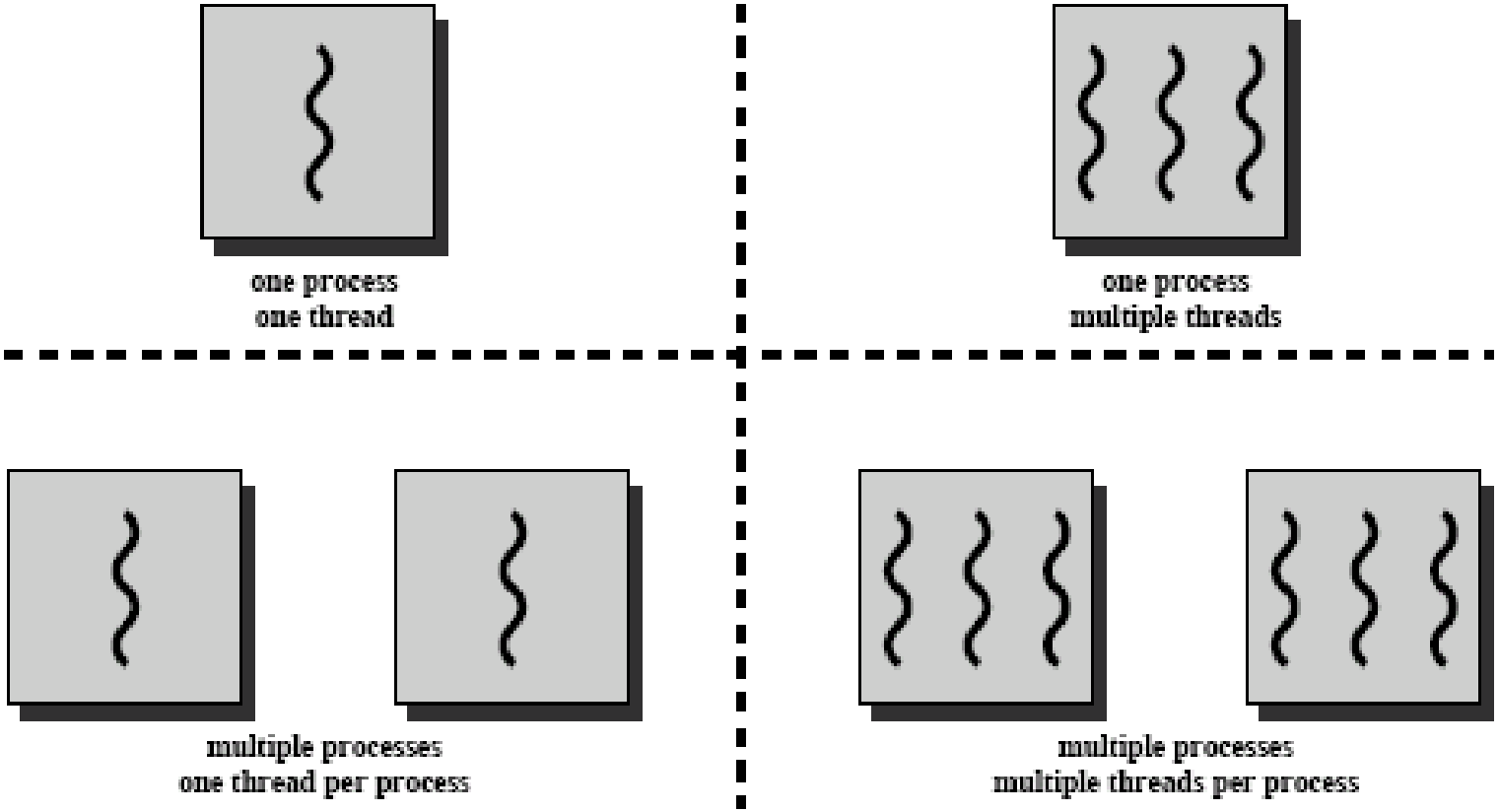
系统编程

基于TaiShan服务器/openEuler OS 的实践

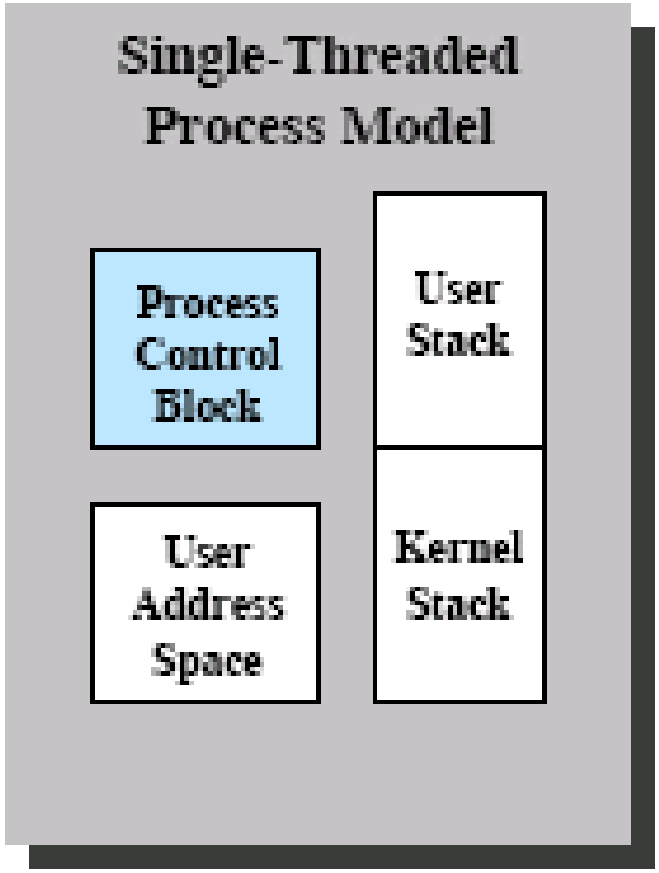
第三讲：多线程编程 – 线程控制

操作系统中的线程和进程

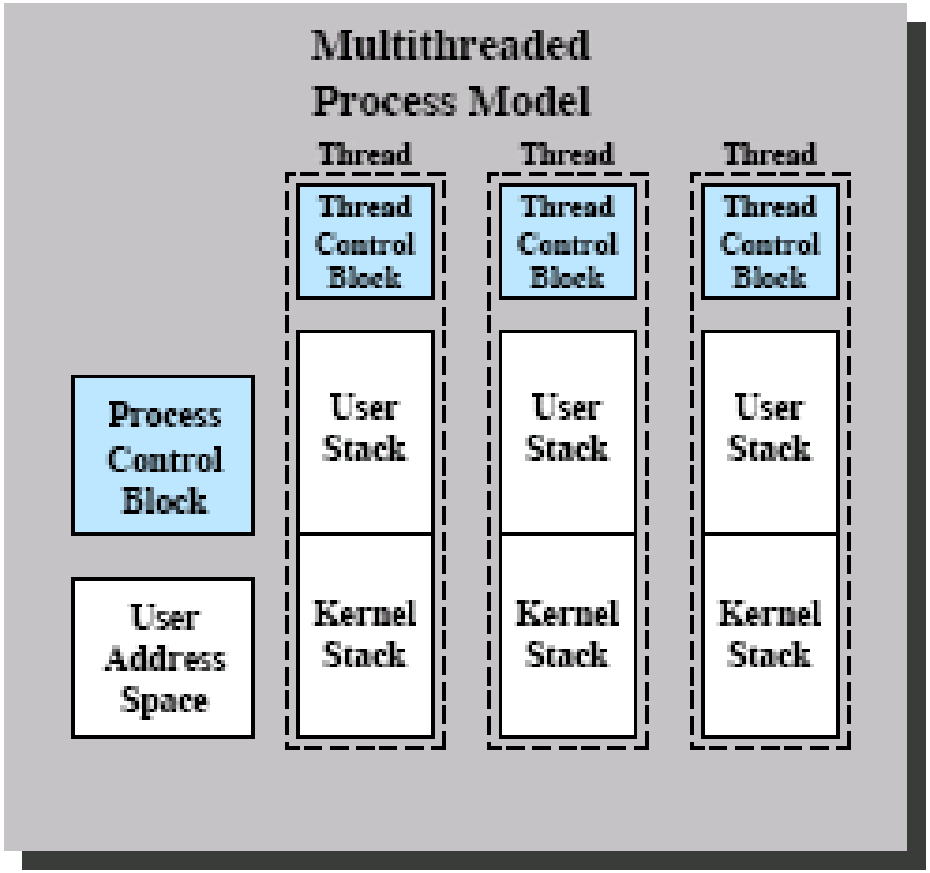


指令追踪

单线程进程模型 vs.多线程进程模型



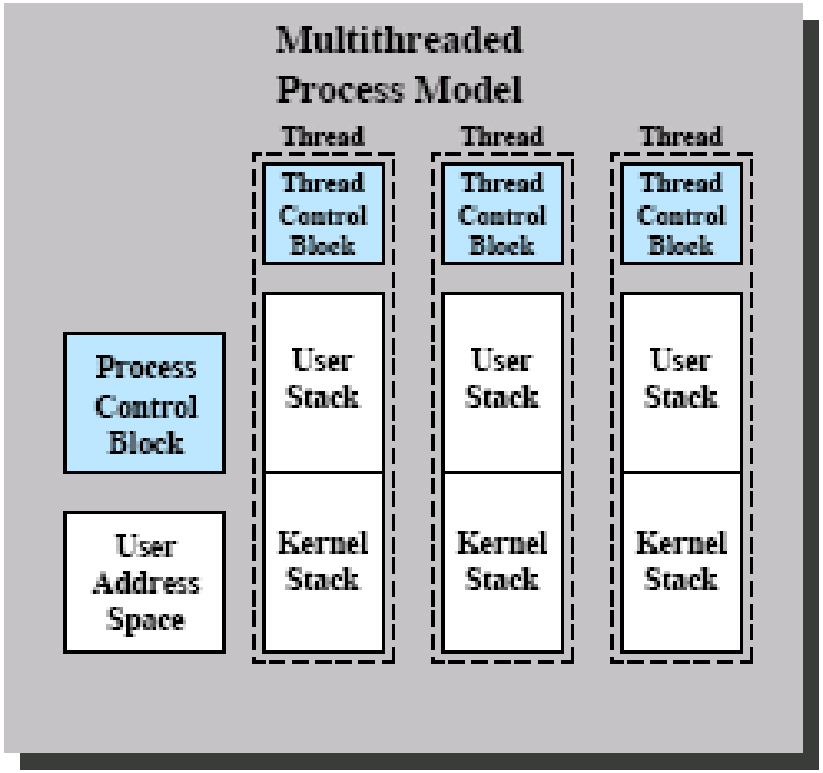
DOS



Solaris, Linux, Window 2000

线程 vs. 进程

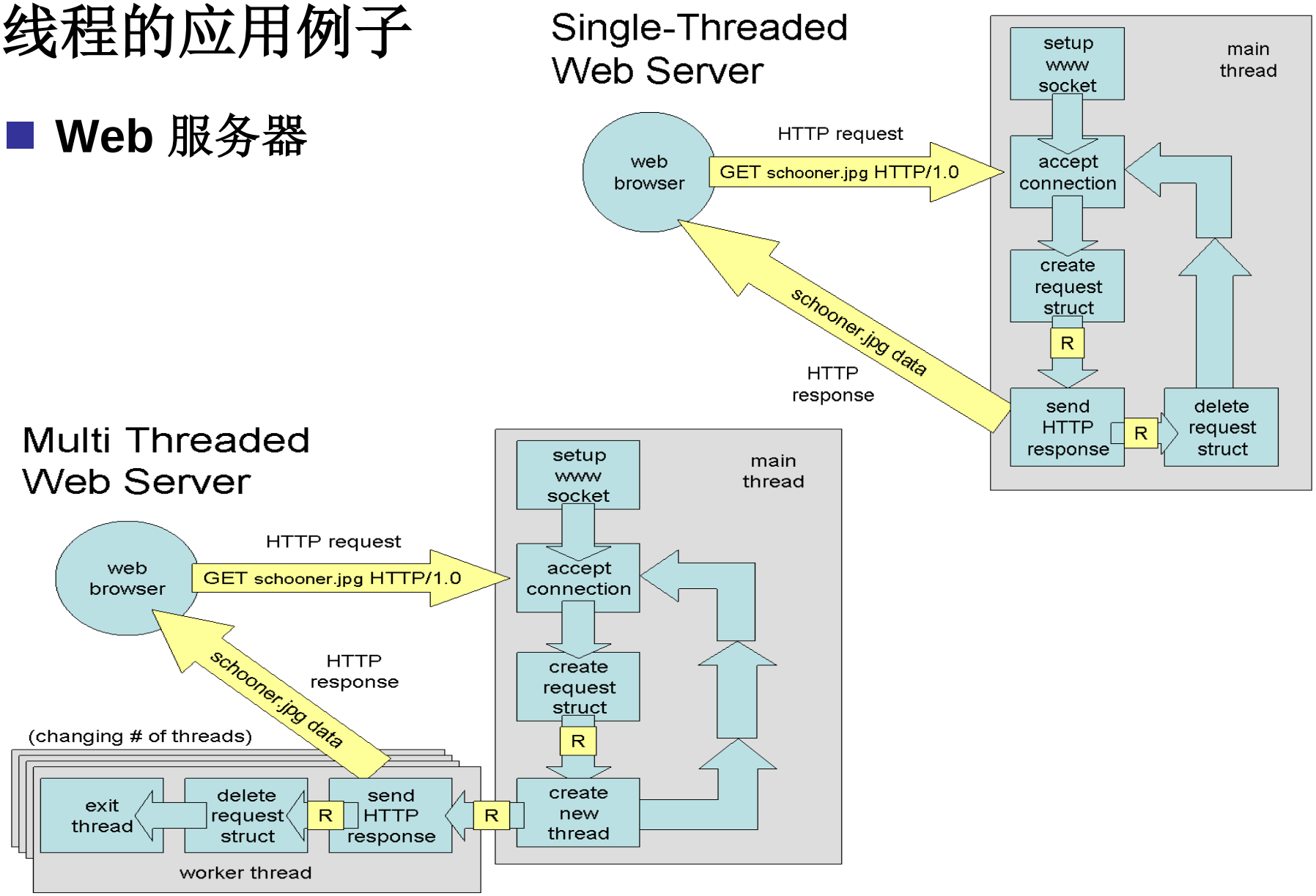
- 生成时间短
- 退出时间短
- 线程间通信时间短
  - 共享地址空间
  - 不用内核传递消息
- 线程间切换时间短
  - 进程资源为其所有线程共享
  - 不用恢复用户地址空间



- 进程
  - 资源分配单元
  - 处理机分配单元
- 线程
  - 处理机分配单元

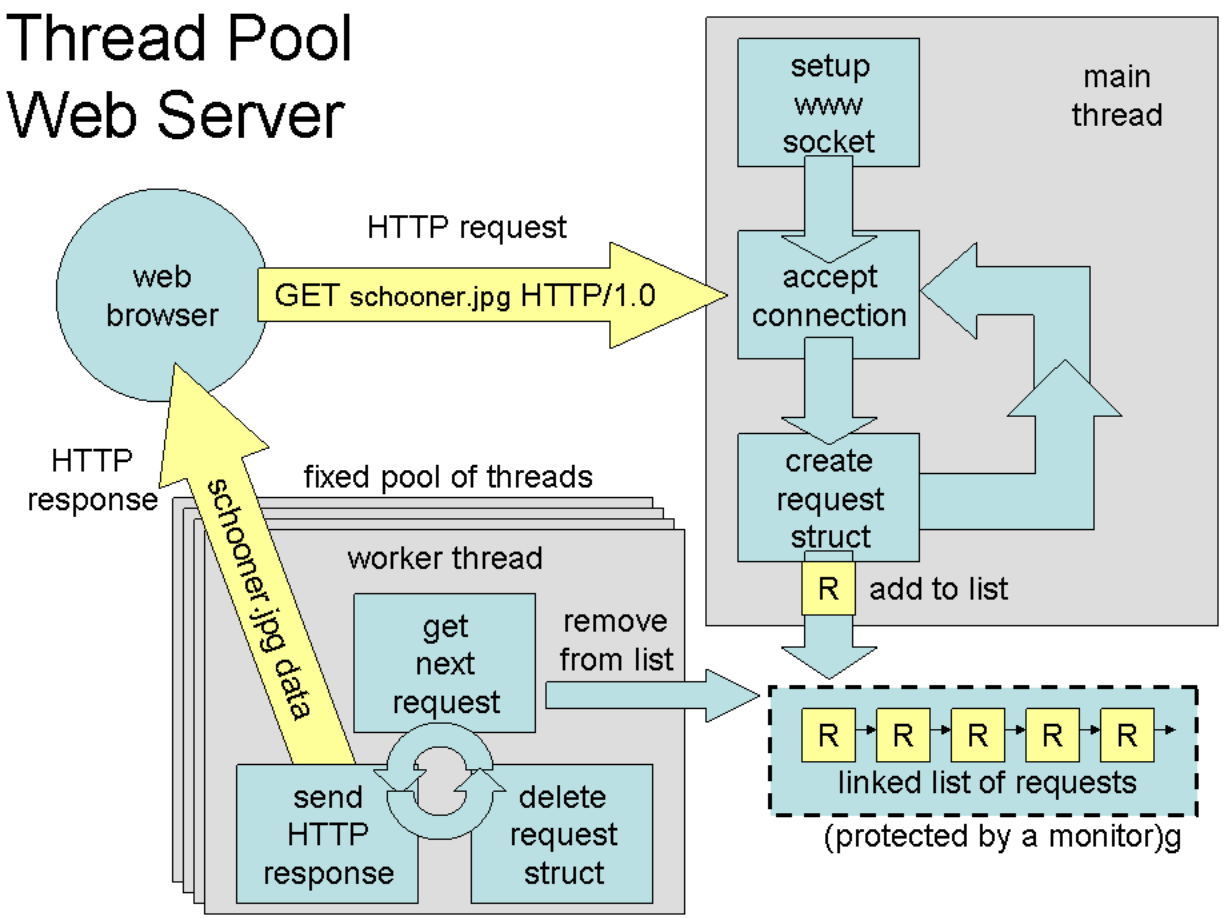
线程的应用例子

- **Web 服务器**

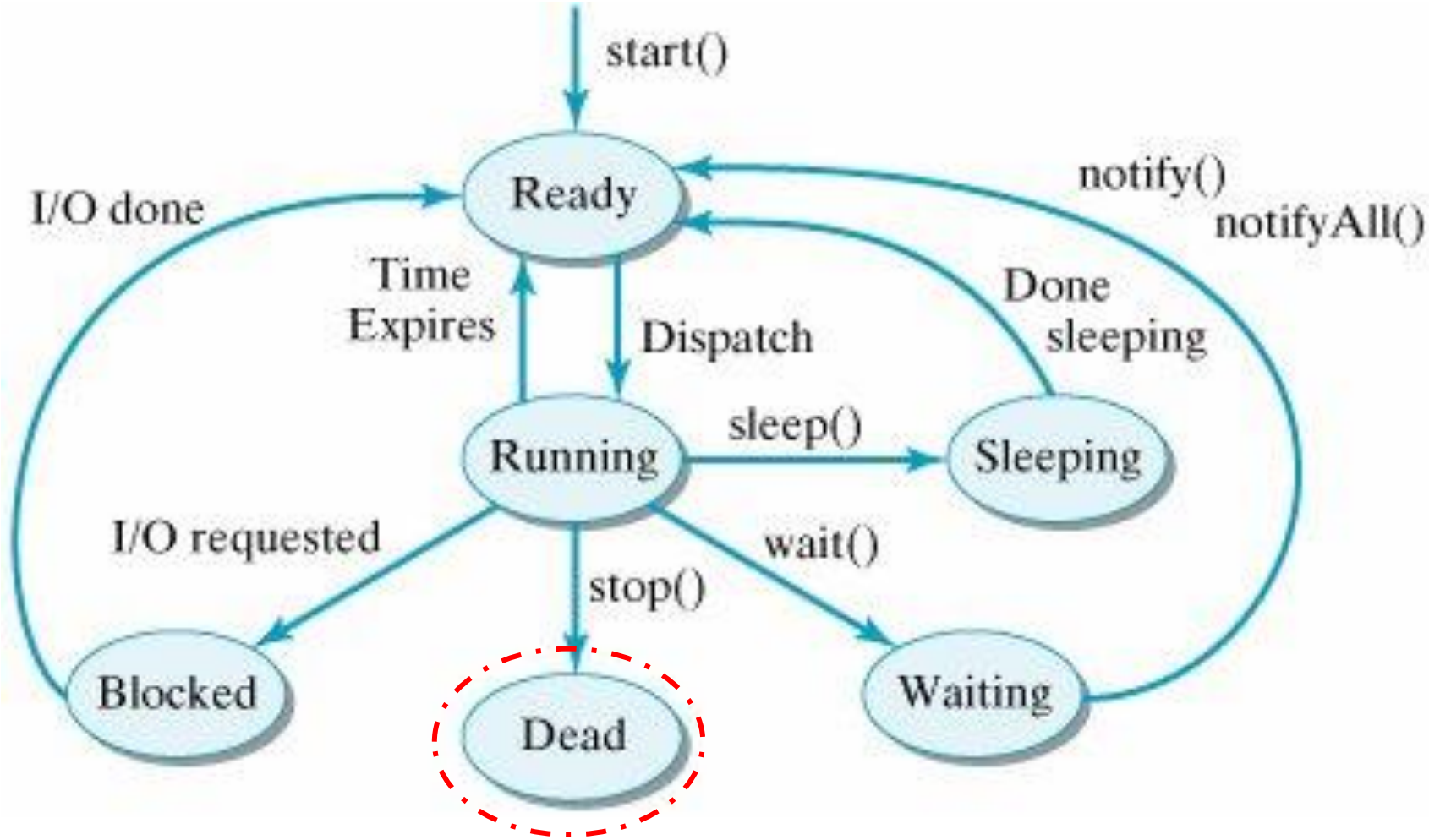


线程的应用例子

- **Web 服务器**



线程状态：一个线程的生命周期



线程控制 – 线程创建

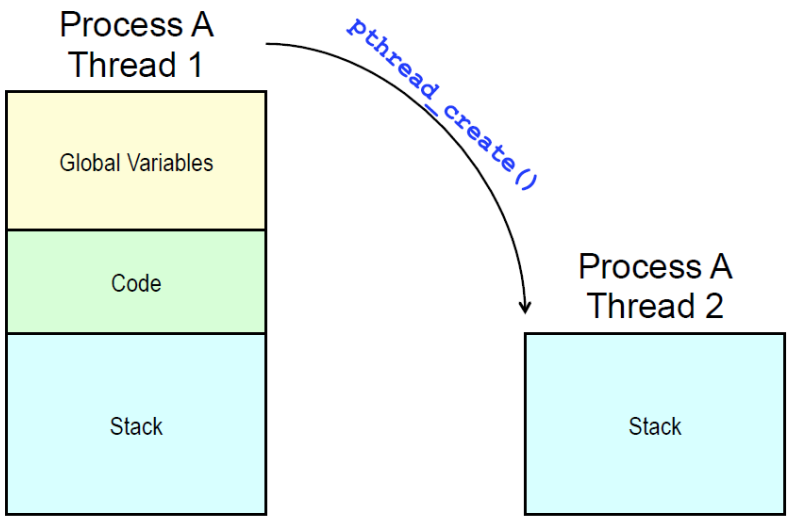
pthread_create()

```
#include <pthread.h>

int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
                   void *(*start_routine) (void *), void *arg);

Compile and link with -pthread.
```

- thread: 指向线程标识符的指针
- attr: 设置线程属性
- start_routine: 线程运行函数的起址
- arg: 运行函数的参数



✓ 没有复制整个进程

```
#include <pthread.h>

int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
                   void *(*start_routine) (void *), void *arg);
```

yuhong@FedoraDVD13:~/C

文件(F) 编辑(E) 查看(V) 终端(T) 帮助(H)

```
#include <pthread.h>
#include <stdio.h>

/* Prints x's to stderr, The parameter is unused, does not return. */

void* print_xs(void* unused)
{
        while (1)
                fputc('o',stderr);
        return NULL;
}

/* The main program. */
int main()
{
        pthread_t thread_id;

        pthread_create(&thread_id,NULL,&print_xs,NULL);
        while(1)
                fputc('_',stderr);
        return 0;
}
~
```

```
#include <pthread.h>

int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
                   void *(*start_routine) (void *), void *arg);
```

- void* 型参数：如果需要传输多项类型不同数据
  - 为线程函数定义结构类型，包含其所需的所有数据
  - 多个线程可复用同一线程函数
    - SIMD

2. 172.31.234.200 (szu)

Re-attach Fullscreen Stay on top Duplicate

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

struct n_chars {
        char chars;
        int  amount;
};

void *print_n_char(void *params)
{
        struct n_chars *p = (struct n_chars*)params;
        int i;

        for (i = 0; i < p->amount; i++)
                fputc(p->chars,stderr);
        return NULL;
}

int main(int argc, char *argv[])
{
        pthread_t t_id1, t_id2;
        struct n_chars t1_args, t2_args;

        t1_args.chars = '*';
        t1_args.amount = 3;
        pthread_create(&t_id1, NULL, &print_n_char, &t1_args);

        t2_args.chars = '^';
        t2_args.amount = 6;
        pthread_create(&t_id2, NULL, &print_n_char, &t2_args);
        return 0;
}
```

```
[szu@taishan02-vm-10 threads]$ gcc -o join join.c -lpthread
[szu@taishan02-vm-10 threads]$ ./join
[szu@taishan02-vm-10 threads]$ ./join
[szu@taishan02-vm-10 threads]$ ./join
**^^^^^^[szu@taishan02-vm-10 threads]$ ./join
[szu@taishan02-vm-10 threads]$ ./join
**^^^[szu@taishan02-vm-10 threads]$ ./join
**^^[szu@taishan02-vm-10 threads]$
```

- 创建进程的同时创建主线程
  - 主线程管理进程资源
  - 主线程退出
    - 进程结束
    - 资源回收
    - 其线程退出
- 线程间没有父子关系

线程控制 – 等待指定线程的结束

```
#include <pthread.h>

int pthread_join(pthread_t thread, void **retval);

Compile and link with -pthread.
```

- 参数
  - thread：等待线程的线程ID
  - retval：等待线程函数的返回值
- 返回值
  - 成功：0
  - 出错：error number

出错代码	出错原因
EDEADLK	死锁
EINVAL	● thread线程属性不是joinable ● 有另一个进程等待join线程thread
ESRCH	找不到ID为thread的线程

系统没有定义多个线程同时等待同一个线程的行为

2. 172.31.234.200 (szu)

Re-attach Fullscreen Stay on to

```
#include <pthread.h>
#include <stdio.h>

struct n_chars {
        char chars;
        int  amount;
};

void *print_n_char(void *params)
{
        struct n_chars *p = (struct n_chars*)params;
        int i;

        for (i = 0; i < p->amount; i++)
                fputc(p->chars,stderr);
        return NULL;
}

int main(int argc, char *argv[])
{
        pthread_t t_id1, t_id2;
        struct n_chars t1_args, t2_args;

        t1_args.chars = '*';
        t1_args.amount = 3;
        pthread_create(&t_id1, NULL, &print_n_char, &t1_args);

        t2_args.chars = '^';
        t2_args.amount = 6;
        pthread_create(&t_id2, NULL, &print_n_char, &t2_args);

        pthread_join(t_id1, NULL);
        pthread_join(t_id2, NULL);
        return 0;
}
```

```
[szu@taishan02-vm-10 threads]$ gcc -o withjoin withjoin.c -lpthread
[szu@taishan02-vm-10 threads]$ ./withjoin
***^^^^^^^[szu@taishan02-vm-10 threads]$ ./withjoin
***^^^^^^^[szu@taishan02-vm-10 threads]$ ./withjoin
***^^^^^^^[szu@taishan02-vm-10 threads]$ ./withjoin
***^^^^^^^[szu@taishan02-vm-10 threads]$ ./withjoin
***^^^^^^^[szu@taishan02-vm-10 threads]$ ./withjoin
***^^^^^^^[szu@taishan02-vm-10 threads]$
```

- 等待线程结束，释放相应资源
  - 有些线程选择自己释放资源
    - 线程属性为detached
    - 不能被pthread_join()
- 等待joinable线程失败
  - 僵死线程(zombie thread)
  - 消耗系统资源

线程控制 – 获取调用线程的ID

```
#include <pthread.h>
pthread_t pthread_self(void);
```

- 返回值：调用线程的ID，永远成功

线程控制 – 终止线程

- 自主终止
  - 线程函数结束
  - 函数pthread_exit()

```
#include <pthread.h>

void pthread_exit(void *retval);
```

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

```
If retval is not NULL, then pthread_join() copies the exit status of
the target thread (i.e., the value that the target thread supplied to
pthread_exit(3)) into the location pointed to by *retval. If the tar-
get thread was canceled, then PTHREAD_CANCELED is placed in *retval.
```

获取线程退出状态的例程（一）

```c
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>

void* adder(void* arg)
{
        int *operand = malloc(sizeof(int));
        *operand = *((int *)arg);
        *operand = 2 + *operand;

        pthread_exit((void*)operand);

}

int main()
{
        pthread_t thread;
        int *result;
        const int operand=3;
        pthread_create(&thread, NULL, &adder, (void *)&operand);
        pthread_join(thread,(void*)&result);
        printf("The result of the adder is %d.\n",*result);
}
~
```

```
[szu@taishan02-vm-10 threads]$ gcc -lpthread -o returnvalue returnvalue.c
[szu@taishan02-vm-10 threads]$ ./returnvalue
The result of the adder is 5.
```

```
[szu@taishan02-vm-10 threads]$ gcc -lpthread -o exitstatus exitstatus.c
[szu@taishan02-vm-10 threads]$ ./exitstatus
Thread 1 exits with code 1
Thread 2 exits with code 2
[szu@taishan02-vm-10 threads]$ cat exitstatus.c
#include <pthread.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

void *thr_fn1(void *arg)
{
        return ((void *)1);
}

void *thr_fn2(void *arg)
{
        pthread_exit((void *)2);
}


int main()
{
        pthread_t tid1, tid2;
        void *tret1, *tret2;

        if (pthread_create(&tid1, NULL, thr_fn1, NULL) < 0) {
                perror("Fail to create pthread tid1.\n");
                exit(-1);
        }

        if (pthread_create(&tid2, NULL, thr_fn2, NULL) < 0) {
                perror("Fail to create pthread tid2.\n");
                exit(-1);
        }

        pthread_join(tid1,&tret1);
        pthread_join(tid2,&tret2);
        printf("Thread 1 exits with code %d\n",(int *)tret1);
        printf("Thread 2 exits with code %d\n",(int *)tret2);
        exit(0);
}
[szu@taishan02-vm-10 threads]$
```

获取线程退出状态的例程（二）

yuhong@localhost:~/Cprogramming/thread

File Edit View Terminal Help

课堂练习：请将以下代码调试正确

```
void* compute_prime(void* arg)
{
        int candidate = 2;
        int n = *((int*)arg);

        while (1) {
                int factor;
                int is_prime = 1;
                /*Test primality by successive division*/
                for (factor = 2; factor < candidate; ++factor)
                        if (candidate % factor == 0){
                                is_prime = 0;
                                break;
                        }

                /*Is this the prime number we're looking for?*/
                if (is_prime) {
                        if (--n == 0)
                                /*Return the desired prime number as the thread return value. */
                                return (void*) candidate;
                }
                ++candidate;
        }
        return NULL;
}

int main()
{
        pthread_t thread;
        int which_prime = 10;
        int prime;

        pthread_create(&thread,NULL,&compute_prime,&which_prime);
        sleep(10);
        pthread_join(thread,(void*)&prime);
        printf("The %dth prime number is %d.\n", which_prime,prime);
        return 0;
}
```

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

void *compute_prime(void *arg)
{
        int *candidate = NULL;
        int n = *((int *)arg);
        candidate = malloc(sizeof(int));
        *candidate = 2;
        while (1){
                int factor;
                int is_prime = 1;
                for (factor = 2; factor < *candidate; ++factor)
                        if (*candidate % factor == 0){is_prime = 0; break;}
                if (is_prime){
                        if (--n == 0) return candidate;
                }
                ++(* candidate);
        }
        return NULL;
}

int main()
{
        pthread_t tid;
        int which_prime = 10;
        void *prime;
        if (pthread_create(&tid, NULL, &compute_prime, &which_prime) < 0) {
                perror("Fail to create pthread tid1.\n"); exit(EXIT_FAILURE);
        }
        pthread_join(tid,&prime);
        printf("The %dth prime number is %d.\n",which_prime,*((int *)prime));
        exit(EXIT_SUCCESS);
}
```

参考修改

线程控制 – 终止线程

- 被动终止：pthread_cancel()

```
#include <pthread.h>
  int pthread_cancel(pthread_t thread);
```

成功：返回0　　失败：返回非0的出错代码

成功并不意味着thread会终止，与目标线程的两个属性有关

- cancelability state
  - PTHREAD_CANCEL_ENABLE(缺省):取决于cancelability type
    - PTHREAD_CANCEL_ASYNCHRONOUS：任意时间都可取消该线程
    - PTHREAD_CANCEL_DEFERRED:设置可取消点,收到取消请求，将其放进队列,等线程运行至下一个取消点再退出
  - PTHREAD_CANCEL_DISENABLE:取消请求被阻塞直到状态改变

- pthread_cancel()应用例子
- 在man手册页中

```c
#include <pthread.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <unistd.h>
#define handle_error_en(en, msg) \
        do { errno = en; perror(msg); exit(EXIT_FAILURE); } while (0)

static void * thread_func(void *ignored_argument)
{
        int s;
        if ((s = pthread_setcancelstate(PTHREAD_CANCEL_DISABLE, NULL)) != 0)
                handle_error_en(s, "pthread_setcancelstate");
        printf("thread_func(): started; cancellation disabled\n");
        sleep(5);
        printf("thread_func(): about to enable cancellation\n");
        if ((s = pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, NULL)) != 0)
                handle_error_en(s, "pthread_setcancelstate");
        sleep(1000);        /* Should get canceled while we sleep */
        /* Should never get here */
        printf("thread_func(): not canceled!\n");
        return NULL;
}
```

```c
int main(void)
{
        pthread_t thr;
        void *res;
        int s;
        if ((s = pthread_create(&thr, NULL, &thread_func, NULL)) != 0)
                handle_error_en(s, "pthread_create");
        sleep(2);           /* Give thread a chance to get started */
        printf("main(): sending cancellation request\n");
        if ((s = pthread_cancel(thr)) != 0)
                handle_error_en(s, "pthread_cancel");
        if ((s = pthread_join(thr, &res)) != 0)
                handle_error_en(s, "pthread_join");
        if (res == PTHREAD_CANCELED)
                printf("main(): thread was canceled\n");
        else
                printf("main(): thread wasn't canceled (shouldn't happen!)\n");
        exit(EXIT_SUCCESS);
}
```

```
[szu@taishan02-vm-10 threads]$ gcc -o cancelthread cancelthread.c -lpthread
[szu@taishan02-vm-10 threads]$ ./cancelthread
thread_func(): started; cancellation disabled
main(): sending cancellation request
thread_func(): about to enable cancellation
main(): thread was canceled
```