
阅码场
YOMOCODE

多线程编程(1)

麦当劳喜欢您来，喜欢您再来

扫描关注
Linux阅码场

多线程解决什么问题

- 线程生命周期问题
- 线程如何划分的问题
- 线程如何通信的问题

多线程编程(1)

多线程生命周期

1.1 线程的创建和退出，pthread_join和detach

1.2 exit与exit_group

1.3 主函数返回会发生什么？

1.4 主线程退出会发生什么？

1.5 如果不join线程会发生什么？

1.6 pthread_kill ?

线程创建实例

```
#define THREAD_NUMBER 2
int retval_hello1= 2, retval_hello2 = 3;

void* hello1(void *arg)
{
  char *hello_str = (char *)arg;
  sleep(1);
  printf("%s\n", hello_str);
  pthread_exit(&retval_hello1);
}

void* hello2(void *arg)
{
  char *hello_str = (char *)arg;
  sleep(2);
  printf("%s\n", hello_str);
  pthread_exit(&retval_hello2);
}
```

```
int main(int argc, char *argv[])
{
  int i;   int ret_val;   int *retval_hello[2];
  pthread_t pt[THREAD_NUMBER];      const char
     *arg[THREAD_NUMBER];
  arg[0] = "hello world from thread1";
  arg[1] = "hello world from thread2";
  printf("Begin to create threads...\n");

  ret_val = pthread_create(&pt[0], NULL, hello1, (void *)arg[0]);
  if (ret_val != 0 ) { printf("pthread_create error!\n");   exit(1); }
  ret_val = pthread_create(&pt[1], NULL, hello2, (void *)arg[1]);
  if (ret_val != 0 ) { printf("pthread_create error!\n");  exit(1);   }
  printf("Now, the main thread returns.\n");
  printf("Begin to wait for threads...\n");
  for(i = 0; i < THREAD_NUMBER; i++) {
     ret_val = pthread_join(pt[i], (void **)&retval_hello[i]);
     …
      }
  return 0;
}
```

pthread_create-> clone

- clone()
1. CLONE_VM
2. CLONE_FS
3. CLONE_FILES
4. CLONE_SIGHAND
5. CLONE_THREAD

共享资源，可调度

POSIX标准的要求

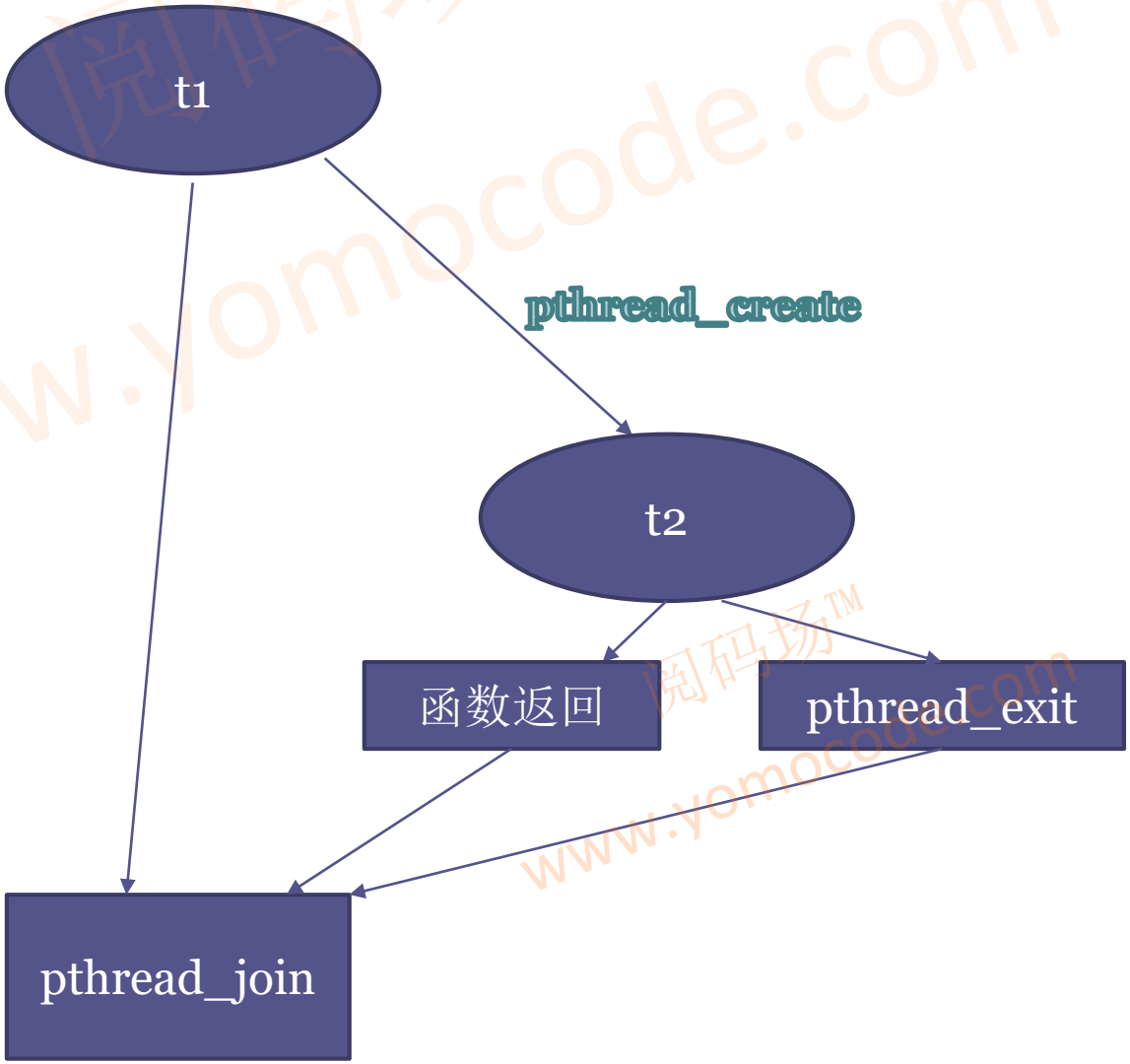
- 1. 查看进程列表的时候, 相关的一组task_struct应当被展现为列表中的一个节点;
- 2. 发送给这个"进程"的信号(对应kill系统调用), 将被对应的这一组task_struct所共享, 并且被其中的任意一个"线程"处理;
- 3. 发送给某个"线程"的信号(对应pthread_kill), 将只被对应的一个task_struct接收, 并且由它自己来处理;
- 4. 当"进程"被停止或继续时(对应SIGSTOP/SIGCONT信号), 对应的这一组task_struct状态将改变;
- 5. 当"进程"收到一个致命信号(比如由于段错误收到SIGSEGV信号), 对应的这一组task_struct将全部退出;

NPTL模型

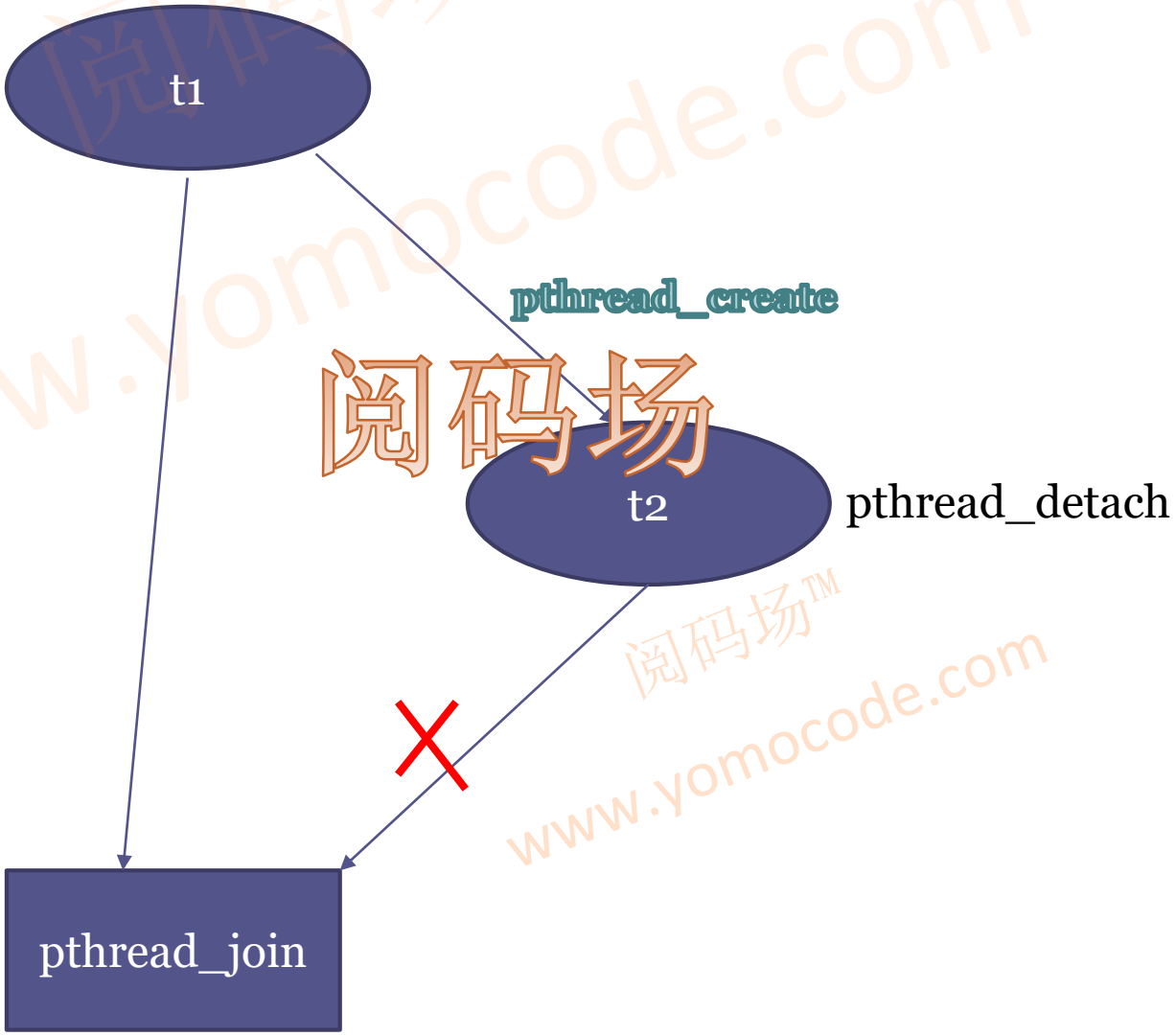
```
barry@barryUbuntu:~$ getconf GNU_LIBPTHREAD_VERSION
NPTL 2.28
```

- ✓ TGID
- ✓ 线程级和进程级信号
- ✓ exit_group

线程生命周期(1)



线程生命周期(2)



线程退出

- 线程函数退出
- pthread_exit
- pthread_cancel

整个进程退出

- main函数返回
- 调用进程级别API：

exit()

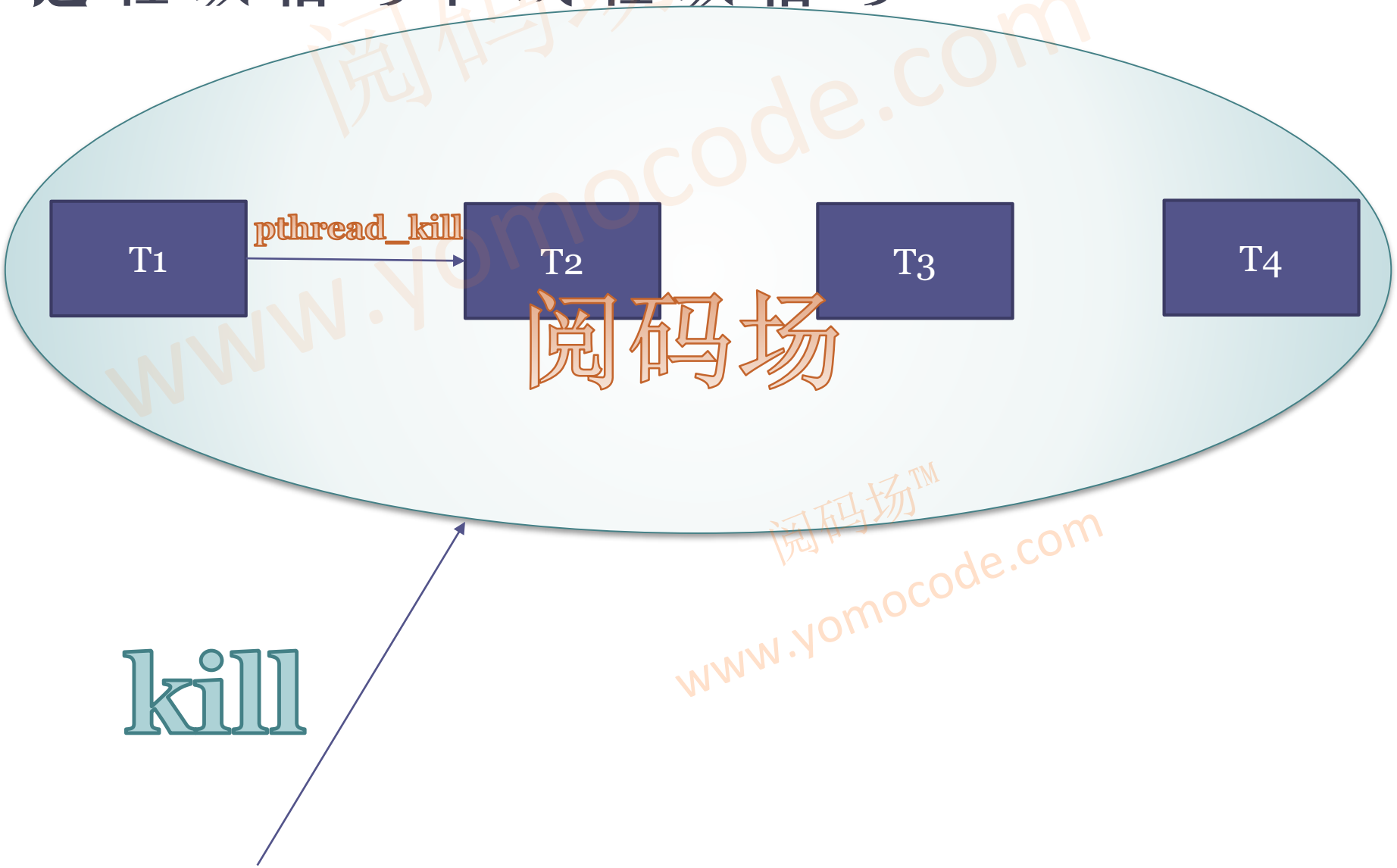
_exit()

- 某一线程做了非法的事情，引起segment fault等错误

线程退出可能引起leak

- 资源的单位是进程；
- 线程退出，可能引起memory leak等leak。
- 不pthread_join可能引起leak

进程级信号和线程级信号



pthread_cancel

- 配个这2个API:

```
void pthread_cleanup_push(void (*routine)(void *),
                          void *arg);
void pthread_cleanup_pop(int execute);
```

- Google Android不支持使用pthread_cancel：

memory leaks

lock no release

other issues

谢谢!

阅码场出品