第十二章 线程控制

1. 线程限制

- Single UNIX Specification定义了与线程操作有关的一些限制
  - 可以通过sysconf函数进行查询
  - 为了增强应用程序在不同操作系统实现之间的可移植性

限制名称	描述	*name* 参数
PTHREAD_DESTRUCTOR_ITERATIONS	线程退出时操作系统实现试图销毁线程特定数据的最大次数（见 12.6 节）	_SC_THREAD_DESTRUCTOR_ITERATIONS
PTHREAD_KEYS_MAX	进程可以创建的键的最大数目（见 12.6 节）	_SC_THREAD_KEYS_MAX
PTHREAD_STACK_MIN	一个线程的栈可用的最小字节数（见 12.3 节）	_SC_THREAD_STACK_MIN
PTHREAD_THREADS_MAX	进程可以创建的最大线程数（见 12.3 节）	_SC_THREAD_THREADS_MAX

1. 线程限制

- Single UNIX Specification定义了与线程操作有关的一些限制
  - 可以通过sysconf函数进行查询
  - 为了增强应用程序在不同操作系统实现之间的可移植性

限制名称	FreeBSD 8.0	Linux 3.2.0	Mac OS X 10.6.8	Solaris 10
PTHREAD_DESTRUCTOR_ITERATIONS	4	4	4	没有确定的限制
PTHREAD_KEYS_MAX	256	1 024	512	没有确定的限制
PTHREAD_STACK_MIN	2 048	16 384	8 192	8 192
PTHREAD_THREADS_MAX	没有确定的限制	没有确定的限制	没有确定的限制	没有确定的限制

2. 线程属性

- pthread接口允许设置每个对象关联的不同属性来细调线程和同步对象的行为
  - 对象与它自己类型的属性对象进行关联（线程与线程属性关联，互斥量与互斥量属性关联等）
  - 有一个初始化函数，把属性设置为默认值
  - 有一个销毁属性对象的函数，释放与属性对象关联的资源
  - 每个属性有一个从属性对象中获取属性值的函数
  - 每个属性都有一个设置属性值的函数

2. 线程属性

- pthread接口允许设置每个对象关联的不同属性来细调线程和同步对象的行为

```
#include<pthread.h>
int pthread_attr_init(pthread_attr_t * attr);
int pthread_attr_destroy(pthread_attr_t *attr);
```

2. 线程属性

- POSIX.1定义的线程属性

名称	描述	FreeBSD 8.0	Linux 3.2.0	Mac OS X 10.6.8	Solaris 10
detachstate	线程的分离状态属性	•	•	•	•
guardsize	线程栈末尾的警戒缓冲区大小（字节数）	•	•	•	•
stackaddr	线程栈的最低地址	•	•	•	•
stacksize	线程栈的最小长度（字节数）	•	•	•	•

```
#include<pthread.h>
int pthread_attr_getdetachstate(const pthread_attr_t *restrict attr,
                                int *detachstate);
int pthread_attr_setdetachstate(pthread_attr_t *attr,
                                int *detachstate);
```

2. 线程属性

- POSIX.1定义的线程属性

```
int
makethread(void *(*fn)(void *), void *arg)
{
    int             err;
    pthread_t       tid;
    pthread_attr_t  attr;

    err = pthread_attr_init(&attr);
    if (err != 0)
        return(err);
    err = pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);
    if (err == 0)
        err = pthread_create(&tid, &attr, fn, arg);
    pthread_attr_destroy(&attr);
    return(err);
}
```

2. 线程属性

- POSIX.1定义的线程属性

名称	描述	FreeBSD 8.0	Linux 3.2.0	Mac OS X 10.6.8	Solaris 10
detachstate	线程的分离状态属性	•	•	•	•
guardsize	线程栈末尾的警戒缓冲区大小（字节数）	•	•	•	•
stackaddr	线程栈的最低地址	•	•	•	•
stacksize	线程栈的最小长度（字节数）	•	•	•	•

```
#include <pthread.h>
int pthread_attr_getguardsize(const pthread_attr_t *restrict attr,
                              size_t *restrict guardsize);
int pthread_attr_setguardsize(pthread_attr_t *attr,
                              size_t guardsize);
```

2. 线程属性

- POSIX.1定义的线程属性

名称	描述	FreeBSD 8.0	Linux 3.2.0	Mac OS X 10.6.8	Solaris 10
detachstate	线程的分离状态属性	•	•	•	•
guardsize	线程栈末尾的警戒缓冲区大小（字节数）	•	•	•	•
stackaddr	线程栈的最低地址	•	•	•	•
stacksize	线程栈的最小长度（字节数）	•	•	•	•

```
#include<pthread.h>
int pthread_attr_getstack(const pthread_attr_t *restrict attr,
                          void **restrict stackaddr, size_t *restrict stacksize);
int pthread_attr_setstack(pthread_attr_t *attr,
                          void *stackaddr,  size_t stacksize);
```

2. 线程属性

- POSIX.1定义的线程属性

名称	描述	FreeBSD 8.0	Linux 3.2.0	Mac OS X 10.6.8	Solaris 10
detachstate	线程的分离状态属性	•	•	•	•
guardsize	线程栈末尾的警戒缓冲区大小（字节数）	•	•	•	•
stackaddr	线程栈的最低地址	•	•	•	•
stacksize	线程栈的最小长度（字节数）	•	•	•	•

```
#include<pthread.h>
int pthread_attr_getstacksize(const pthread_attr_t *restrict attr,
                              size_t *restrict stacksize);
int pthread_attr_setstacksize(pthread_attr_t *attr,
                              size_t stacksize);
```

3. 同步属性

- 互斥量属性

```
#include <pthread.h>
int pthread_mutexattr_init(pthread_mutexattr_t * attr);
int pthread_mutexattr_destroy(pthread_mutexattr_t *attr);
```

3. 同步属性

- 互斥量属性

```
#include <pthread.h>
int pthread_mutexattr_getpshared(const pthread_mutexattr_t
                                 *restrict attr, int *restrict pshared);
int pthread_mutexattr_setpshared( pthread_mutexattr_t *attr,
                                 int pshared);
```

3. 同步属性

- 互斥量属性

```
#include <pthread.h>
int pthread_mutexattr_getrobust(const pthread_mutexattr_t
                    *restrict attr, int *restrict robust);
int pthread_mutexattr_setrobust(pthread_mutexattr_t *attr,
                    int robust);

int pthread_mutex_consistent(pthread_mutex_t *mutex);
```

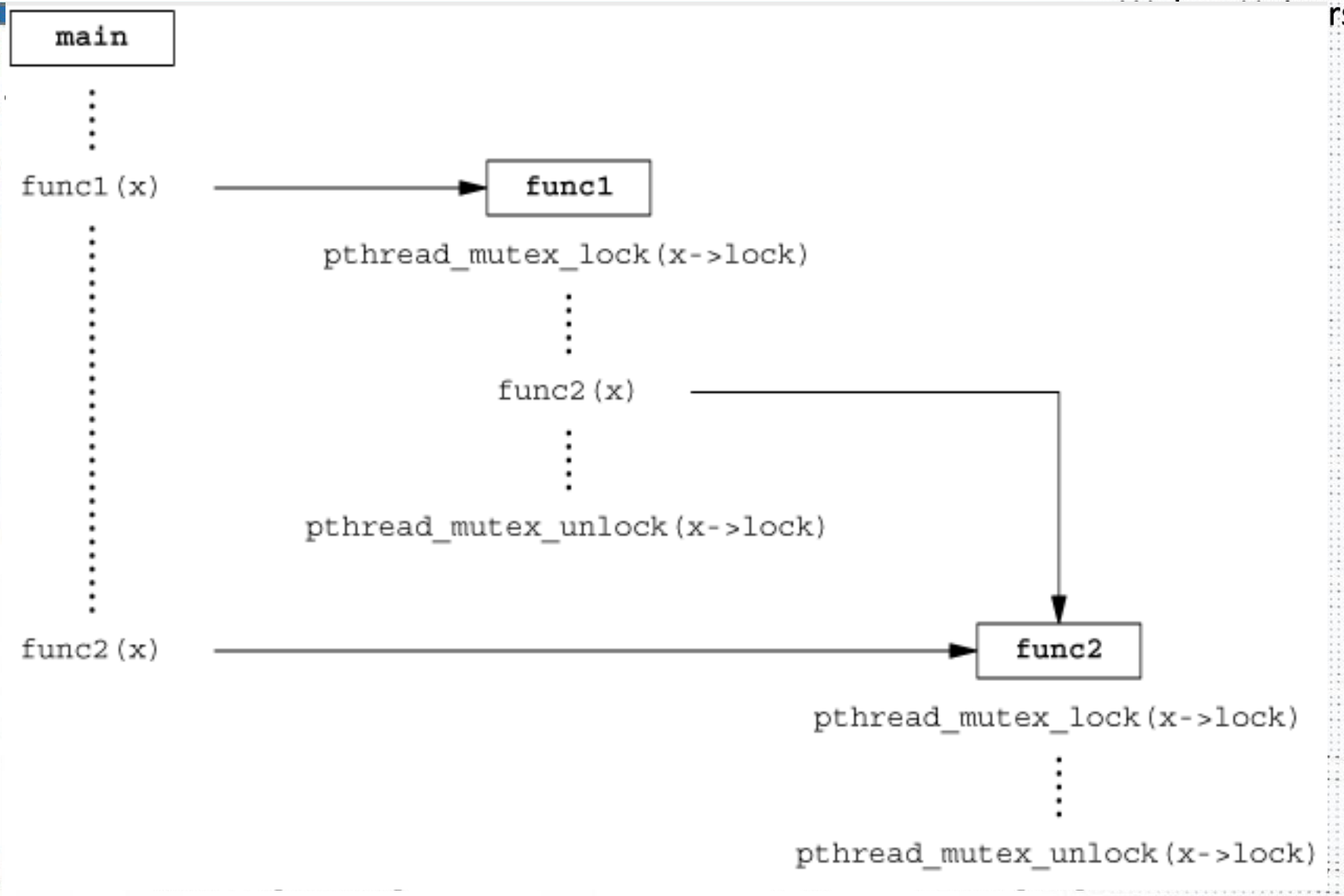
3. 同步属性

- 互斥量属性
  - 互斥量类型

互斥量类型	没有解锁时重新加锁?	不占用时解锁?	在已解锁时解锁?
PTHREAD_MUTEX_NORMAL	死锁	未定义	未定义
PTHREAD_MUTEX_ERRORCHECK	返回错误	返回错误	返回错误
PTHREAD_MUTEX_RECURSIVE	允许	返回错误	返回错误
PTHREAD_MUTEX_DEFAULT	未定义	未定义	未定义

```
#include <pthread.h>
int pthread_mutexattr_gettype(const pthread_mutexattr_t
                              *restrict attr, int *restrict type);
int pthread_mutexattr_settype(pthread_mutexattr_t *attr,
                              int type);
```

3. 同步属性


main
func1(x)
func1
pthread_mutex_lock(x->lock)
func2(x)
pthread_mutex_unlock(x->lock)
func2(x)
func2
pthread_mutex_lock(x->lock)
pthread_mutex_unlock(x->lock)

3. 同步属性

- 读写锁属性

```
#include <pthread.h>
int pthread_rwlockattr_init (pthread_rwlockattr_t * attr);
int pthread_rwlockattr_destroy ( pthread_rwlockattr_t *attr);

int pthread_rwlockattr_getshared (const pthread_rwlockattr_t
                    *restrict attr, int *restrict pshared);
int pthread_rwlockattr_setshared (pthread_rwlockattr_t
                    int pshared);
```

3. 同步属性

- 条件变量属性

```
#include <pthread.h>
int  pthread_condattr_init (pthread_condattr_t  *  attr);
int  pthread_condattr_destroy ( pthread_condattr_t  *attr);
```

3. 同步属性

- 条件变量属性

```
#include <pthread.h>
int pthread_condattr_getpshared(pthread_condattr_t
                    *restrict attr, int *restrict pshared);
int pthread_condattr_setpshared(pthread_condattr_t
                    *attr, int pshared);
```

3. 同步属性

- 条件变量属性

```
#include <pthread.h>
int pthread_condattr_getclock(pthread_condattr_t *restrict attr,
                              clockid_t *restrict clock_id);
int pthread_condattr_setclock(pthread_condattr_t *attr,
                              clockid_t clock_id);
```

标识符	选项	说明
CLOCK_REALTIME		实时系统时间
CLOCK_MONOTONIC	_POSIX_MONOTONIC_CLOCK	不带负跳数的实时系统时间
CLOCK_PROCESS_CPUTIME_ID	_POSIX_CPUTIME	调用进程的 CPU 时间
CLOCK_THREAD_CPUTIME_ID	_POSIX_THREAD_CPUTIME	调用线程的 CPU 时间

3. 同步属性

- 屏蔽属性

```
#include <pthread.h>
int pthread_barrierattr_init(pthread_barrierattr_t * attr);
int pthread_barrierattr_destroy( pthread_barrierattr_t *attr);

int pthread_barrierattr_getshared(const pthread_barrierattr_t
                    *restrict attr, int *restrict pshared);
int pthread_barrierattr_setshared(pthread_barrierattr_t *attr,
                    int pshared);
```

4. 重入

- 线程安全
  - Single UNIX Specification中定义的所有函数除了列出的函数，都是线程安全的。
  - ctermid和tmpnam函数在参数传入空指针时不能保证线程安全
  - wcrtomb和wcsrtombs函数在参数mbstate_t传入空指针时不能保证线程安全
  - 支持线程安全函数的操作系统实现在<unistd.h>中定义符号_POSIX_THREAD_SAFE_FUNCTIONS
  - 应用程序通过sysconf函数传入_SC_THREAD_SAFE_FUNCTIONS参数运行时检查是否支持线程安全函数

basename	getchar_unlocked	getservent	putc_unlocked
catgets	getdate	getutxent	putchar_unlocked
crypt	getenv	getutxid	putenv
dbm_clearerr	getgrent	getutxline	pututxline
dbm_close	getgrgid	gmtime	rand
dbm_delete	getgrnam	hcreate	readdir
dbm_error	gethostent	hdestroy	setenv
dbm_fetch	getlogin	hsearch	setgrent
dbm_firstkey	getnetbyaddr	inet_ntoa	setkey
dbm_nextkey	getnetbyname	l64a	setpwent
dbm_open	getnetent	lgamma	setutxent
dbm_store	getopt	lgammaf	strerror
dirname	getprotobyname	lgammal	strsignal
dlerror	getprotobynumber	localeconv	strtok
drand48	getprotoent	localtime	system
encrypt	getpwent	lrand48	ttyname
endgrent	getpwnam	mrand48	unsetenv
endpwent	getpwuid	nftw	wcstombs
endutxent	getservbyname	nl_langinfo	wctomb
getc_unlocked	getservbyport	ptsname	

4. 重入

- 线程安全

getgrgid_r	localtime_r
getgrnam_r	readdir_r
getlogin_r	strerror_r
getpwnam_r	strtok_r
getpwuid_r	ttyname_r
gmtime_r	

4. 重入

- 线程安全
  - 如果一个函数对多个线程来说是可重入的，那这个函数是线程安全的。但并不能说明对信号处理程序来说该函数可是可重入的。
  - 如果函数对异步信号处理程序的重入是安全的，那就可以说函数是异步信号安全的。

abort	faccessat	linkat	select	socketpair
accept	fchmod	listen	sem_post	stat
access	fchmodat	lseek	send	symlink
aio_error	fchown	lstat	sendmsg	symlinkat
aio_return	fchownat	mkdir	sendto	tcdrain
aio_suspend	fcntl	mkdirat	setgid	tcflow
alarm	fdatasync	mkfifo	setpgid	tcflush
bind	fexecve	mkfifoat	setsid	tcgetattr
cfgetispeed	fork	mknod	setsockopt	tcgetpgrp
cfgetospeed	fstat	mknodat	setuid	tcsendbreak
cfsetispeed	fstatat	open	shutdown	tcsetattr
cfsetospeed	fsync	openat	sigaction	tcsetpgrp
chdir	ftruncate	pause	sigaddset	time
chmod	futimens	pipe	sigdelset	timer_getoverrun
chown	getegid	poll	sigemptyset	timer_gettime
clock_gettime	geteuid	posix_trace_event	sigfillset	timer_settime
close	getgid	pselect	sigismember	times
connect	getgroups	raise	signal	umask
creat	getpeername	read	sigpause	uname
dup	getpgrp	readlink	sigpending	unlink
dup2	getpid	readlinkat	sigprocmask	ulinkat
execl	getppid	recv	sigqueue	utime
execle	getsockname	recvfrom	sigset	utimensat
execv	getsockopt	recvmsg	sigsuspend	utimes
execve	getuid	rename	sleep	wait
_Exit	kill	renameat	socketmark	waitpid
_exit	link	rmdir	socket	write

ty

4. 重入

- FILE对象的线程安全管理

```
#include <stdio.h>
int  ftrylockfile (FILE  *fp);

void  flockfile (FILE  *fp);
void  funlockfile (FILE  *fp);
```

5. 线程特定数据

- 线程特定数据，也称为线程私有数据
  - 存储和查询某个特定线程相关数据的一种机制
- 采用线程特定数据的原因
  - 为了维护基于每线程（per-thread）的数据
  - 提供让基于进程的接口适应多线程环境的机制（如errno）
- 管理线程特定数据的函数可以提高线程间的数据独立性，使得线程不太容易访问其它线程的线程特定数据

5. 线程特定数据

- 创建与线程特定数据关联的键，该键将用于获取对线程特定数据的访问

```
#include <pthread.h>

int  pthread_key_create (pthread_key_t  *keyp,
                         void  (*destructor) (void *));
```

5. 线程特定数据

- 创建键时避免出现冲突的方法

```
void destructor(void *);

pthread_key_t key;
int init_done = 0;

int
threadfunc(void *arg)
{
      if (!init_done) {
            init_done = 1;
            err = pthread_key_create(&key, destructor);
      }
      ...
}
```

5. 线程特定数据

- 创建键时避免出现冲突的方法

```
#include <pthread.h>

pthread_once_t  initflag = PTHREAD_ONCE_INIT;

int  pthread_once (pthread_once_t  *initflag, void (*initfn) (void));
```

5. 线程特定数据

- 创建键时避免出现冲突的方法

```
void destructor(void *);

pthread_key_t key;
pthread_once_t init_done = PTHREAD_ONCE_INIT;

void
thread_init(void)
{
     err = pthread_key_create(&key, destructor);
}

int
threadfunc(void *arg)
{
     pthread_once(&init_done, thread_init);
     ...
}
```

5. 线程特定数据

- 创建与线程特定数据关联的键，该键将用于获取对线程特定数据的访问

```
#include <pthread.h>

int  pthread_key_delete (pthread_key_t  key);
```

5. 线程特定数据

- 建立键和线程特定数据的关联关系

```
#include <pthread.h>

int *pthread_getspecific(pthread_key_t key);

int pthread_setspecific(pthread_key_t key, const void *value);
```

6. 取消选项

- 没有包含在pthread_attr_t结构中两个线程属性
  - 可取消状态
    - PTHREAD_CANCEL_ENABLE
    - PTHREAD_CANCEL_DISABLE
  - 可取消类型
    - PTHREAD_CANCEL_DEFERRED
    - PHTREAD_CANCEL_ASYNCHRONOUS

6. 取消选项

- 没有包含在pthread_attr_t结构中两个线程属性
  - 可取消状态
    - PTHREAD_CANCEL_ENABLE
    - PTHREAD_CANCEL_DISABLE

```
#include <pthread.h>

int pthread_setcancelstate(int state, int *oldstate);
```

6. 取消选项

- 没有包含在pthread_attr_t结构中两个线程属性
  - 取消点

accept	mq_timedsend	pthread_join	sendto
aio_suspend	msgrcv	pthread_testcancel	sigsuspend
clock_nanosleep	msgsnd	pwrite	sigtimedwait
close	msync	read	sigwait
connect	nanosleep	readv	sigwaitinfo
creat	open	recv	sleep
fcntl	openat	recvfrom	system
fdatasync	pause	recvmsg	tcdrain
fsync	poll	select	wait
lockf	pread	sem_timedwait	waitid
mq_receive	pselect	sem_wait	waitpid
mq_send	pthread_cond_timedwait	send	write
mq_timedreceive	pthread_cond_wait	sendmsg	writev

6. 取消选项

- 添加取消点
  - 如果应用程序在很长的一段时间内都不会调用取消点函数，则可以添加自己的取消点

```
#include <pthread.h>

void  pthread_testcancel (void);
```

6. 取消选项

- 没有包含在pthread_attr_t结构中两个线程属性
  - 可取消类型
    - PTHREAD_CANCEL_DEFERRED
    - PHTREAD_CANCEL_ASYNCHRONOUS

```
#include <pthread.h>

int pthread_setcanceltype(int type, int *oldtype);
```

7. 线程和信号


```
#include <signal.h>

int pthread_sigmask(int how, const sigset_t *restrict set,
                    sigset_t *restrict oset);

int sigwait(const sigset_t *restrict set, int *restrict signop);

int pthread_kill(pthread_t thread, int signo);
```

8. 线程和fork

- 线程中调用fork
  - 子进程创建了整个进程地址空间的副本
  - 子进程继承了每个互斥量、读写锁和条件变量的状态
  - 如果父进程包含一个以上的线程，子进程在fork返回以后，如果不是紧接着调用exec，则需要清理锁状态

```
#include <pthread.h>

int  pthread_atfork(void  (*prepare)(void),
                    void  (*parent)(void),
                    void  (*child)(void));
```

9. 线程和I/O

- pread可以确保多线程对同一文件描述符中读取不同的内容
- pwrite可以解决并发线程对同一文件进行写操作的问题