day12: 线程 exit() _exit() wait waited

deamon: 木马
    创建子进程

```
#include <stdio.h>
#include <stdlib.h>
#include <syslog.h>

int main()
{
    if(0 < fork() ) //第一步，创建子进程，退出父进程
        exit(0);

    setsid(); //第二步，创建新会话（脱离原会话、原进程组、原控制终端）

    chdir("/"); // 第三步， 改变当前目录

    umask(0); //重设文件权限掩码

    int maxfd = getdtablesize();  //关闭文件描述符
    while(maxfd--)
        close(maxfd);

    open

    while(1)
    {
        syslog(LOG_INFO, "hello %d\n", 250); //syslog 是Linux中的系统日志管理服务
        sleep(1);
    }
}
```

fork
kill 父亲

fork
kill 父亲

syslog / openlog

往系统文件打日记 /var/log/syslog

[1] 线程：
    pthread_self() 获得自己的线程号

    clone可以用来 复制进程，比较难使用，使用库POSIX pthread等

[2]

串行效率 比 并发的效率高

但是现在cpu强大，有可能并发的效率更高

[3]

纯线程

纯进程

混合型：线程进程并发

bzero() == memset()

[4] 线程私有数据 ， 使用map

```
#include<pthread.h>

PTHREAD_ONCE_INIT
int phread_once(pthread_once_t *onceptr, vid(*init)(void));

int pthread_key_create(pthread_key_t *keyptr, void(* destructor)(void *value)); //申请表单

int pthread_setspecific(pthread_key_t key, const void *value); //加入记录

void *pthread_getspecific(pthread_key_t key); //提取记录
```

[5] 内存泄露：

浅拷贝：

[6]

静态数组： 单线程效率高

malloc：多线程，效率低，安全

[7] 加宏定义，用来调试

需要的时候，加上，不需要的时候，不加_REENTRANT 宏

线程真正安全是 使用带_r的函数

多线程的时候才加 -DREENTRANT,单线程就不要了

[8] 同步 和 互斥

互斥：解决一个事物，多个事物要配合

同步： 按照顺序，配合完成某个事情

线程互斥锁：mutex

加锁最好定位到某一部分，提高效率

操作系统会监视 ptread_mutex_lock() 函数

[9] 条件变量

pthread_cond_wait()

pthread_cond_broadcast()　[pthread_cond_signal()]

先解锁再睡觉

[10] 线程池

threadpool

homework:

线程池　－－－－　thread_pool

动态：worker（线程，唤醒的时候，剩下的线程抢任务－－互斥）

静态：task，链表（临界资源，需要保护，，，，必须有任务才有线程去执行－－同步）

特殊结构体： 锁在哪，头在哪

create_pool ： 避免线程风暴，太多的线程可能一下把cpu资源用尽，发现某个线程或任务超过10s，线程池能扩展最好（每天都是满负荷工作，就进行扩展），记录日志信息

pool_add_task： 增加任务

条件变量 线程创建 队列（链式）

线程池通常适合以下几种场景：

①、单位时间内处理的任务频繁，且任务时间较短

②、对实时性要求较高。如果接收到任务之后再创建线程，可能无法满足实时性的要求，此时必须使用线程池。

③、必须经常面对高突发性事件。比如Web服务器。如果有足球转播，则服务器将产生巨大冲击，此时使用传统方法，则必须不停的大量创建、销毁线程。

此时采用动态线程池可以避免这种情况的发生。