

OS Assignment 2

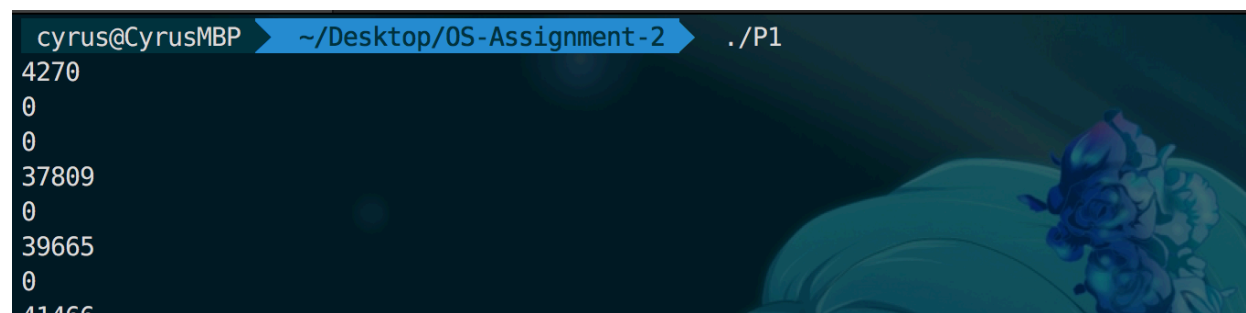
Problem 1

在Linux操作系统中实现2.1节中例子，运行多次，每次的结果有什么不同？

Codes :

```
1  #include <pthread.h>
2  #include <stdio.h>
3
4  int n = 0;
5
6  void *A(void *arg) {
7      while (1) {
8          n++;
9      }
10 }
11
12 void *B(void *arg) {
13     while (1) {
14         printf("%d\n", n);
15         n = 0;
16     }
17 }
18
19
20 int main(void) {
21     pthread_t p1, p2;
22     pthread_create(&p1, NULL, A, NULL);
23     pthread_create(&p2, NULL, B, NULL);
24     pthread_join(p1, NULL);
25     pthread_join(p2, NULL);
26 }
```

Result 1:



```
cyrus@cyrusMBP ~/Desktop/OS-Assignment-2 ./P1
4270
0
0
37809
0
39665
0
41466
```

Result 2:

```
cyrus@CyrusMBP ~/Desktop/OS-Assignment-2 ./P1
5316
0
0
0
0
0
0
```

Problem 2

阅读Linux3.0以上版本的内核代码中进程控制块和进程调度的代码，然后回答下面的问题：

Linux的进程控制块的组织方式是什么？

请问它里面设定了那些进程状态，这些状态代表什么意义？

状态之间如何转换？并画出状态转换图。

Linux的进程控制块的组织方式是什么？

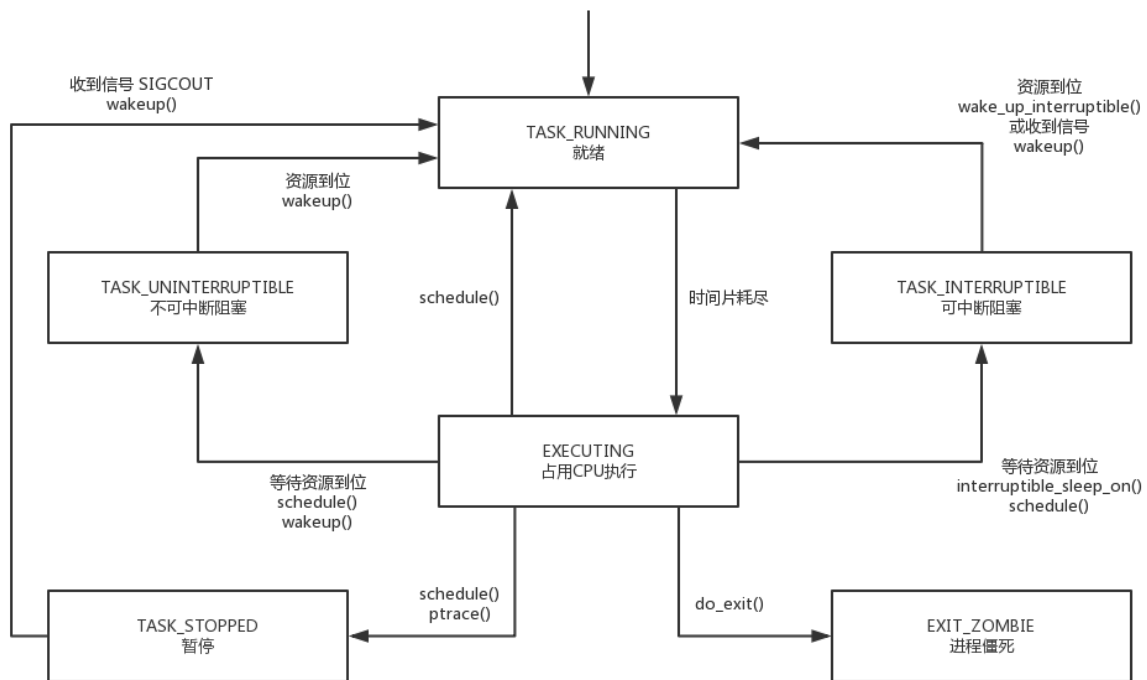
Linux 使用了一个双向循环列表来组织PCB。这个双向循环列表的头和尾都是一个叫做 `init_task` 的，pid 为 0 的 PCB (https://github.com/torvalds/linux/blob/e0d072250a54669dce876d8ade70e417356aae74/init/init_task.c#L19)。Linux 中也有一个单独的运行队列，只存放处于 TASK_RUNNING 状态的进程，使用双向循环列表实现。还有一个等待队列，由循环列表实现。

请问它里面设定了那些进程状态，这些状态代表什么意义？

```
1 #define TASK_RUNNING      0
2 #define TASK_INTERRUPTIBLE 1
3 #define TASK_UNINTERRUPTIBLE 2
4 #define __TASK_STOPPED    4
5 #define __TASK_TRACED     8
6 /* in tsk->exit_state */
7 #define EXIT_ZOMBIE       16
8 #define EXIT_DEAD        32
```

- TASK_RUNNING: 进程正在执行或者正在准备执行
- TASK_INTERRUPTIBLE: 可中断的阻塞状态
- TASK_UNINTERRUPTIBLE: 不可中断的阻塞状态
- __TASK_STOPPED: 收到SIGSTOP、SIGTSTP、SIGTTIN、SIGTTOU等信号
- __TASK_TRACED: 使用 gdb 跟踪进程
- EXIT_DEAD: 进程死掉了
- EXIT_ZOMBIE: 进程死掉了，父进程忽略了他的退出信号，成为僵尸进程

状态之间如何转换？并画出状态转换图？



Problem 3

Codes :

```

1  #include <stdio.h>
2  #include <unistd.h>
3  #include <sys/wait.h>
4
5  int main(void) {
6
7      int child;
8
9      char * argv[] = {"echo", "Hello, World!", NULL};
10     char * envp[] = {NULL};
11
12     if (!(child = fork())) {
13         printf("pid %d: pid %d is my father\n", getpid(), getppid());
14         execve("/bin/echo", argv, envp);
15         printf("pid %d: I'm back, something is wrong!\n", getpid());
16     } else {
17         int selfpid = getpid();
18         printf("pid %d: pid %d is my son\n", selfpid, child);
19         wait4(child, NULL, 0, NULL);
20         printf("pid %d: done\n", selfpid);
21     }
22 }

```

Result :

```
× ..-Assignment-2 (z... 81
cyrus@CyrusMBP ~/Desktop/OS-Assignment-2 ./P3
pid 3593: pid 3594 is my son
pid 3594: pid 3593 is my father
Hello, World!
pid 0: done
cyrus@CyrusMBP ~/Desktop/OS-Assignment-2
```

在以上代码中，可以将 Line 19 改为以下语句来使用 wait4 或 waitpid：

```
1 wait3(&child, 0, NULL);
```

```
1 waitpid(child, &selfpid, 0);
```

Problem 4

用Linux的消息队列机制编程实现生产者-消费者问题。

Codes :

```
1 #include <pthread.h>
2 #include <stdio.h>
3 #include <sys/msg.h>
4 #include <unistd.h>
5
6 typedef struct {
7     long msgtype;
8     char msg;
9 } Message;
10
11 int msgid;
12 int MSGSZ = 1;
13 int MSGTYPE = 233;
14
15 void producter(void *arg) {
16     Message msg;
17     msg.msgtype = MSGTYPE;
18     FILE *fp;
19     fp = fopen("test.txt", "r");
20     char ch;
21     while ((ch = fgetc(fp)) != EOF) {
22         msg.msg = ch;
23         msgsnd(msgid, &msg, MSGSZ, 0);
24         printf("in: %c\n", msg.msg);
25         sleep(1);
26     }
27 }
```

```

28
29 void consumer(void *arg) {
30     Message msg;
31     int ret;
32     while (1) {
33         ret = msgrcv(msgid, &msg, MSGSZ, MSGTYPE, IPC_NOWAIT);
34         if (ret != -1) {
35             printf("\t\tout: %c\n", msg.msg);
36         } else {
37             break;
38         }
39         sleep(2);
40     }
41 }
42
43 int main() {
44     int producer_N = 2;
45     int consumer_N = 3;
46     pthread_t ps[producer_N], cs[consumer_N];
47     int selfpid = getpid();
48     printf("pid: %d\n", selfpid);
49     msgid = msgget(selfpid, IPC_CREAT | 0666);
50     printf("msgid key: %d\n", msgid);
51
52     for (int i = 0; i < producer_N; i++) {
53         pthread_create(&ps[i], NULL, (void *)producer, NULL);
54     }
55     for (int i = 0; i < consumer_N; i++) {
56         pthread_create(&cs[i], NULL, (void *)consumer, NULL);
57     }
58
59     for (int i = 0; i < producer_N; i++) {
60         pthread_join(ps[i], NULL);
61     }
62     for (int i = 0; i < consumer_N; i++) {
63         pthread_join(cs[i], NULL);
64     }
65 }

```

Result :

```
× ..-Assignment-2 (z... 81
cyrus@CyrusMBP ~/Desktop/OS-Assignment-2 ./P4
pid: 859
msgid key: 65538
in: c
in: c
out: c
out: c
in: y
in: y
out: y
out: y
in: r
in: r
in: u
in: u
out: r
out: r
in: s
in: s
out: u
out: u
out: s
out: s
cyrus@CyrusMBP ~/Desktop/OS-Assignment-2
```

test.txt 文件中的内容为：

```
1 | cyrus
```

Problem 5

利用上面提到的系统调用，在Linux系统中实现一个可加载的内核模块，里面至少包含一个内核线程。

os_assignment.c

```
1 | #include <linux/init.h>
2 | #include <linux/kthread.h>
```

```

3  #include <linux/module.h>
4  #include <linux/delay.h>
5
6  static struct task_struct *tsk;
7
8  static int thread_function(void *data) {
9      int time_count = 0;
10     do {
11         printk(KERN_INFO "Hello, World!");
12         msleep(1000);
13     } while (!kthread_should_stop() && time_count <= 15);
14     return time_count;
15 }
16
17 static int hello_init(void) {
18     tsk = kthread_run(thread_function, NULL, "mythread%d", 1);
19     return 0;
20 }
21
22 static void hello_exit(void) {
23     if (!IS_ERR(tsk)) {
24         int ret = kthread_stop(tsk);
25         printk(KERN_INFO "thread function has run %ds\n", ret);
26     }
27 }
28
29 module_init(hello_init);
30 module_exit(hello_exit);

```

Makefile

```

1  obj-m+=os_assignment.o
2
3  all:
4      make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) modules
5  clean:
6      make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) clean

```

Result :

```
cyrus@ubuntu: ~/Desktop/OS-Assignment-2/P5

# cyrus @ ubuntu in ~ [2:45:59]
$ cd Desktop/OS-Assignment-2/P5

# cyrus @ ubuntu in ~/Desktop/OS-Assignment-2/P5 [2:46:15]
$ make
make -C /lib/modules/4.13.0-36-generic/build/ M=/home/cyrus/Desktop/OS-Assignmen
t-2/P5 modules
make[1]: Entering directory '/usr/src/linux-headers-4.13.0-36-generic'
CC [M] /home/cyrus/Desktop/OS-Assignment-2/P5/os_assignment.o
Building modules, stage 2.
MODPOST 1 modules
CC /home/cyrus/Desktop/OS-Assignment-2/P5/os_assignment.mod.o
LD [M] /home/cyrus/Desktop/OS-Assignment-2/P5/os_assignment.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.13.0-36-generic'

# cyrus @ ubuntu in ~/Desktop/OS-Assignment-2/P5 [2:46:18]
$ sudo insmod os_assignment.ko
[sudo] password for cyrus:

# cyrus @ ubuntu in ~/Desktop/OS-Assignment-2/P5 [2:46:43]
$ dmesg | tail
[32026.010953] Hello, World!
[32026.010953] Hello, World!
[32026.010954] Hello, World!
[32026.010954] Hello, World!
[32026.010954] Hello, World!
[32026.010955] Hello, World!
[32026.010955] Hello, World!
[32026.010956] Hello, World!
[32026.010956] Hello, World!
[32026.010956] Hello, World!

# cyrus @ ubuntu in ~/Desktop/OS-Assignment-2/P5 [2:47:13]
$ sudo lsmod| grep os_assignment
os_assignment      16384  0

# cyrus @ ubuntu in ~/Desktop/OS-Assignment-2/P5 [2:47:32]
$ sudo rmmod os_assignment

# cyrus @ ubuntu in ~/Desktop/OS-Assignment-2/P5 [2:47:40]
$ sudo lsmod| grep os_assignment

# cyrus @ ubuntu in ~/Desktop/OS-Assignment-2/P5 [2:48:12] C:1
$
```

注：除 Problem 5 的编译和测试环境为 Ubuntu，其余均为 MacOS。