1;j<=5000000;j++){}
   printf("Execute thread function 1.\n");
}
printf("Pthread 1 Exit.\n\n");
```

```
}
void TestThread2Func()
{
sleep(2);
int i,j;
int iPolicy;
struct sched_param sParam;
pthread_getschedparam(pthread_self(),&iPolicy,&sParam);

if(iPolicy==SCHED_OTHER)
   printf("SCHED_OTHER.\n");

if(iPolicy==SCHED_FIFO)
   printf("SCHED_FIFO.\n");

if(iPolicy==SCHED_RR)
   printf("SCHED_RR TEST002.\n");

for(i=1;i<=6;i++)
{
   for(j=1;j<=6000000;j++){}
   printf("Execute thread function 2.\n");
```

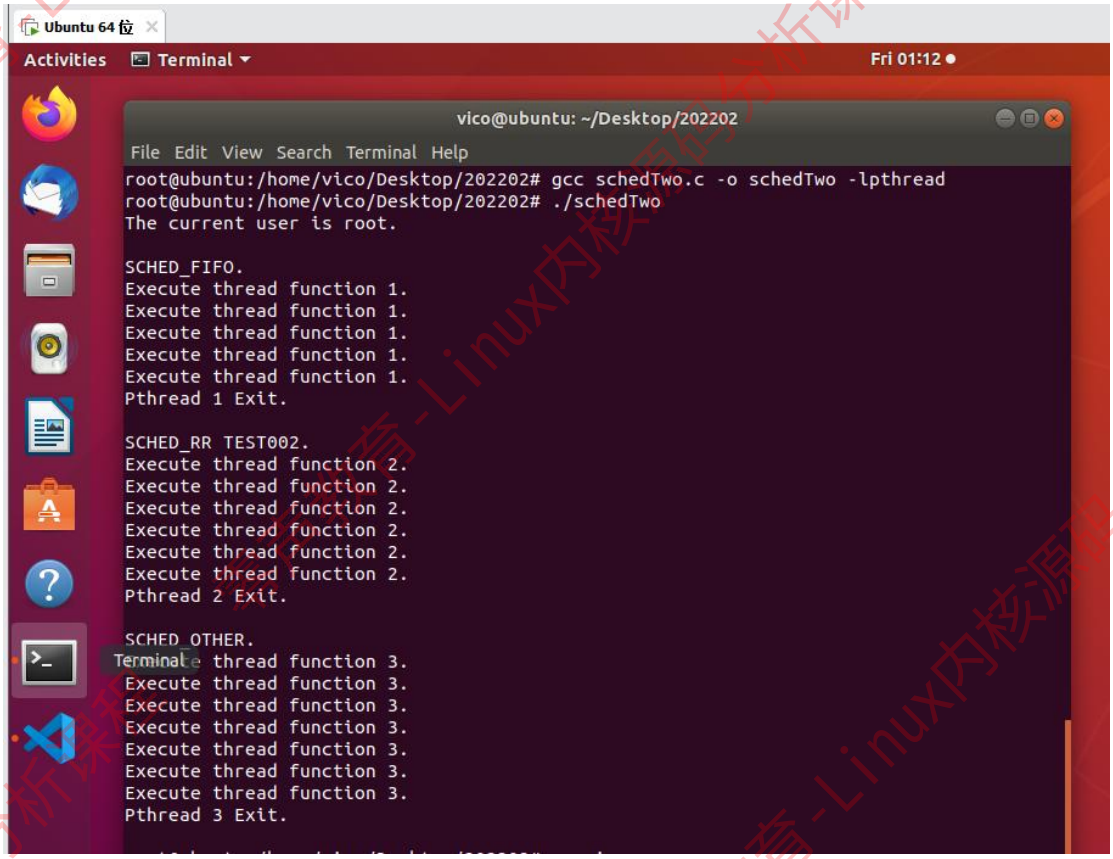
```
}
printf("Pthread 2 Exit.\n\n");
}
void TestThread3Func()
{
sleep(3);
int i,j;
int iPolicy;
struct sched_param sParam;
pthread_getschedparam(pthread_self(),&iPolicy,&sParam);

if(iPolicy==SCHED_OTHER)
   printf("SCHED_OTHER.\n");

if(iPolicy==SCHED_FIFO)
   printf("SCHED_FIFO.\n");

if(iPolicy==SCHED_RR)
   printf("SCHED_RR TEST003.\n");

for(i=1;i<=7;i++)
{
```

```
    for(j=1;j<=7000000;j++){}
    printf("Execute thread function 3.\n");
}
printf("Pthread 3 Exit.\n\n");
}
int main(int argc,char* argv[])
{
int i=0;
i=getuid();
if(0==i)
    printf("The current user is root.\n\n");
else
    printf("The current user is not root.\n\n");

pthread_t ppid1,ppid2,ppid3;
struct sched_param sParam;
pthread_attr_t pAttr1,pAttr2,pAttr3;

pthread_attr_init(&pAttr1);
pthread_attr_init(&pAttr2);
pthread_attr_init(&pAttr3);
```

```
sParam.sched_priority=31;
pthread_attr_setschedpolicy(&pAttr2,SCHED_RR);
pthread_attr_setschedparam(&pAttr2,&sParam);
pthread_attr_setinheritsched(&pAttr2,PTHREAD_EXPLICIT_SCH
ED);

sParam.sched_priority=11;
pthread_attr_setschedpolicy(&pAttr1,SCHED_FIFO);
pthread_attr_setschedparam(&pAttr1,&sParam);
pthread_attr_setinheritsched(&pAttr1,PTHREAD_EXPLICIT_SCH
ED);

pthread_create(&ppid3,&pAttr3,(void*)TestThread3Func,NULL)
;
pthread_create(&ppid2,&pAttr2,(void*)TestThread2Func,NULL)
;
pthread_create(&ppid1,&pAttr1,(void*)TestThread1Func,NULL)
;

pthread_join(ppid3,NULL);
pthread_join(ppid2,NULL);
pthread_join(ppid1,NULL);
```

```
pthread_attr_destroy(&pAttr3);
pthread_attr_destroy(&pAttr2);
pthread_attr_destroy(&pAttr1);

return 0;
}
```

运行结果 1（管理员权限）：



运行结果 2（普通用户）：