

void

```
int pthread_create(..., // first two args are the same
                   int      (*start_routine)(void *),
                   void *    arg);
```

arg

void

void

start_routine

27.1

myarg_t

```
int pthread_join(pthread_t thread, void **value_ptr);
```

```
1  #include <pthread.h>
2
3  typedef struct  myarg_t {
4      int a;
5      int b;
6  } myarg_t;
7
8  void *mythread(void *arg) {
9      myarg_t *m = (myarg_t *) arg;
10     printf("%d %d\n", m->a, m->b);
11     return NULL;
12 }
13
14 int
15 main(int argc, char *argv[]) {
16     pthread_t p;
17     int rc;
18
19     myarg_t args;
20     args.a = 10;
21     args.b = 20;
22     rc = pthread_create(&p, NULL, mythread, &args);
23     ...
24 }
```

27.1

\$) Z\$

pthread_join()

pthread_t

pthread_create()

void

pthread_join()

27.2

myarg_t

myret_t

pthread_join()

myret_t

NULL

pthread_join()

NULL

```

1  #include <stdio.h>
2  #include <pthread.h>
3  #include <assert.h>
4  #include <stdlib.h>
5
6  typedef struct  myarg_t {
7      int a;
8      int b;
9  } myarg_t;
10
11 typedef struct  myret_t {
12     int x;
13     int y;
14 } myret_t;
15
16 void *mythread(void *arg) {
17     myarg_t *m = (myarg_t *) arg;
18     printf("%d %d\n", m->a, m->b);
19     myret_t *r = Malloc(sizeof(myret_t));
20     r->x = 1;
21     r->y = 2;
22     return (void *) r;
23 }
24
25 int
26 main(int argc, char *argv[]) {
27     int rc;
28     pthread_t p;
29     myret_t *m;

```

```

30
31     myarg_t args;
32     args.a = 10;
33     args.b = 20;
34     Pthread_create(&p, NULL, mythread, &args);
35     Pthread_join(p, (void **) &m);
36     printf("returned %d %d\n", m->x, m->y);
37     return 0;
38 }

```

27.2

int

27.3

```

void *mythread(void *arg) {
    int m = (int) arg;
    printf("%d\n", m);
    return (void *) (arg + 1);
}

int main(int argc, char *argv[]) {
    pthread_t p;
    int rc, m;
    Pthread_create(&p, NULL, mythread, (void *) 100);
    Pthread_join(p, (void **) &m);
    printf("returned %d\n", m);
    return 0;
}

```

27.3

27.2

```

1 void *mythread(void *arg) {
2     myarg_t *m = (myarg_t *) arg;
3     printf("%d %d\n", m->a, m->b);
4     myret_t r; // ALLOCATED ON STACK: BAD!
5     r.x = 1;
6     r.y = 2;
7     return (void *) &r;
8 }

```

r

mythread

pthread_create()

pthread_join()

procedure call

join

Web

join

join

\$) %

join

POSIX

lock

```
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

```
pthread_mutex_t lock;
pthread_mutex_lock(&lock);
x = x + 1; // or whatever your critical section is
pthread_mutex_unlock(&lock);
```

pthread_mutex_lock()

lack of proper

initialization

POSIX

PTHREAD_MUTEX_

INITIALIZER

```
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
```

pthread_mutex_init()

```
int rc = pthread_mutex_init(&lock, NULL);
assert(rc == 0); // always check success!
```

NULL

pthread_mutex_destroy()

UNIX

27.4

```
// Use this to keep your code clean but check for failures
// Only use if exiting program is OK upon failure
void Pthread_mutex_lock(pthread_mutex_t *mutex) {
    int rc = pthread_mutex_lock(mutex);
    assert(rc == 0);
}
```

27.4

pthread

```
int pthread_mutex_trylock(pthread_mutex_t *mutex);
int pthread_mutex_timedlock(pthread_mutex_t *mutex,
                             struct timespec *abs_timeout);
```

trylock

timedlock

timedlock

trylock

\$) &

POSIX

condition variable

```
int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex);
int pthread_cond_signal(pthread_cond_t *cond);
```

pthread_cond_wait()

```
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
```

```

Pthread_mutex_lock(&lock);
while (ready == 0)
    Pthread_cond_wait(&cond, &lock);
Pthread_mutex_unlock(&lock);

```

ready

```

Pthread_mutex_lock(&lock);
ready = 1;
Pthread_cond_signal(&cond);
Pthread_mutex_unlock(&lock);

```

ready

pthread_cond_wait()

while

if

while

pthread

```

while (ready == 0)
    ; // spin

```

ready = 1;

CPU

[X+10]

\$) ž

-pthread pthread.h pthread

```
prompt> gcc -o main main.c -Wall -pthread
```

main.c pthreads

\$) ž(

pthread

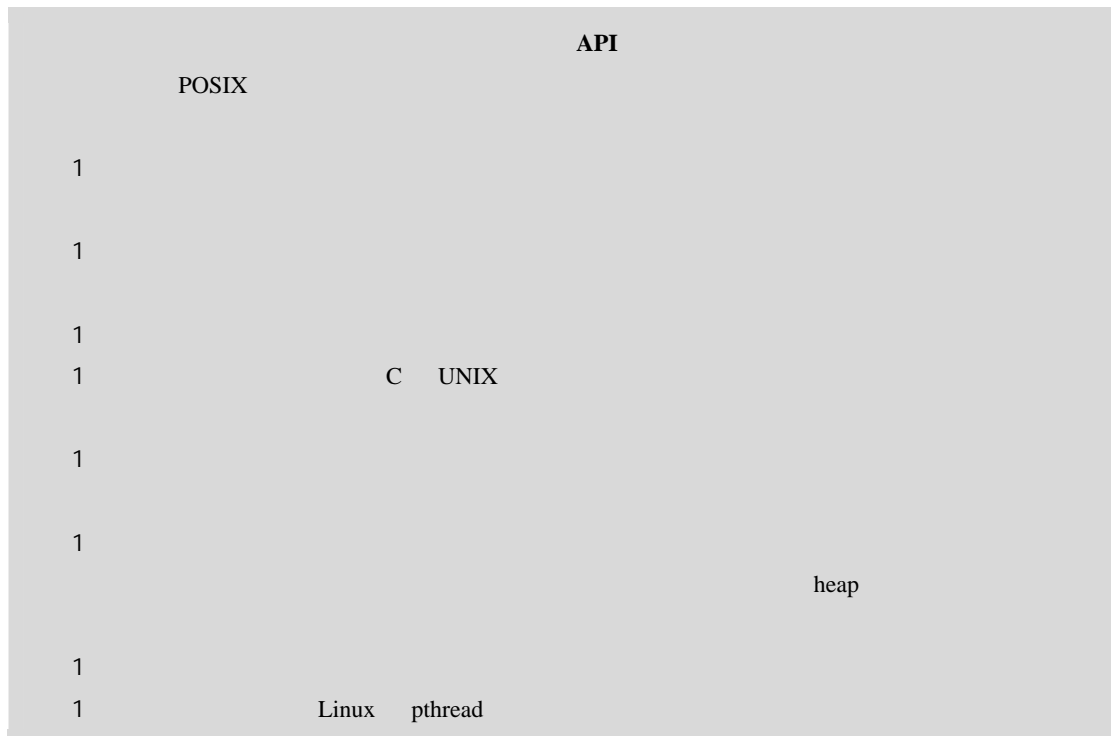
API

Linux

man -k pthread

API

API



[B89] An Introduction to Programming with Threads Andrew D. Birrell
DEC Technical Report, January, 1989

[B97] Programming with POSIX Threads David R. Butenhof
Addison-Wesley, May 1997

[B+96] PThreads Programming: A POSIX Standard for Better Multiprocessing
Dick Buttlar, Jacqueline Farrell, Bradford Nichols O'Reilly, September 1996
O'Reilly

Perl Python

JavaScript

Crockford

JavaScript: The Good Parts

[K+96] Programming With Threads
Steve Kleiman, Devang Shah, Bart Smaalders Prentice Hall, January 1996

[X+10] Ad Hoc Synchronization Considered Harmful
Weiwei Xiong, Soyeon Park, Jiaqi Zhang, Yuanyuan Zhou, Zhiqiang Ma OSDI 2010, Vancouver, Canada