江上之明月 L31 2016-12-29 17:39 Linux 计算机专业课

实验代码

“操作系统原理与实践”实验报告

进程运行轨迹的跟踪与统计

进程运动轨迹的跟踪与统计

实验内容

基于模板“process.c”编写多进程的样本程序

在linux0.11上实现进程运动轨迹的跟踪

在修改过的0.11 允许样本程序，通过分析log文件，统计该程序建立的所有进程的等待时间、完成实际和运行实际，然后计算平均等待时间，平均完成时间和吞吐量

修改0.11进程调度的时间片，然后再运行同样的样本程序，再次统计

1.编写process.c

从/home/teacher目录拷贝process.c到自己的oslab目录，只需要完成main函数创建几个进程即可，其他函数已源文件已完成。代码如下

```
int main(int argc, char * argv[])
{
    int i,exit_code;
    pid_t pid[4],pid_chid;
    char param[][3]={
        {10, 1, 0},
        {10, 0, 1},
        {10, 1, 1},
        {10, 1, 9},
    };

    memset(pid, 0,sizeof(pid));
    for (i=0; i< 4; i++){
        printf ("pid=%d i=%d\r\n", getpid(), i);
        if ( (pid[i] = fork ()) == 0){
            printf ("child %d run\r\n", getpid());
            cpuio_bound(param[i][0],param[i][1],param[i][2]);
            exit (0);
        }
    }

    for (i= 0; i< 4; i++){
        pid_chid= wait (&exit_code);
        if (pid_chid > 0)
            printf ("child %d exit with code %d\r\n", pid_chid, exit_code);
    }

    return 0;
}
```

2. 实现输出日志到/var/process.log

首先在printk.c文件添加fprintk函数

```
#include "linux/sched.h"
#include "sys/stat.h"

static char logbuf[1024];
int fprintk int const char
{
va_list args;
int count;
struct file * file;
struct m_inode * inode;

va_start(args, fmt);
count= vsprintf (logbuf, fmt, args);
va_end(args);

if (fd < 3) /* 如果输出到stdout或stderr，直接调用sys_write即可 */
{
__asm__( "push %%fs\n\t"
"push %%ds\n\t"
"pop %%fs\n\t"
"pushl %0\n\t"
"pushl $logbuf\n\t" /* 注意对于Windows环境来说，是_logbuf，下同 */
"pushl %1\n\t"
"call sys_write\n\t" /* 注意对于Windows环境来说，是_sys_write，下同 */
"addl $8,%%esp\n\t"
"popl %0\n\t"
"pop %%fs"
:: "r" (count), "r" (fd): "ax", "cx", "dx" );
}
else /* 假定>=3的描述符都与文件关联。事实上，还存在很多其它情况，这里并没有考虑。*/
{
if (!(file=task[0]->filp[fd])) /* 从进程0的文件描述符表中得到文件句柄 */
return 0;
inode=file->f_inode;

__asm__( "push %%fs\n\t"
"push %%ds\n\t"
"pop %%fs\n\t"
"pushl %0\n\t"
"pushl $logbuf\n\t"
"pushl %1\n\t"
"pushl %2\n\t"
"call file_write\n\t"
"addl $12,%%esp\n\t"
"popl %0\n\t"
"pop %%fs"
:: "r" (count), "r" (file), "r" (inode): "ax", "cx", "dx" );
}
return count;
}
```

接着修改init函数中打开stdin，stdout，stderr部分带代码，将这部分代码提前到fork之前。

在init 中修改

```
void init void
{
    int pid,i;

//    setup((void *) &drive_info);
//    (void) open("/dev/tty0",O_RDWR,0);
//    (void) dup(0);
//    (void) dup(0);
...
}
```

在main函数中修改

```
vod main void
{
    ...

    /***************添加开始***************/
    setup((void *) &drive_info);
    (void) open("/dev/tty0",O_RDWR,0); //建立文件描述符0和/dev/tty0的关联
    (void) dup(0); //文件描述符1也和/dev/tty0关联
    (void) dup(0); //文件描述符2也和/dev/tty0关联
    (void) open("/var/process.log",O_CREAT|O_TRUNC|O_WRONLY, 0666);
    /***************添加结束***************/

    if (!fork()) {    /* we count on this going ok */
        init();
    }

    ...
}
```

3 添加进程的运动轨迹

进程的状态是N,J,R,W和E中的任意一个，分别表示进程新建(N)、进入就绪态(J)、进入运行态(R)、进入阻塞态(W)和退出(E)；

具体代码修改请查看代码补丁。

进程调用fork创建一个新进程调用copy_process,在copy_process将新建一个进程 并使进程进入就绪态。所有在copy_process 要打印新建(N),进入就绪态（J）

schedule函数将进行进程切换。 当一个进程的时间片用完后、进程进入阻塞态、进程退出和进程进行系统调用的时候，就会调用schedule进行进程切换。如果是时间片用完，进程会从运行态进入就绪态。如果调用IO操作，就会进入阻塞。

进程运行时会被时间中断打断，在时钟中断处理函数中，如果检查到进程的时间片已经用完了，就会调用 schedule进程进程切换，进程由运行态（R）进入就绪态(J)。

进入阻塞态的原因有进程主动休眠sys_pause、sleep_on、interruptible_sleep_on。

sleep_on和interruptible_sleep_on函数，进程将由运行态（N）进入阻塞态（W）。在这两个函数中进程不但要切换本身进程的状态，还需要将等待

进程退出调用do_exit,进程由运行态（N）切换到退出态（E）。

进程调用sys_waitpid函数由运行态（N）进入阻塞态（W），直到子进程返回。

Wake_up进程唤醒等待队列中的进程，使其从阻塞态（W）进入就绪态（J）

打印的process.log 部分截图：

linux启动

运行process程序

问题

1.结合自己的体会，谈谈从程序设计者的角度看，单进程编程和多进程编程最大的区别是什么？ A：多进程编程需要考虑资源是否被别的进程占用，进程的同步，系统的实时性。

2.你是如何修改时间片的？仅针对样本程序建立的进程，在修改时间片前后，log文件的统计结果（不包括Graphic）都是什么样？结合你的修改分析一下为什么会这样变化，或者为什么没变化？

代码补丁：

```
From fb88965f299e4f4e830c6009895e07f869c217ee Mon Sep 17 00:00:00 2001
From: justin <xiazhiping@foxmail.com>
Date: Wed, 28 Dec 2016 18:06:49 +0800
Subject: [PATCH 1/2] print process


 linux-0.11/include/linux/kernel.h |  1 +
 linux-0.11/init/main.c|  18 +++++++++++
 linux-0.11/kernel/exit.c  |  3 +++
 linux-0.11/kernel/fork.c  |  7 ++++++
 linux-0.11/kernel/printk.c|  51 ++++++++++++++++++++++++++++++++++++++++
 linux-0.11/kernel/sched.c  | 39 +++++++++++++++++++++++++++
 6 files changed, 101 insertions(+), 13 deletions(-)

diff --git a/linux-0.11/include/linux/kernel.h b/linux-0.11/include/linux/kernel.h
index cb40dd5..feb7d84 100644
--- a/linux-0.11/include/linux/kernel.h
+++ b/linux-0.11/include/linux/kernel.h
@@ -5,6 +5,7 @@ void verify_area(void * addr, int count);
 volatile void panic(const char * str);
 int printf(const char * fmt, ...);
 int printk(const char * fmt, ...);
+int fprintk(int fd, const char *fmt, ...);
 int tty_write(unsigned ch, char * buf, int count);
 void * malloc(unsigned int size);
 void free_s(void * obj, int size);
diff --git a/linux-0.11/init/main.c b/linux-0.11/init/main.c
index 25de9e0..84da4f4 100644
--- a/linux-0.11/init/main.c
+++ b/linux-0.11/init/main.c
@@ -135,6 +135,15 @@ void main(void) /* This really IS void, no error here. */
     floppy_init();
     sti();
     move_to_user_mode();
+
+    /***************添加开始***************/
+    setup((void *) &drive_info);
+    (void) open("/dev/tty0",O_RDWR,0); //建立文件描述符0和/dev/tty0的关联
+    (void) dup(0); //文件描述符1也和/dev/tty0关联
+    (void) dup(0); //文件描述符2也和/dev/tty0关联
+    (void) open("/var/process.log",O_CREAT|O_TRUNC|O_WRONLY, 0666);
+    /***************添加结束***************/
+
     if (!fork()) {    /* we count on this going ok */
         init();
     }
@@ -169,10 +178,11 @@ void init(void)
 {
     int pid,i;

-    setup((void *) &drive_info);
-    (void) open("/dev/tty0",O_RDWR,0);
-    (void) dup(0);
-    (void) dup(0);
+//    setup((void *) &drive_info);
+//    (void) open("/dev/tty0",O_RDWR,0);
+//    (void) dup(0);
+//    (void) dup(0);
+
     printf("%d buffers = %d bytes buffer space\n\r",NR_BUFFERS,
         NR_BUFFERS*BLOCK_SIZE);
     printf("Free mem: %d bytes\n\r",memory_end-main_memory_start);
diff --git a/linux-0.11/kernel/exit.c b/linux-0.11/kernel/exit.c
index b22de34..0c41f74 100644
--- a/linux-0.11/kernel/exit.c
+++ b/linux-0.11/kernel/exit.c
@@ -129,6 +129,9 @@ int do_exit(long code)
     current->state = TASK_ZOMBIE;
     current->exit_code = code;
     tell_father(current->father);
+
+    fprintk(3, "%ld\t%c\t%ld\n", current->pid, 'E', jiffies); //向log文件输出跟踪进程运行轨迹
+
     schedule();
     return (-1);    /* just to suppress warnings */
 }
diff --git a/linux-0.11/kernel/fork.c b/linux-0.11/kernel/fork.c
index 2486b13..c8e76ab 100644
--- a/linux-0.11/kernel/fork.c
+++ b/linux-0.11/kernel/fork.c
@@ -111,6 +111,10 @@ int copy_process(int nr, long ebp, long edi, long esi, long gs, long none,
     p->tss.gs = gs & 0xffff;
     p->tss.ldt = _LDT(nr);
     p->tss.trace_bitmap = 0x80000000;
+
+    fprintk(3, "%ld\t%c\t%ld\n", p->pid, 'N', jiffies); //向log文件输出跟踪进程运行轨迹
+
+
     if (last_task_used_math == current)
         __asm__("clts ; fnsave %0"::"m" (p->tss.i387));
     if (copy_mem(nr,p)) {
@@ -130,6 +134,9 @@ int copy_process(int nr, long ebp, long edi, long esi, long gs, long none,
     set_tss_desc(gdt+(nr<<1)+FIRST_TSS_ENTRY,&(p->tss));
     set_ldt_desc(gdt+(nr<<1)+FIRST_LDT_ENTRY,&(p->ldt));
     p->state = TASK_RUNNING;    /* do this last, just in case */
+
+    //fprintk(1, "The ID of running process is %d\n", current->pid); //向stdout打印正在运行的进程的ID
+    fprintk(3, "%ld\t%c\t%ld\n", p->pid, 'J', jiffies); //向log文件输出跟踪进程运行轨迹
     return last_pid;
 }

diff --git a/linux-0.11/kernel/printk.c b/linux-0.11/kernel/printk.c
index 0daa097..1e88036 100644
--- a/linux-0.11/kernel/printk.c
+++ b/linux-0.11/kernel/printk.c
@@ -39,3 +39,54 @@ int printk(const char *fmt, ...)
         :: "r" (i):"ax","cx","dx");
     return i;
 }
+
+
+#include "linux/sched.h"
+#include "sys/stat.h"
+static char logbuf[1024];
+int fprintk(int fd, const char *fmt, ...)
+{
+va_list args;
+int count;
+struct file * file;
+struct m_inode * inode;
+
+va_start(args, fmt);
+count=vsprintf(logbuf, fmt, args);
+va_end(args);
+
+if (fd < 3) /* 濡傛灉杈撳嚭鍒皊tdout鎴杝tderr锛岀洿鎺ヨ皟鐢╯ys_write鍗冲彲 */
+{
+__asm__("push %%fs\n\t"
+"push %%ds\n\t"
+"pop %%fs\n\t"
+"pushl %0\n\t"
+"pushl $logbuf\n\t" /* 娉ㄦ剰瀵逛簬Windows鐜鏉ヨ锛屾槸_logbuf锛屼笅鍚? */
+"pushl %1\n\t"
+"call sys_write\n\t" /* 娉ㄦ剰瀵逛簬Windows鐜鏉ヨ锛屾槸_sys_write锛屼笅鍚? */
+"addl $8,%%esp\n\t"
+"popl %0\n\t"
+"pop %%fs"
+:: "r" (count), "r" (fd): "ax", "cx", "dx");
+}
+else /* 鍋囧畾>=3鐨勬弿杩扮閮戒笌鏂囦欢鍏宠仈銆備簨瀹炰笂锛岃繕瀛樺湪寰堝鍏跺畠鎯呭喌锛岃繖閲屽苟娌℃湁鑰冭檻銆?/
+{
+if (!(file=task[0]->filp[fd])) /* 浠庤繘绋?鐨勬枃浠舵弿杩扮琛ㄤ腑寰楀埌鏂囦欢鍙ユ焺 */
+return 0;
+inode=file->f_inode;
+
+__asm__("push %%fs\n\t"
+"push %%ds\n\t"
+"pop %%fs\n\t"
+"pushl %0\n\t"
+"pushl $logbuf\n\t"
+"pushl %1\n\t"
+"pushl %2\n\t"
+"call file_write\n\t"
+"addl $12,%%esp\n\t"
+"popl %0\n\t"
+"pop %%fs"
+:: "r" (count), "r" (file), "r" (inode): "ax", "cx", "dx");
+}
+return count;
+}
diff --git a/linux-0.11/kernel/sched.c b/linux-0.11/kernel/sched.c
index 15d839b..7c61215 100644
--- a/linux-0.11/kernel/sched.c
+++ b/linux-0.11/kernel/sched.c
@@ -115,8 +115,10 @@ void schedule(void)
                 (*p)->alarm = 0;
             }
             if (((*p)->signal & ~(_BLOCKABLE & (*p)->blocked)) &&
-            (*p)->state==TASK_INTERRUPTIBLE)
+            (*p)->state==TASK_INTERRUPTIBLE){
                 (*p)->state=TASK_RUNNING;
+                fprintk(3, "%ld\t%c\t%ld\n", (*p)->pid, 'J', jiffies); //向log文件输出跟踪进程运行轨迹
+            }
         }

 /* this is the scheduler proper: */
@@ -132,17 +134,30 @@ void schedule(void)
             if ((*p)->state == TASK_RUNNING && (*p)->counter > c)
                 c = (*p)->counter, next = i;
         }
+
         if (c) break;
         for(p = &LAST_TASK ; p > &FIRST_TASK ; --p)
             if (*p)
                 (*p)->counter = ((*p)->counter >> 1) +
                         (*p)->priority;
     }
+
+    if (task[next]->pid !=current->pid){
+        if (current->state == TASK_RUNNING){
+            fprintk(3, "%ld\t%c\t%ld\t%ld\n", current->pid, 'J', jiffies, current->counter);//向log文件输出跟踪进程运行轨迹
+        }
+        if (task[next]->state == TASK_RUNNING){
+            fprintk(3, "%ld\t%c\t%ld\n", task[next]->pid, 'R', jiffies); //向log文件输出跟踪进程运行轨迹
+        }
+    }
     switch_to(next);
 }

 int sys_pause(void)
 {
+    if (current->state != TASK_INTERRUPTIBLE){
+        fprintk(3, "%ld\t%c\t%ld\n", current->pid, 'W', jiffies); //向log文件输出跟踪进程运行轨迹
+    }
     current->state = TASK_INTERRUPTIBLE;
     schedule();
     return 0;
@@ -154,14 +169,21 @@ void sleep_on(struct task_struct **p)

     if (!p)
         return;
+
+
     if (current == &(init_task.task))
         panic("task[0] trying to sleep");
     tmp = *p;
     *p = current;
+    if (current->state != TASK_UNINTERRUPTIBLE)
+        fprintk(3, "%ld\t%c\t%ld\n", current->pid, 'W', jiffies);
     current->state = TASK_UNINTERRUPTIBLE;
     schedule();
-    if (tmp)
+    if (tmp){
+        if (tmp->state != TASK_RUNNING)
+            fprintk(3, "%ld\t%c\t%ld\n", tmp->pid, 'J', jiffies); //向log文件输出跟踪进程运行轨迹
         tmp->state=0;
+    }
 }

 void interruptible_sleep_on(struct task_struct **p)
@@ -170,24 +192,33 @@ void interruptible_sleep_on(struct task_struct **p)

     if (!p)
         return;
+
     if (current == &(init_task.task))
         panic("task[0] trying to sleep");
     tmp=*p;
     *p=current;
-repeat:    current->state = TASK_INTERRUPTIBLE;
+
+repeat:
+    if (current->state != TASK_INTERRUPTIBLE)
+        fprintk(3, "%ld\t%c\t%ld\n", current->pid, 'W', jiffies); //向log文件输出跟踪进程运行轨迹
+    current->state = TASK_INTERRUPTIBLE;
     schedule();
     if (*p && *p != current) {
         (**p).state=0;
         goto repeat;
     }
     *p=NULL;
-    if (tmp)
+    if (tmp){
+        if (tmp->state!=TASK_RUNNING)
+            fprintk(3, "%ld\t%c\t%ld\n", tmp->pid, 'J', jiffies); //向log文件输出跟踪进程运行轨迹
         tmp->state=0;
+    }
 }

 void wake_up(struct task_struct **p)
 {
     if (p && *p) {
+        fprintk(3, "%ld\t%c\t%ld\n", (*p)->pid, 'W', jiffies); //向log文件输出跟踪进程运行轨迹
         (**p).state=0;
         *p=NULL;
     }

--
1.9.4.msysgit.1
```