第十二章 线程控制

习题：

12-1、老话重提，之前应该被问到过；

```
tongue aupe # gcc 12-1.c -o 12-1 -lpthread
tongue aupe # ./12-1
thread started...
parent about to fork...
preparing locks...
parent unlocking locks...
parent returned from fork
child unlocking locks...
child returned from fork
tongue aupe # ./12-1  > out.msg
tongue aupe # cat out.msg
thread started...
parent about to fork...
preparing locks...
parent unlocking locks...
parent returned from fork
thread started...
parent about to fork...
preparing locks...
child unlocking locks...
child returned from fork
```

与上面比较多出来的部分

程序其它部分与书中程序清单12-7一样；

```
int main(int argc, const char *argv[]) {
    int err;
    pid_t pid;
    pthread_t tid;
    setvbuf( stdout, NULL, _IONBF, 0);
    if (0 != (err = pthread_atfork(prepare, parent, child))) {
        printf("can't install fork handlers\n"); _exit(-1);
    }
```

加入该行

```
tongue aupe # gcc 12-1.c -o 12-1 -lpthread
tongue aupe # ./12-1
thread started...
parent about to fork...
preparing locks... 郁闷解除呼？
parent unlocking locks...
parent returned from fork
child unlocking locks...
child returned from fork
tongue aupe # ./12-1  > out.msg
tongue aupe # cat out.msg
thread started...
parent about to fork...
preparing locks...
parent unlocking locks...
parent returned from fork
child unlocking locks...
child returned from fork
```

12-2、这题应该是指通过锁，把原有的putenv封装一下，变成putenv_r，信号方面由pthread_sigmask细化每个线程实现，若并不需要控制每个线程的信号，用sigprocmask在主线程屏蔽下就ok了；

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <pthread.h>

struct thr_args {
    char env_str[256];
};

struct thr_args thr_argv_01;
struct thr_args thr_argv_02;
sigset_t t_mask_01;
sigset_t t_mask_02;

pthread_mutex_t putenv_lock = PTHREAD_MUTEX_INITIALIZER;

int putenv_r(char *env_str) {
    pthread_mutex_lock( &putenv_lock);
    if(0 != putenv(env_str)) {
        pthread_mutex_unlock( &putenv_lock);
        return -1;
    }
    pthread_mutex_unlock( &putenv_lock);
    return 0;
}
```

```c
void * thr_fn_01(void *arg) {
    int signo;
    sigdelset( &t_mask_01, SIGUSR2);
    pthread_sigmask( SIG_BLOCK, &t_mask_01, NULL);
    putenv_r(((struct thr_args *) arg)->env_str);
    while(1) {
        sigwait( &t_mask_01, &signo);
        if (strcmp( "ENV_01=thr_argv_01", ((struct thr_args *) arg)->env_str)) {
            printf("%c[7;32mThread ID %ld signo: %c[0m%d\n", 27, pthread_self(), 27, signo);
        } else {
            printf("%c[7;33mThread ID %ld signo: %c[0m%d\n", 27, pthread_self(), 27, signo);
        }
    }
    return (void *) 0;
}

void * thr_fn_02(void *arg) {
    int signo;
    sigdelset( &t_mask_02, SIGUSR1);
    pthread_sigmask( SIG_BLOCK, &t_mask_02, NULL);
    while(1) {
        sigwait( &t_mask_02, &signo);
        printf("%c[7;31mThread ID %ld signo: %c[0m%d\n", 27, pthread_self(), 27, signo);
    }
    return (void *) 0;
}
```

```c
int main(int argc, const char *argv[]) {
    int i = 0;
    pid_t pid = getpid();
    pthread_t tid01, tid02, tid03;
    sigfillset( &t_mask_01);
    sigfillset( &t_mask_02);
    // 方便crtl+c退出；
    sigdelset( &t_mask_01, SIGINT);
    sigprocmask( SIG_BLOCK, &t_mask_01, NULL);
    strcpy( thr_argv_01.env_str, "ENV_01=thr_argv_01");
    strcpy( thr_argv_02.env_str, "ENV_02=thr_argv_02");
    pthread_create( &tid01, NULL, thr_fn_01, &thr_argv_01);
    pthread_create( &tid02, NULL, thr_fn_01, &thr_argv_02);
    pthread_create( &tid03, NULL, thr_fn_02, NULL);
    sleep(1);
    for (i=0; i<atoi(argv[1]); i++) {
        kill( pid, SIGUSR1);
        kill( pid, SIGUSR2);
        kill( pid, SIGQUIT);
    }
    printf( "%s\n", getenv("ENV_01"));
    printf( "%s\n", getenv("ENV_02"));
    pause();
    return 0;
}
```

/opt/drill_ground/aupe/12-2.c [FORMAT=unix:utf-8] [TYPE=C]

```
tongue aupe # ./12-2 100
Thread ID 139687802107648 signo: 3
Thread ID 139687802107648 signo: 10
Thread ID 139687785322240 signo: 3
thr_argv_01
thr_argv_02
Thread ID 139687785322240 signo: 12
Thread ID 139687793714944 signo: 3
^C
tongue aupe #
```

这里演示只用100，要更好的效果可把数值写更大

12-3、不可以！在时间点T1恢复屏蔽字，时间点T2程序返回，在T1和T2之间有时间窗，这个窗口间发生的信号会阻断线程并影响到整个进程；

12-4、linux下实现正常，对BSD不感兴趣；

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
#define ARG_MAX BUFSIZ

static pthread_key_t key;
static pthread_once_t init_done = PTHREAD_ONCE_INIT;
pthread_mutex_t env_mutex = PTHREAD_MUTEX_INITIALIZER;

extern char **environ;

struct thr_args {
    int cycle;
    char identified;
};
struct thr_args thr_args_01;
struct thr_args thr_args_02;

static void thread_init() {
    pthread_key_create( &key, free);
}
```

```
char * getenv(const char *name) {
    int i, len;
    char *envbuf;
    pthread_once( &init_done, thread_init);
    pthread_mutex_lock( &env_mutex);
    envbuf = (char *) pthread_getspecific(key);
    if (NULL == envbuf) {
        envbuf = malloc(ARG_MAX);
        if (NULL == envbuf) {
            pthread_mutex_unlock( &env_mutex);
            return(NULL);
        }
        pthread_setspecific( key, envbuf);
    }
    len = strlen(name);
    for ( i=0; NULL != environ[i]; i++) {
        if (0 == (strncmp( name, environ[i], len)) && ('=' == environ[i][len])) {
            strcpy( envbuf, &environ[i][len+1]);
            pthread_mutex_unlock( &env_mutex);
            return( envbuf);
        }
    }
    pthread_mutex_unlock( &env_mutex);
    return( NULL);
}
```

```
void * thr_fn(void *arg) {
    int i;
    for (i=0; i < ((struct thr_args *) arg)->cycle; i++) {
        if ('A' == ((struct thr_args *) arg)->identified) {
            printf("%c[7;31mThread id %ld envbuf: %c[0m%s\n", 27, pthread_self(), 27, getenv("SHELL"));
        } else {
            printf("%c[7;32mThread id %ld envbuf: %c[0m%s\n", 27, pthread_self(), 27, getenv("SHELL"));
        }
    }
    return (void *) 0;
}

int main(int argc, const char *argv[]) {
    pthread_t tid;
    thr_args_01.cycle = atoi(argv[1]);
    thr_args_01.identified = 'A';
    thr_args_02.cycle = atoi(argv[1]);
    thr_args_02.identified = 'B';
    pthread_create( &tid, NULL, thr_fn, &thr_args_01);
    pthread_create( &tid, NULL, thr_fn, &thr_args_02);
    pause();
    return 0;
}
/opt/drill_ground/aupe/12-4.c [FORMAT=unix:utf-8] [TYPE=C] [COL=001] [ROW=072/72(100%)]
```

```
tongue aupe # gcc -g 12-4.c -o 12-4 -lpthread
tongue aupe # ./12-4 10
Thread id 139795269392128 envbuf: /bin/bash
Thread id 139795269392128 envbuf: /bin/bash
Thread id 139795269392128 envbuf: /bin/bash
Thread id 139795277784832 envbuf: /bin/bash
Thread id 139795277784832 envbuf: /bin/bash
Thread id 139795269392128 envbuf: /bin/bash
Thread id 139795269392128 envbuf: /bin/bash
Thread id 139795269392128 envbuf: /bin/bash
Thread id 139795269392128 envbuf: /bin/bash
Thread id 139795269392128 envbuf: /bin/bash
Thread id 139795269392128 envbuf: /bin/bash
Thread id 139795269392128 envbuf: /bin/bash
Thread id 139795277784832 envbuf: /bin/bash
Thread id 139795277784832 envbuf: /bin/bash
Thread id 139795277784832 envbuf: /bin/bash
Thread id 139795277784832 envbuf: /bin/bash
Thread id 139795277784832 envbuf: /bin/bash
Thread id 139795277784832 envbuf: /bin/bash
Thread id 139795277784832 envbuf: /bin/bash
Thread id 139795277784832 envbuf: /bin/bash
^C
```

12-5、从下面几个方面讲，是有这样的需求的：

1、新线程是在进程内部协助主线程运行的，当一个进程要处理多个独立事务时，就需要线程的加入。

2、多核处理器的出现，使得多进程能更充分发挥机器性能，提升业务处理效率。

3、多线程不能代替多进程，多进程可以更充分的发挥机器新能，所以类似业务都有这样的需求。

比如，一个进程既要被动接收客户端来的请求，又要主动发送信息到其它服务器，这就需要多线程来实现。所在机器有多个cpu核心，要充分利用它们提高业务效率，通过多进程最为直接。这就是一个既需多线程，又有多进程的例子。当然进程由亲缘和非亲缘关系之分，需不需要由一个主进程来fork多个子进程是另一事；

12-6、骨架及思路如下；

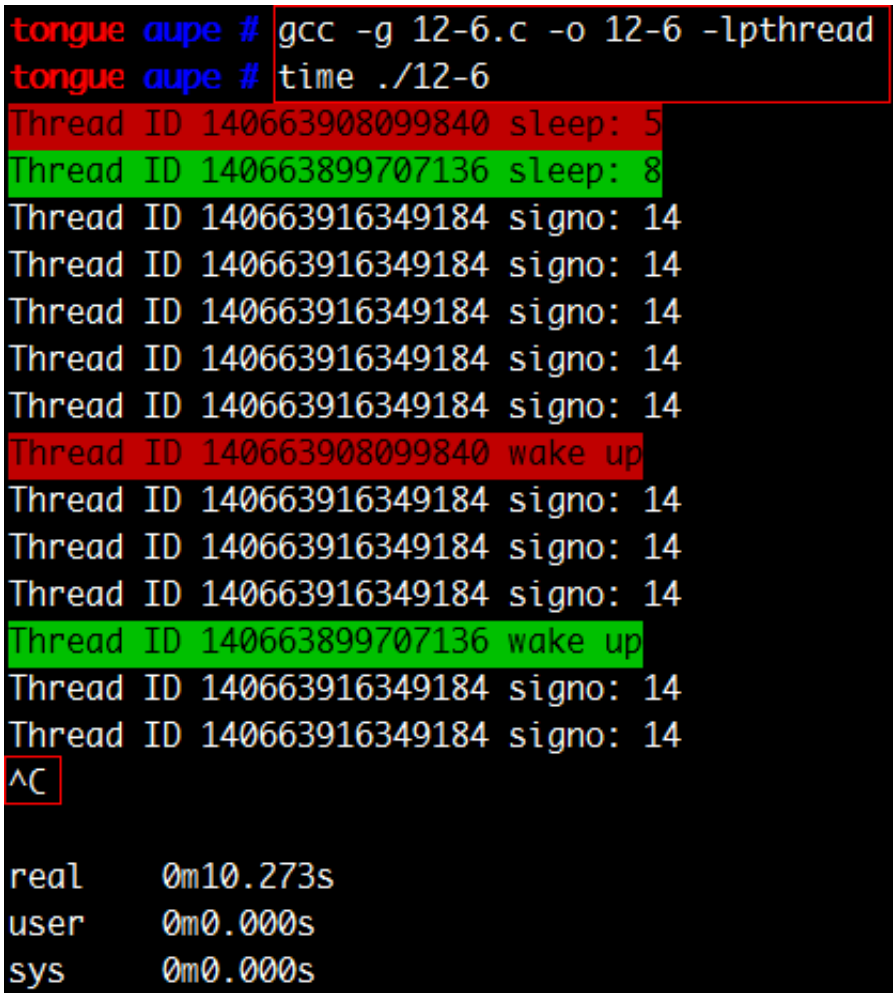
```c
#include <stdio.h>
#include <string.h>
#include <signal.h>
#include <pthread.h>

#define MAX_ALARM 1027

struct thr_args {
    char identified;
};
struct thr_args thr_args_01;
struct thr_args thr_args_02;

struct thr_alarm {
    pthread_t tid;
    unsigned int nsec;
};

static pthread_t tid01, tid02, tid_dmr;
struct thr_alarm arr_thr_alarm[MAX_ALARM];
static sigset_t mask;

static void sig_usr1(int signo) {
}

static void sig_alrm(int signo) {
        printf("Thread ID %ld signo: %d\n", pthread_self(), signo);
        pthread_kill( tid_dmr, SIGUSR1);
}
```

```c
unsigned int sleep(unsigned int arg_nsecs) {
    int signo;
    unsigned int thr_hashno, nsecs = arg_nsecs;
    thr_hashno = pthread_self() % MAX_ALARM;
    bzero( &arr_thr_alarm[thr_hashno], sizeof(struct thr_alarm));
    arr_thr_alarm[thr_hashno].tid = pthread_self();
    arr_thr_alarm[thr_hashno].nsec = nsecs;
    sigsuspend( &mask);
    return( arr_thr_alarm[thr_hashno].nsec);
}

void * thr_fn(void *arg) {
    if ('A' == ((struct thr_args *) arg)->identified) {
        printf("%c[7;31mThread ID %ld sleep: 5%c[0m\n", 27, pthread_self(), 27);
        sleep(5);
        printf("%c[7;31mThread ID %ld wake up%c[0m\n", 27, pthread_self(), 27);
    } else {
        printf("%c[7;32mThread ID %ld sleep: 8%c[0m\n", 27, pthread_self(), 27);
        sleep(8);
        printf("%c[7;32mThread ID %ld wake up%c[0m\n", 27, pthread_self(), 27);
    }
    pause();
    return (void *) 0;
}
```

```c
void * thr_drummer(void *arg) {
    int i, signo;
    while(1) {
        alarm(1);
        sigsuspend( &mask);
        for (i=0; i<MAX_ALARM; i++) {
            arr_thr_alarm[i].nsec--;
            if (0 == arr_thr_alarm[i].nsec) {
                pthread_kill( arr_thr_alarm[i].tid, SIGUSR1);
            }
        }
    }
    pthread_exit((void *) 0);
}

void init_arr_alarm() {
    int i;
    for (i=0; i<MAX_ALARM; i++) {
        arr_thr_alarm[i].nsec = 9999999;
    }
}
```

```c
int main(int argc, const char *argv[]) {
    init_arr_alarm();
    sigfillset( &mask);
    // 方便crtl+c退出：
    sigdelset( &mask, SIGINT);
    sigdelset( &mask, SIGALRM);
    sigdelset( &mask, SIGUSR1);
    sigprocmask( SIG_BLOCK, &mask, NULL);
    pthread_sigmask( SIG_BLOCK, &mask, NULL);
    signal( SIGALRM, sig_alrm);
    // signal( SIGUSR1, SIG_IGN);
    signal( SIGUSR1, sig_usr1);
    thr_args_01.identified = 'A';
    thr_args_02.identified = 'B';
    pthread_create( &tid01, NULL, thr_fn, &thr_args_01);
    pthread_create( &tid02, NULL, thr_fn, &thr_args_02);
    pthread_create( &tid_dmr, NULL, thr_drummer, NULL);
    while(1){
        pause();
    }
    return 0;
}
```

/opt/drill_ground/aupe/12-6.c [FORMAT=unix:utf-8] [TYPE=C



12-7、按理是可以的，除非使用pthread_cond_destroy的平台实现它不到位。