**Subject:** Practice on Operating System Lecture Practice Thread
**From:** IKENOYA Katsutoshi <j05002@ie.u-ryukyu.ac.jp>
**Date:** Sun, 14 Jan 2007 19:30:06 +0900
**To:** Shinji KONO <kono@ie.u-ryukyu.ac.jp>

学籍番号 : 055702B

・開発環境
OS : Mac OS X 10.4.8
コンパイラ :gcc version 4.0.1 (Apple Computer, Inc. build 5250)
java version "1.5.0_06"

・実行環境
同上,pw002

---
１.軽量プロセスと手続きの比較

・実行結果(nw0502)

```
[j05002@5-process]% ./mp-create
creating process_A() ...
done.
creating process_B() ...
done.
creating process_C() ...
done.
thread 41944064 active.
process_A( 0,1,2 ) created.
&x == 0xf0101e0c
thread 41945088 active.
process_A() exit.
process_B( 1,3,5 ) created.
&x == 0xf0182e0c
thread 41946112 active.
process_B() exit.
process_C( 2,3,5 ) created.
&x == 0xf0203e0c
thread 41947136 active.
process_C() exit.
mp_user_main() exit.
[j05002@5-process]% ./function-call
calling function_A() ...
function_A( 0,1,2 ) called.
&x == 0xbffffa28
done.
calling function_B() ...
function_B( 1,3,5 ) called.
&x == 0xbffffa28
done.
calling function_C() ...
function_C( 2,3,5 ) called.
&x == 0xbffffa28
done.
```

・実行結果(pw002)

```
[j05002@pw002 ~/5-process]% ./mp-create
creating process_A() ...
done.
creating process_B() ...
process_A( 0,1,2 ) created.
```

```
&x == 0x411b2ac8
thread 32771 active.
process_A() exit.
process_B( 1,3,5 ) created.
&x == 0x419b2ac8
thread 49156 active.
process_B() exit.
done.
creating process_C() ...
process_C( 2,3,5 ) created.
&x == 0x421b2ac8
thread 65541 active.
process_C() exit.
done.
thread 16386 active.
mp_user_main() exit.
[j05002@pw002 ~/5-process]% ./function-call
calling function_A() ...
function_A( 0,1,2 ) called.
&x == 0xbffff274
done.
calling function_B() ...
function_B( 1,3,5 ) called.
&x == 0xbffff274
done.
calling function_C() ...
function_C( 2,3,5 ) called.
&x == 0xbffff274
done.
```

・考察
mp-createでは、各プロセスに用いた時間を見てみるとあまり差がない。
このことより、プロセスが並列に実行されていることがわかる。
また&xがprovess_A,B,Cで0x00081000ずつ増加している。
pwにおいて実行した結果、生成や実行の順番が規則正しくなっていない。
これはthreadがkernel level であるため、kernelによって制御されている
ためである。

function-callでは各関数が呼び出され、終了するまで他の関数が呼び出されたり
することがないので、逐次実行されていることがわかる。
また&xがfunction_A,B,Cで変わっていないので同じアドレスにおいて実行されている
ことがわかる。

---

２.プロセスのコンテキスト切り替えの観察

・実行結果

```
[j05002@5-process]% ./mp-yield
*** process_A() looped 0 times.
*** process_B() looped 0 times.
*** process_A() looped 1 times.
*** process_B() looped 1 times.
*** process_A() looped 2 times.
*** process_B() looped 2 times.
*** process_A() looped 3 times.
*** process_B() looped 3 times.
*** process_A() looped 4 times.
*** process_B() looped 4 times.
```

２つの軽量プロセスがsched_yield()によって交互に動作しているのが分かる。

---

３.3つのプロセスの間のコンテキスト切り替え

・作成したプログラム

```
/*
mp-yield.c -- 3つのプロセスの間を行き来するプログラム
$Header: /home/y05/j05002/5-process/mp-yield3.c
*/

#include <stdio.h>
#include <pthread.h>
#ifdef sun
#define sched_yield thr_yield
#endif


void *process_A(void *);
void *process_B(void *);
void *process_C(void *);
pthread_t thread_a,thread_b,thread_c;

int
main(int ac,char *av[])
{
省略
pthread_create(&thread_c, NULL, &process_C, NULL);
省略
pthread_join(thread_c, NULL);
}

省略

void *
process_C(void *arg)
{
int i ;
for( i=0 ; i<5 ; i++ )
{
fprintf(stderr,"*** process_C() looped %d times.¥n", i );
sched_yield();
}
}
```

・実行結果

```
[j05002@5-process]% ./mp-yield3
*** process_A() looped 0 times.
*** process_B() looped 0 times.
*** process_C() looped 0 times.
*** process_A() looped 1 times.
*** process_B() looped 1 times.
*** process_C() looped 1 times.
*** process_A() looped 2 times.
*** process_B() looped 2 times.
*** process_C() looped 2 times.
*** process_A() looped 3 times.
*** process_B() looped 3 times.
*** process_C() looped 3 times.
*** process_A() looped 4 times.
*** process_B() looped 4 times.
*** process_C() looped 4 times.
```

結果より３つの軽量プロセスが交互に実行されていることがわかる。

---

４.FIFOスケジューリングの観察

・実行結果

(1)A,B,Cのpriorityがそれぞれ10,20,30の場合(デフォルト)

```
[j05002@5-process]% ./mp-wakeup
creating process_A() ...
done.
creating process_B() ...
done.
creating process_C() ...
done.
thread 41944064 active.
process_C( 2,3,5 ) created.
thread 41947136 active.
process_C() exit.
process_B( 1,3,5 ) created.
thread 41946112 active.
process_B() exit.
process_A( 0,1,2 ) created.
thread 41945088 active.
process_A() exit.
mp_wakeup C, B, A.
thread 41944064 active.
mp_user_main() exit.
```

(2)A,B,Cのpriorityがそれぞれ30,20,10の場合

```
[j05002@5-process]% ./mp-wakeup
creating process_A() ...
done.
creating process_B() ...
done.
creating process_C() ...
done.
thread 41944064 active.
process_A( 0,1,2 ) created.
thread 41945088 active.
process_A() exit.
process_B( 1,3,5 ) created.
thread 41946112 active.
process_B() exit.
process_C( 2,3,5 ) created.
thread 41947136 active.
process_C() exit.
mp_wakeup C, B, A.
thread 41944064 active.
mp_user_main() exit.
```

(3)A,B,Cのpriorityがそれぞれ30,20,30の場合

```
[j05002@5-process]% ./mp-wakeup
creating process_A() ...
done.
creating process_B() ...
done.
creating process_C() ...
done.
thread 41944064 active.
process_A( 0,1,2 ) created.
thread 41945088 active.
process_A() exit.
```

```
process_C( 2,3,5 ) created.
thread 41947136 active.
process_C() exit.
process_B( 1,3,5 ) created.
thread 41946112 active.
process_B() exit.
mp_wakeup C, B, A.
thread 41944064 active.
mp_user_main() exit.
```

(4)A,B,Cのpriorityがそれぞれ20,30,30の場合

```
[j05002@5-process]% ./mp-wakeup
creating process_A() ...
done.
creating process_B() ...
done.
creating process_C() ...
done.
thread 41944064 active.
process_B( 1,3,5 ) created.
thread 41946112 active.
process_B() exit.
process_C( 2,3,5 ) created.
thread 41947136 active.
process_C() exit.
process_A( 0,1,2 ) created.
thread 41945088 active.
process_A() exit.
mp_wakeup C, B, A.
thread 41944064 active.
mp_user_main() exit.
```

・考察

mp-createの場合は軽量プロセスA,B,Cが順番に実行されたが、mp-wakeupではデフォルトの場合でC,B,Aの順番で実行されている。これは各軽量プロセスにpriorityが設定されており、priorityの高い順に実行されているためである。priorityの値が同じ場合は、生成された順番が早い方が先に実行される。