线程

2023年4月29日 9:30

轻量级进程

正在执行的程序

并发

①每个线程以为自己独占cpu

②cpu调度单位为线程

③原来的进程 单线程进程

内存资源以进程为单位为分配

一个进程有一个地地空间

多个线程

共用

从进程到线程

2023年4月29日 10:07

用户级线程 好处 切换效率高 协程 (go)

坏处 不能利用多核

$\downarrow$

内核级线程

进程和线程本质无区别，

$\downarrow$ 独立地址空间 $\downarrow$ 共享地址空间

Linux中，每个进程/线程有一个 task_struct

线程库

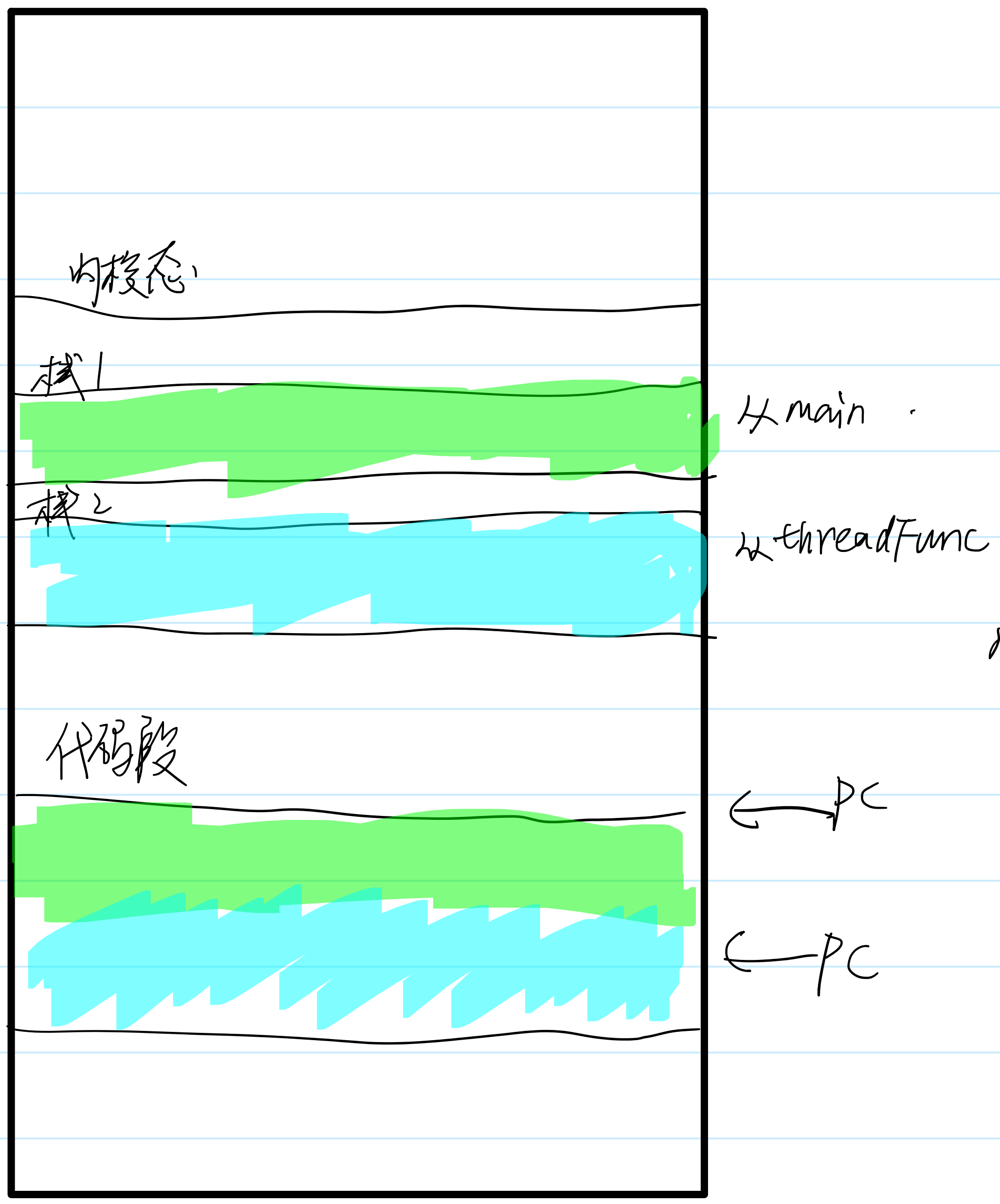
2023年4月29日 10:25

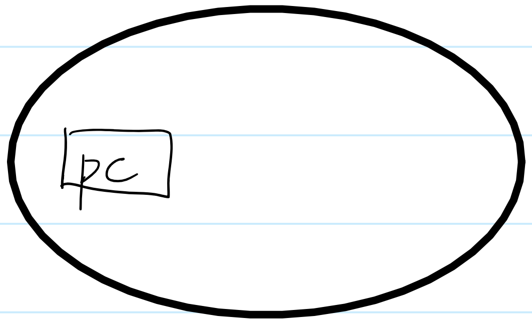
man 7 pthreads

```
Linux implementations of POSIX threads
    Over time, two threading implementations have been provided by the GNU C library on Linux:

    LinuxThreads                IBM
           This is the original Pthreads implementation.  Since glibc 2.4, this implementation is no longer supported.

    NPTL (Native POSIX Threads Library)        RedHat
           This  is  the modern Pthreads implementation.  By comparison with LinuxThreads, NPTL provides closer conformance to the requirements of the
           POSIX.1 specification and better performance when creating large numbers of threads.  NPTL is available since  glibc  2.3.2,  and  requires
           features that are present in the Linux 2.6 kernel.

    Both  of these are so-called 1:1 implementations, meaning that each thread maps to a kernel scheduling entity.  Both threading implementations em-
    ploy the Linux clone(2) system call.  In NPTL, thread synchronization primitives (mutexes, thread joining, and so on) are  implemented  using  the
    Linux futex(2) system call.
```

多线程下的内存模型

2023年4月29日 10:28





每多一个线程，就多一个栈区

创建子线程

2023年4月29日 11:05

线程id. 一个进程内的不同线程，id不同

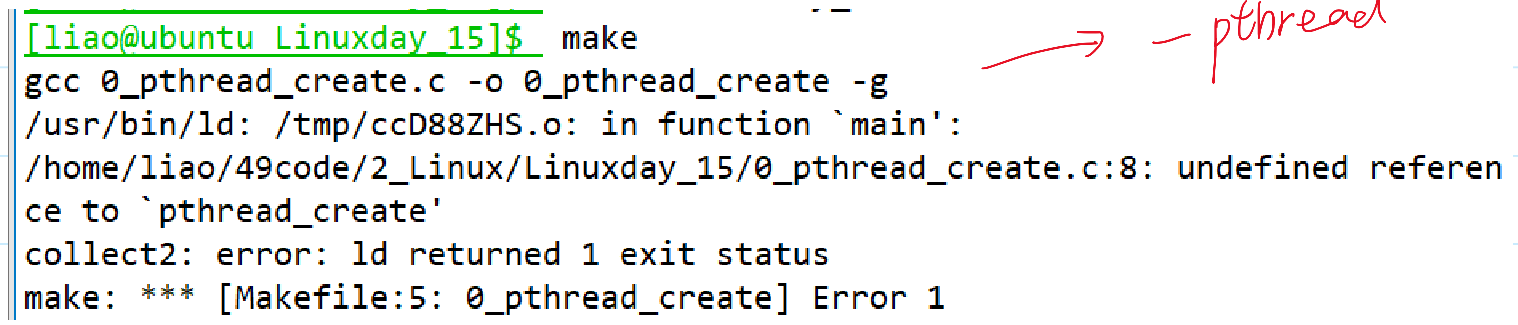
用来存储子线程 tid，

NULL 默认属性

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

start_routine 是线程入口函数 (类似线程的main)

arg 传递给start_routine的参数



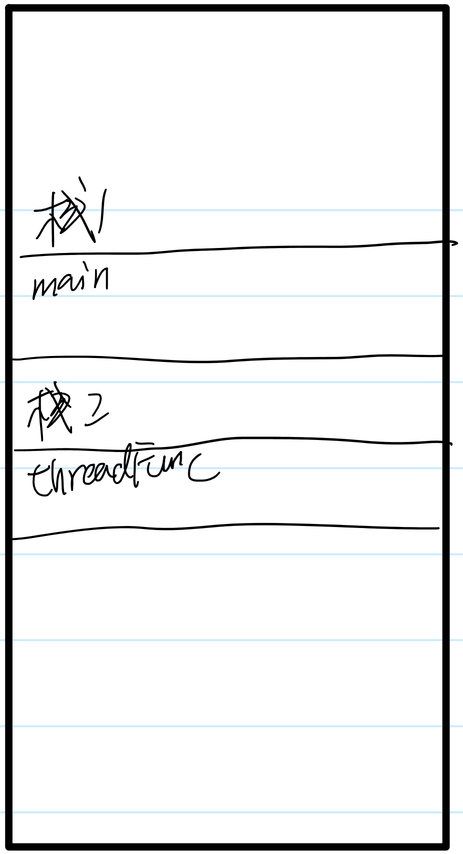
```
[liao@ubuntu Linuxday_15]$  make
gcc 0_pthread_create.c -o 0_pthread_create -g
/usr/bin/ld: /tmp/ccD88ZHS.o: in function `main':
/home/liao/49code/2_Linux/Linuxday_15/0_pthread_create.c:8: undefined referen
ce to `pthread_create'
collect2: error: ld returned 1 exit status
make: *** [Makefile:5: 0_pthread_create] Error 1
```

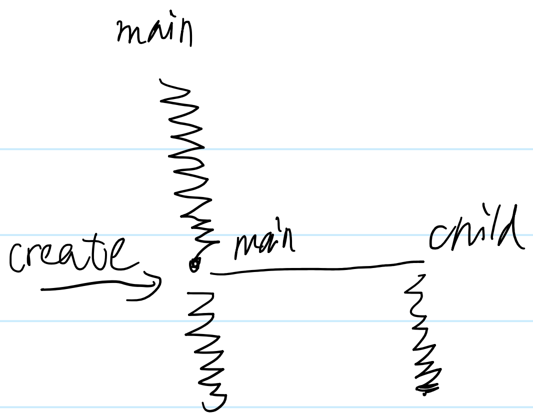
→ -pthread

多线程的运行

2023年4月29日 11:25

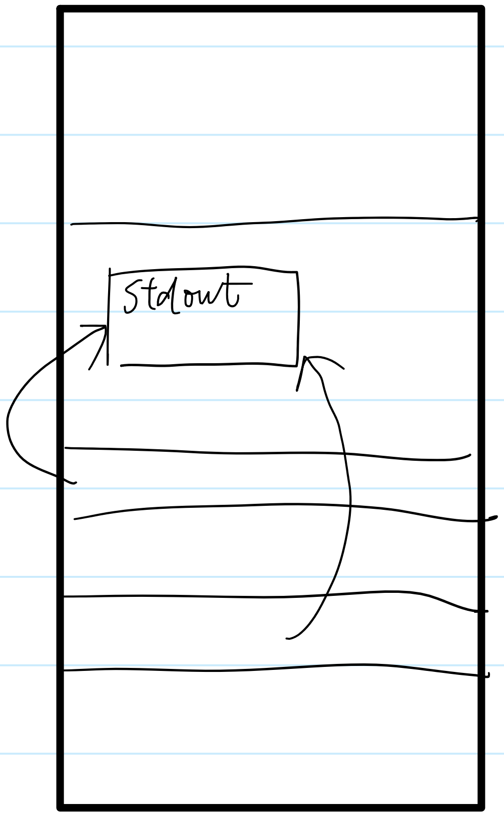
```c
#include <49func.h>
void *threadFunc(void *arg){
    printf("I am child thread\n");
}
int main()
{
    pthread_t tid;//将要用来保存子线程的tid
    pthread_create(&tid,NULL,threadFunc,NULL);
    //创建一个子线程，线程id填入tid，线程属性是默认属性，线程启动函数是threadFunc，传递的参数是NULL
    printf("I am main thread\n");
    sleep(1);
    return 0;
}
```

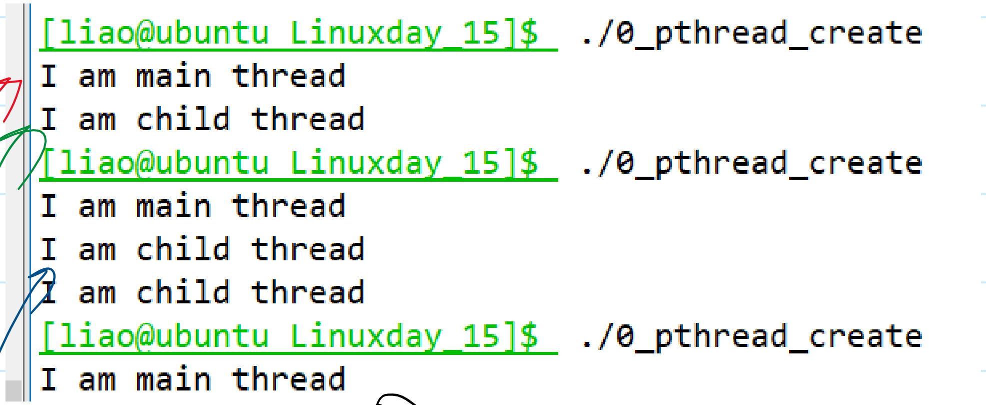




stdout

2023年4月29日 11:31





```
[liao@ubuntu Linuxday_15]$  ./0_pthread_create
I am main thread
I am child thread
[liao@ubuntu Linuxday_15]$  ./0_pthread_create
I am main thread
I am child thread
I am child thread
[liao@ubuntu Linuxday_15]$  ./0_pthread_create
I am main thread
```

子线程

① $\longrightarrow$ "I am child thread" $\longrightarrow$ stdout

② $\longrightarrow$ stdout $\longrightarrow$ 文件对象

③ $\longrightarrow$ 清空stdout

④ $\longrightarrow$

多线程场景下的报错处理

2023年4月29日　　11:44

之前使用库函数/系统调用. perror $\longrightarrow$ errno

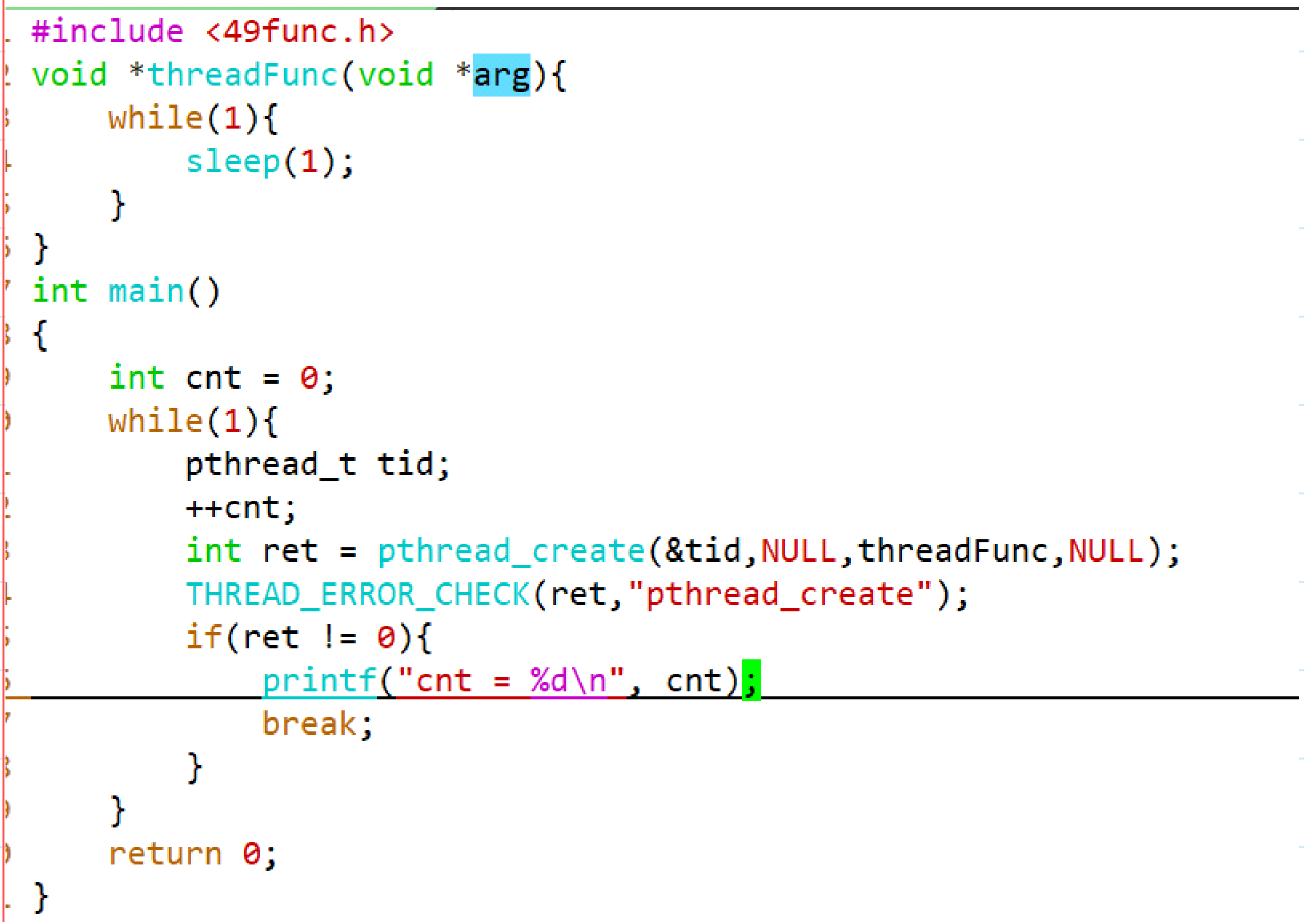
全局变量

```
RETURN VALUE
       On success, pthread_create() returns 0; on error, it returns an error number
```

```
char *strerror(int errnum);
```

```
#define THREAD_ERROR_CHECK(ret,msg) {if(ret != 0){fprintf(stderr,"%s:%s\n",msg,strerror(ret));}}
```

2023年4月29日 11:55



```c
#include <49func.h>
void *threadFunc(void *arg){
    while(1){
        sleep(1);
    }
}
int main()
{
    int cnt = 0;
    while(1){
        pthread_t tid;
        ++cnt;
        int ret = pthread_create(&tid,NULL,threadFunc,NULL);
        THREAD_ERROR_CHECK(ret,"pthread_create");
        if(ret != 0){
            printf("cnt = %d\n", cnt);
            break;
        }
    }
    return 0;
}
```