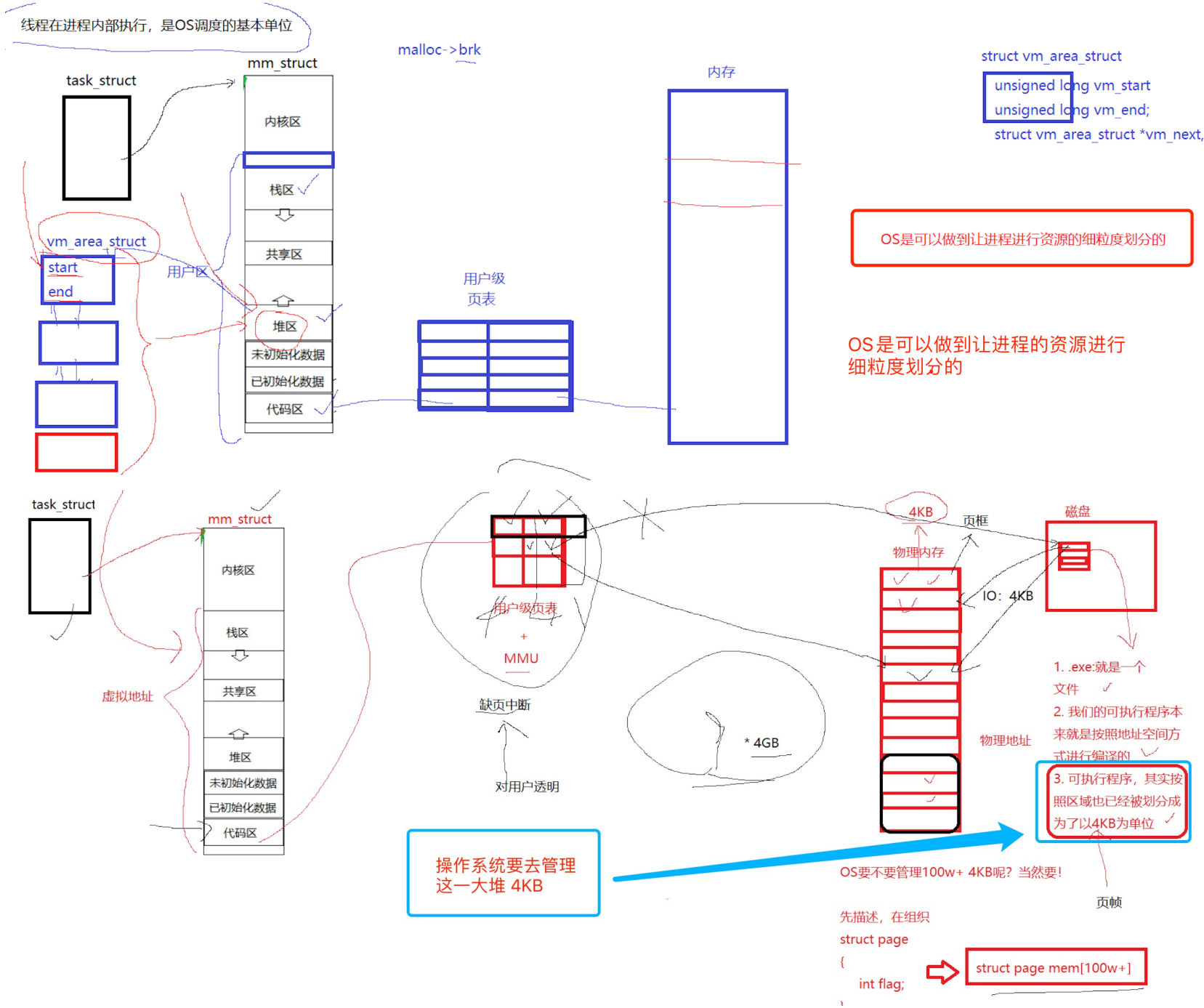
0109线程



先描述再组织

描述这些的叫做page结构体

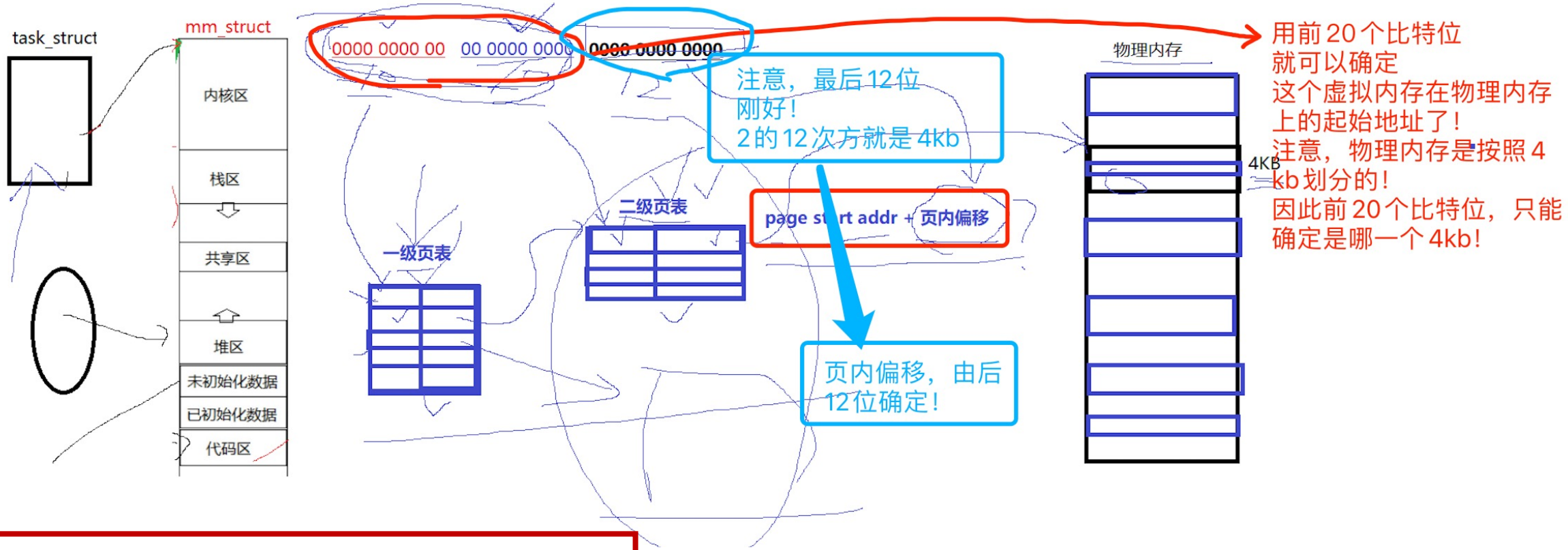
磁盘上的4kb称为页帧

物理内存上的4kb称为页框

把页帧装到页框里（或着反过来）
这叫IO

这两个4kb为单位的IO，都是必须需要OS和编译器支持的

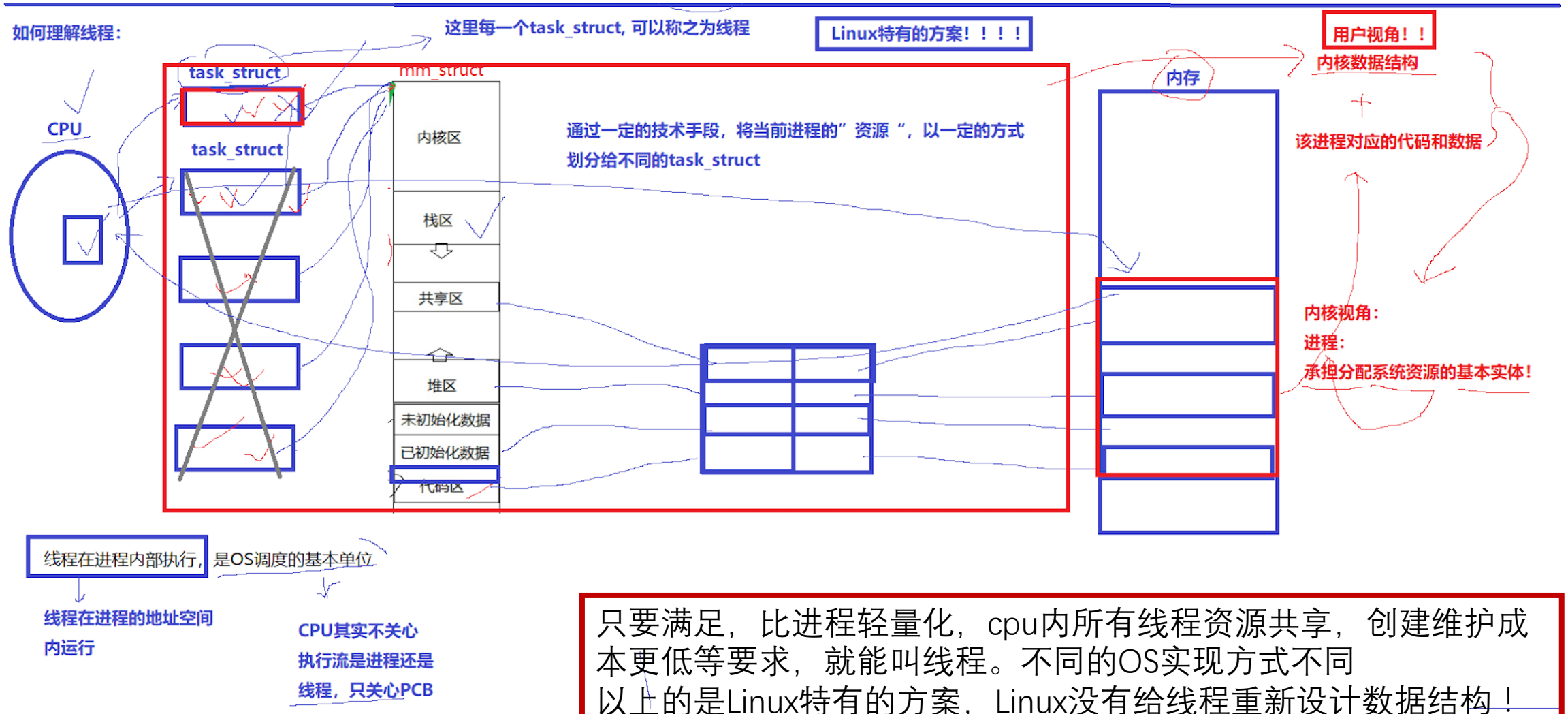
缺页中断
是对用户透明的，用户不知道。



这个才是映射的真正过程！
虚拟地址到物理地址到映射是不准确的！
而是：虚拟地址到特定页的映射！

线程

如何理解线程？



只要满足，比进程轻量化，cpu内所有线程资源共享，创建维护成本更低等要求，就能叫线程。不同的OS实现方式不同
以上的是Linux特有的方案，Linux没有给线程重新设计数据结构！

什么叫做进程？
pcb+地址空间+页表...
重新理解进程（用户视角）
内核数据结构+进程对应的代码和数据
现在可以有多个pcb
内核视角
承担分配系统资源的基本实体！

如何理解以前我们写的和进程相关的代码？
以前学习和用的进程：**内部只有一个执行流的进程**
现在的：内部具有多个执行流的进程

CPU调度的基本单位：线程！

CPU视角：
在Linux下，PCB<=其他OS的PCB！

Linux的进程：统一称之为轻量级进程

Linux没有真正意义上的线程结构，Linux是用进程pcb模拟的线程的

在用户层实现了一套用户层多线程方案
以库的方式提供给用户进行使用，
pthread线程库 --- 原生线程库

```
PTHREAD_CREATE(3)                Linux Programmer's Manual                PTHREAD_CREATE(3)

NAME
       pthread_create - create a new thread

SYNOPSIS
       #include <pthread.h>

       int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
                          void *(*start_routine) (void *), void *arg);

       Compile and link with -pthread.

DESCRIPTION
       The  pthread_create()  function starts a new thread in the calling process.  The new thread starts execution
       by invoking start_routine(); arg is passed as the sole argument of start_routine().

       The new thread terminates in one of the following ways:

       * It calls pthread_exit(3), specifying an exit status value that is available to another thread in the  same
         process that calls pthread_join(3).
```

```
(base) [yufc@VM-12-12-centos:~/Files/BitCodeField/0109]$ make
g++ -o mythread mythread.cc -std=c++11
/tmp/ccDJDanb.o: In function `main':
mythread.cc:(.text+0x11d): undefined reference to `pthread_create'
collect2: error: ld returned 1 exit status
make: *** [mythread] Error 1
(base) [yufc@VM-12-12-centos:~/Files/BitCodeField/0109]$
```

链接错误

pthread库不是C库，所以必须在makefile的时候带上-lpthread选项

```
(base) [yufc@VM-12-12-centos:~/Files/BitCodeField/0109]$ ./mythread
thread-0, pid: 9158
thread-1, pid: 9158
thread-0, pid: 9158
thread-2, pid: 9158
thread-1, pid: 9158
thread-0, pid: 9158
thread-2, pid: 9158
thread-3, pid: 9158
thread-1, pid: 9158
thread-0, pid: 9158
thread-2, pid: 9158
thread-3, pid: 9158
thread-1, pid: 9158
thread-4, pid: 9158
thread-0, pid: 9158
thread-2, pid: 9158
main thread, pid: 9158
thread-1, pid: 9158
thread-3, pid: 9158
```

证明了，线程在进程内运行
pid都一样

```
(base) [yufc@VM-12-12-centos:~/Files]$ ps axj | grep mythread
30728  9743  9743 30728 pts/74    9743 Sl+   1001   0:00 ./mythread
 9778  9920  9919  9778 pts/75    9919 S+    1001   0:00 grep --color=auto mythread
(base) [yufc@VM-12-12-centos:~/Files]$
```

只能查到一个进程

```
(base) [yufc@VM-12-12-centos:~/Files]$ ps -aL
   PID   LWP TTY          TIME CMD
  9743  9743 pts/74   00:00:00 mythread
  9743  9744 pts/74   00:00:00 mythread
  9743  9749 pts/74   00:00:00 mythread
  9743  9755 pts/74   00:00:00 mythread
  9743  9760 pts/74   00:00:00 mythread
  9743  9761 pts/74   00:00:00 mythread
 10080 10080 pts/75   00:00:00 ps
(base) [yufc@VM-12-12-centos:~/Files]$
```

实际上可以看
到有6个执行流

PID是一样的
但是
LWP
lighter weight process pid
是不一样的！
LWP和PID是一样的那个，叫主线程

###进程的多个线程共享 同一地址空间,因此Text Segment、Data Segment都是共享的,如果定义一个函数,在各线程中都可以调用,如果定义一个全局变量,在各线程中都可以访问到,除此之外,各线程还共享以下进程资源和环境:

- 文件描述符表
- 每种信号的处理方式(SIG_ IGN、SIG_ DFL或者自定义的信号处理函数)
- 当前工作目录
- 用户id和组id

那么线程私有的东西有哪些呢？

- 线程共享进程数据，但也拥有自己的一部分数据:
  - 线程ID
  - 一组寄存器
  - 栈
  - errno
  - 信号屏蔽字
  - 调度优先级

栈
寄存器
这两个一定要记住，很重要！

3. 进程 vs 线程

调度层面：上下文

为什么线程切换的成本更低？？

地址空间 && 页表不需要切换

CPU内部是有L1~L3 cache　对内存的代码和数据，根据局部性原理，预读CPU内部！！

如果，进程切换 cache就立即失效：新进程过来，只能重新缓存