


第7章 线程

——线程基本操作（1）

主　讲：王小银

线程创建

函数名称	pthread_create
函数功能	创建一个新的线程
头文件	#include <pthread.h>
函数原型	int pthread_create(pthread_t *thread, const pthread_attr_t *attr,void *(*start_routine) (void *), void *arg);
参数	thread：指向线程标识符的指针，用来存储线程标识符； attr：设置线程属性，主要设置与栈相关的属性。一般情况下，该参数设置为NULL，新的线程将使用系统默认的属性； start_routine：是线程运行函数的起始地址，即在此线程中运行哪段代码； arg：运行函数的参数地址。
返回值	如果线程创建成功，返回0。如果创建失败，则返回出错编号。 创建成功时，由thread 指向的内存单元被设置为新创建线程的线程ID。

获取线程ID

函数名称	pthread_self
函数功能	获取线程ID
头文件	#include <pthread.h>
函数原型	pthread_t pthread_self(void);
参数	无
返回值	返回调用线程的ID。

示例程序

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <pthread.h>
 void printids(const char *s)
{
    pid_t   pid;
    pthread_t   tid;

    pid = getpid();
    tid = pthread_self();
   printf("%s pid is %u,tid is %lu (0x%lx)\n", s, pid, tid,tid);
}
```

```
void* thr_fn(void *arg)
{
    printids("new thread:");
    return((void*)0);
}
 int main()
{
    int     ret;
    pthread_t ntid;

    ret = pthread_create(&ntid, NULL, thr_fn, NULL);
    if(ret!=0)
    {  printf("can't create thread\n");
       exit(1);
    }
    printids("main thread:");
    sleep(2);
    exit(0);
}
```

示例程序8.2

```
#define BACKLOG 1
#define MAXRECVLEN 1024


int main(int argc, char *argv[])
{ char buf[MAXRECVLEN];
   int listenfd, connectfd;        //套接字
   struct sockaddr_in server;   //服务端地址信息
   struct sockaddr_in client;   // 客户端地址信息
   socklen_t addrlen;
    char *ip = argv[1];
   int port = atoi(argv[2]);
  if (argc != 3)
 {  printf("argument error, please input ip and port\n");
    exit(1);
}
```

示例程序8.2


```
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
void thread(void)
{       int i;
        for(i=0;i<10;i++)
                printf("This is a pthread.TID: %lu\n",pthread_self());
}
int main(void)
{       pthread_t id;
        int i,ret;
        ret=pthread_create(&id,NULL,(void *)thread,NULL);
        if(ret!=0)
{
                printf("Create pthread error!\n");
```

函数名称	pthread_exit
函数功能	结束调用线程
头文件	#include <pthread.h>
函数原型	void pthread_exit(void *retval);
参数	retval : void类型的指针，指向退出信息。
返回值	无。

线程退出/等待

函数名称	pthread_join
函数功能	等待另一个线程结束
头文件	#include <pthread.h>
函数原型	int pthread_join(pthread_t thread, void **retval);
参数	thread: 要等待线程的pid。 retval: 用来存储被等待线程的返回值。
返回值	0: 成功; 错误号: 失败。

函数名称	pthread_cleanup_push
函数功能	注册清理函数
头文件	#include <pthread.h>
函数原型	void pthread_cleanup_push(void (*routine)(void *);
参数	routine:　　要注册的函数
返回值	无。

线程终止

函数名称	pthread_cancel
函数功能	取消一个线程
头文件	#include <pthread.h>
函数原型	int pthread_cancel(pthread_t thread);
参数	thread: 要取消的线程的pid。
返回值	0: 成功; 错误号: 失败。

一个线程能够被取消并终止执行需满足的条件:

- 线程是否可以被取消
- 线程是否处于可取消点

线程终止

函数名称	pthread_setcancelstate
函数功能	设置可取消状态
头文件	#include <pthread.h>
函数原型	int pthread_setcancelstate(int state, int *oldstate);
参数	state：　　设置的取消状态； oldstate：　保存上一次取消状态的指针。
返回值	0：　　成功； 错误号：　失败。

参数state的合法值：

- PTHREAD_CANCEL_DISABLE
- PTHREAD_CANCEL_ENABLE

函数名称	pthread_setcanceltype
函数功能	设置取消类型
头文件	#include <pthread.h>
函数原型	int pthread_setcanceltype(int type, int *oldtype);
参数	type：　要设置的取消类型； oldtype：　指针类型，保存上一次取消的类型。
返回值	成0：　成功； 错误号：　失败。

参数type的合法值：

- PTHREAD_CANCEL_ASYNCHRONOUS
- PTHREAD_CANCEL_DEFERRED

函数名称	pthread_testcancel
函数功能	检查本线程是否处于Cancled状态，如果是，则进行取消动作，否则直接返回。
头文件	#include <pthread.h>
函数原型	void pthread_testcancel(void);
参数	无。
返回值	无。

谢谢大家！