线程

条件变量

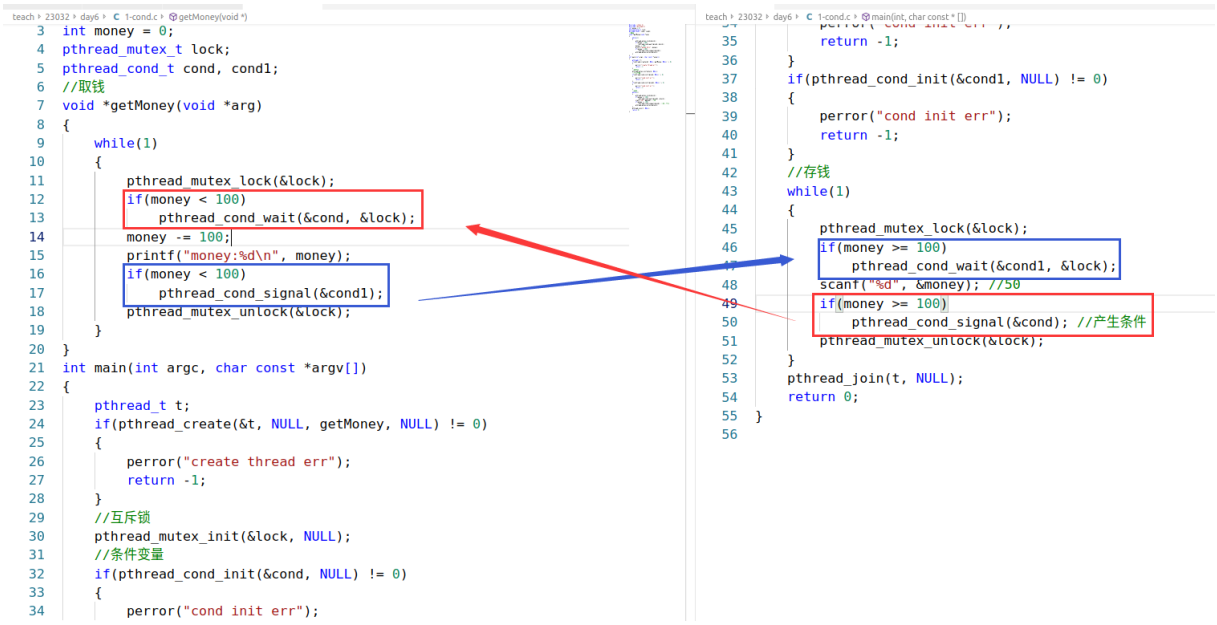
和互斥锁搭配使用实现同步机制

```
int pthread_cond_init(pthread_cond_t *restrict cond,const
pthread_condattr_t *restrict attr);
功能：初始化条件变量
参数：cond：是一个指向结构 pthread_cond_t 的指针
    restrict attr：是一个指向结构 pthread_condattr_t 的指针，一般设为
NULL
返回值：成功：0 失败：非 0
int pthread_cond_wait(pthread_cond_t *restrict cond,
pthread_mutex_t *restrict mutex);
功能：等待条件的产生
参数：restrict cond：要等待的条件
     restrict mutex：对应的锁
返回值：成功：0，失败：不为 0
注：当没有条件产生时函数会阻塞，同时会将锁解开；如果等待到条件产生，函数会结束
阻塞同时进行上锁。
int pthread_cond_signal(pthread_cond_t *cond);
功能：产生条件变量
参数：cond：条件变量值
返回值：成功：0，失败：非 0
注：必须等待 pthread_cond_wait 函数先执行，再产生条件才可以
int pthread_cond_destroy(pthread_cond_t *cond);
功能：将条件变量销毁
参数：cond：条件变量值
返回值：成功：0， 失败：非 0
```

pthread_mtex_init(&lock, NULL);

案例：存钱和取钱的例子

主线程循环存钱，子线程循环取钱，每次取 100 直到余额为 0，再进行存钱；



```c
int money = 0;
pthread_mutex_t lock;
pthread_cond_t cond, cond1;
//取钱
void *getMoney(void *arg)
{
    while(1)
    {
        pthread_mutex_lock(&lock);
        if(money < 100)
            pthread_cond_wait(&cond, &lock);
        money -= 100;
        printf("money:%d\n", money);
        if(money < 100)
            pthread_cond_signal(&cond1);
        pthread_mutex_unlock(&lock);
    }
}
int main(int argc, char const *argv[])
{
    pthread_t t;
    if(pthread_create(&t, NULL, getMoney, NULL) != 0)
    {
        perror("create thread err");
        return -1;
    }
    //互斥锁
    pthread_mutex_init(&lock, NULL);
    //条件变量
    if(pthread_cond_init(&cond, NULL) != 0)
    {
        perror("cond init err");
        return -1;
    }
    if(pthread_cond_init(&cond1, NULL) != 0)
    {
        perror("cond init err");
        return -1;
    }
    //存钱
    while(1)
    {
        pthread_mutex_lock(&lock);
        if(money >= 100)
            pthread_cond_wait(&cond1, &lock);
        scanf("%d", &money); //50
        if(money >= 100)
            pthread_cond_signal(&cond); //产生条件
        pthread_mutex_unlock(&lock);
    }
    pthread_join(t, NULL);
    return 0;
}
```

进程间通信

传统的进程间通信方式：

无名管道、有名管道、信号

system V IPC 对象：

共享内存、消息队列、信号灯集

BSD:

套接字(socket)

1. 有名管道

1.1 特点

a. 有名管道可以使互不相关的两个进程互相通信。

b. 有名管道可以通过路径名来指出，并且在文件系统中可见，但内容存放在内存中。

c. 进程通过文件 IO 来操作有名管道

d. 有名管道遵循先进先出规则

e. 不支持如 lseek() 操作

1.2 函数接口

```
int mkfifo(const char *filename,mode_t mode);
功能：创健有名管道
参数：filename：有名管道文件名
        mode：权限
返回值：成功：0
        失败：-1，并设置 errno 号
注意对错误的处理方式：
如果错误是 file exist 时，注意加判断，如：if(errno == EEXIST)
```

1.3 注意事项

a. 只写方式，写阻塞，一直到另一个进程把读打开

b. 只读方式，读阻塞，一直到另一个进程把写打开

c. 可读可写，如果管道中没有数据，读阻塞

练习：实现两个不相关进程间通信。

read.c ：从终端读取数据

write.c : 向终端输出数据

当输入 quit 时结束。

1.4 有名管道和无名管道区别

	无名管道	有名管道
特点	只能在亲缘关系进程间使用 半双工通信方式 有固定的读端和写端，fd[0]：读，fd[1]:写端 通过文件 IO 进行操作 步骤：创建管道、读写操作	不相关的任意进程间使用 在路径中有管道文件，实际数据存在内核空间 通过文件 IO 进行操作 步骤：创建管道、打开管道、读写操作
函数	pipe	mkfifo
读写特性	当管道中没有数据，读阻塞 当写满管道时，写阻塞	

2. 信号

2.1 概念

1）信号是在软件层次上对中断机制的一种模拟，是一种 异步通信方式

2）信号可以直接进行用户空间进程和内核进程之间的交互，内核进程也可以利用它来通知用户空间进程发生了哪些系统事件。

3）如果该进程当前并未处于执行态，则该信号就由内核保存起来，直到该进程恢复执行再传递给它；如果一个信号被进程设置为阻塞，则该信号的传递被延迟，直到其阻塞被取消时才被传递给进程。

2.2. 信号的响应方式

1）忽略信号：对信号不做任何处理，但是有两个信号不能忽略：即 SIGKILL 及 SIGSTOP

2）捕捉信号：定义信号处理函数，当信号发生时，执行相应的处理函数。

3）执行缺省操作：Linux 对每种信号都规定了默认操作

2.3. 信号种类

SIGKILL：结束进程，不能被忽略不能被捕捉

SIGSTOP：结束进程，不能被忽略不能被捕捉

SIGCHLD：子进程状态改变时给父进程发的信号,不会结束进程

SIGINT：结束进程，对应快捷方式 ctrl+c

SIGTSTP：暂停信号，对应快捷方式 ctrl+z

SIGQUIT：退出信号，对应快捷方式 ctrl+\

SIGALRM：闹钟信号，alarm 函数设置定时，当到设定的时间时，内核会向进程发送此信号结束进程。

SIGTERM：结束终端进程，kill 使用时不加数字默认是此信号

2.4 函数接口

```
int kill(pid_t pid, int sig);
功能：信号发送
参数：pid：指定进程
   sig：要发送的信号
返回值：成功 0
        失败 -1
int raise(int sig);
功能：进程向自己发送信号
参数：sig：信号
返回值：成功 0
        失败 -1
int pause(void);
功能：用于将调用进程挂起，直到收到信号为止。
#include <signal.h>
 typedef void (*sighandler_t)(int);
sighandler_t signal(int signum, sighandler_t handler);
功能：信号处理函数
参数：signum：要处理的信号
      handler：信号处理方式
             SIG_IGN：忽略信号
             SIG_DFL：执行默认操作
            handler：捕捉信号  void handler(int sig){} //函数名可以自
定义
返回值：成功：设置之前的信号处理方式
      失败：-1
```

```
typedef void (*sighandler_t)(int);   //typedef  unsigned int  INT;
typedef  void (*)(int)   sighandler_t;
sighandler_t signal(int signum, void (*handler)(int) );

void handler(int sig)
{
    if(sig == SIGINT)
        printf("xxx\n");
```

```
    else if(sig == SIGQUIT)
}
signal(SIGINT, handler);
signal(SIGQUIT, handler);
```

作业：

1. 两个进程实现 cp 功能

   ./r srcfile

   ./w newfile

2. 用信号的知识实现司机和售票员问题。

   1）售票员捕捉 SIGINT（代表开车）信号，向司机发送 SIGUSR1 信号，司机打印（let's gogogo）

   2）售票员捕捉 SIGQUIT（代表停车）信号，向司机发送 SIGUSR2 信号，司机打印（stop the bus）

   3）司机捕捉 SIGTSTP（代表到达终点站）信号，向售票员发送 SIGUSR1 信号，售票员打印（please get off the bus）

   4）司机等待售票员下车，之后司机再下车。

   司机：父进程

      捕捉信号：SIGUSR1 SIGUSR2 SIGTSTP

      忽略信号：SIGINT SIGQUIT

   售票员：子进程

      捕捉信号：SIGINT SIGQUIT SIGUSR1

      忽略信号：SIGTSTP

   ./a.out