Operating Systems
Labs

Lab1: user level thread

Qiu Qichen

Outline

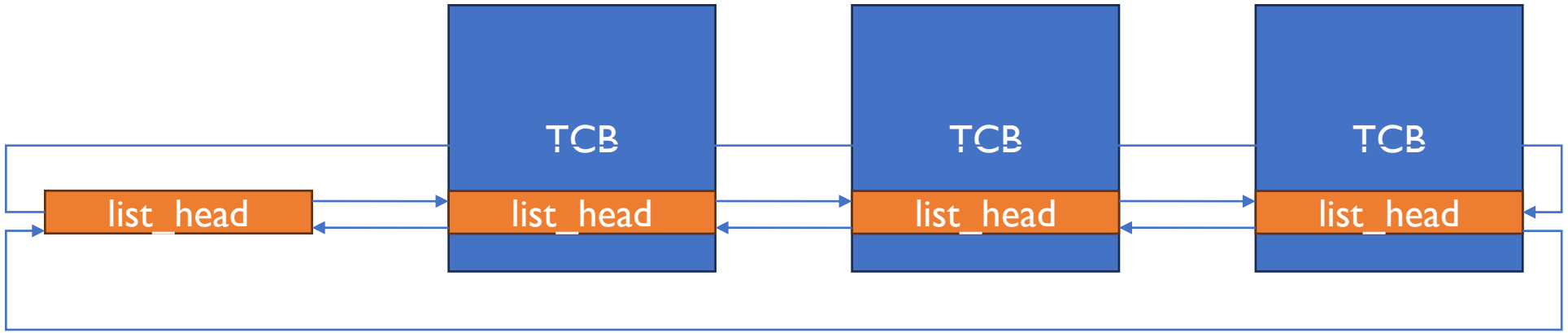
- **Solution**
- **Challenge**

uthread_create

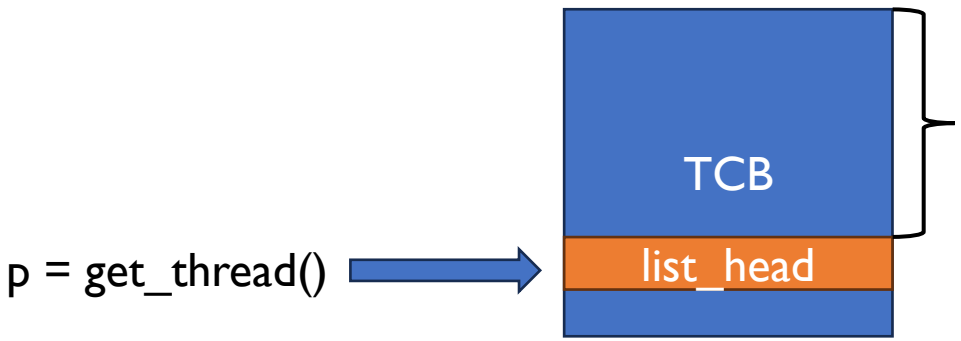
- 设置入口为_uthread_entry
- 参数为uthread,func,arg
- 节点入队

```
long long stack =
    (((long long)(&uthread->stack) + STACK_SIZE) & -16L) - 8;
make_dummpy_context(&uthread->context);
uthread->context.rip = (long long)_uthread_entry;
uthread->context.rsp = stack;
uthread->state = THREAD_INIT;
uthread->context.rdi = (long long)uthread;
uthread->context.rsi = (long long)func;
uthread->context.rdx = (long long)arg;
INIT_LIST_HEAD(&uthread->list);
push_back(&schedule_thread, &uthread->list);
```

Linux-style list



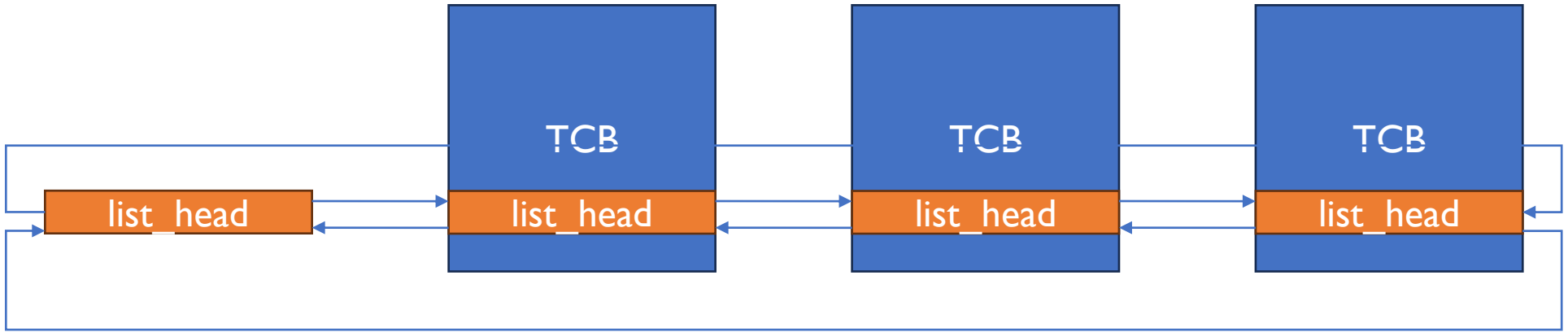
- Example: get_thread from list




```
struct uthread {
  char stack[STACK_SIZE];
  struct context context;
  enum thread_state state;
  struct list_head list;
  const char *name;
};
```

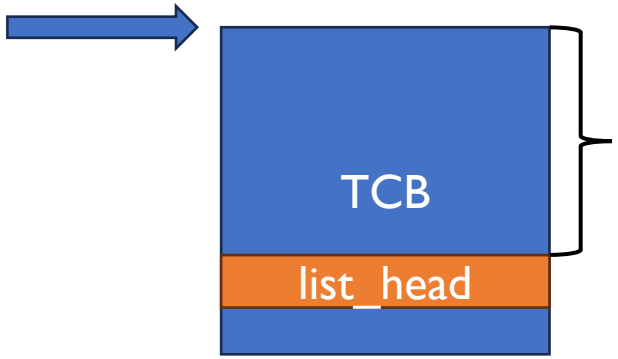
```
((struct uthread *)((char *)pointer_to_node - offsetof(struct uthread, list)))
```

Linux-style list



- Example:

p = (struct uthread*)((char*)p – 4192)



offset=4192 bytes

```
struct uthread {
  char stack[STACK_SIZE];
  struct context context;
  enum thread_state state;
  struct list_head list;
  const char *name;
};
```

```
((struct uthread *)((char *)pointer_to_node - offsetof(struct uthread, list)))
```

schedule

```
void schedule() {
    struct uthread *uthread = NULL;

  while (1) {
    if (is_empty(&schedule_thread)) {
      break;
    }
    uthread = get_thread();

    if (NULL == uthread) {
      sleep(0);
    } else {
      uthread->state = THREAD_RUNNING;
      current_thread = uthread;
      thread_switch(&main_thread->context, &uthread->context);
    }
    if (uthread->state == THREAD_STOP) {
      thread_destroy(uthread);
    }
  }
}
```

_uthread_entry

```
void _uthread_entry(struct uthread *tcb, void (*thread_func)(void *),
                    void *arg) {
  tcb->state = THREAD_RUNNING;
  thread_func(arg);
  tcb->state = THREAD_STOP;
  thread_switch(&tcb->context, &main_thread->context);
}
```

Outline

- Solution
- Challenge

Challenge:float context

- thread_swtich里只保存了整数寄存器的上下文。如何拓展到浮点数?
- 看看ucontext.h是怎么做的

```
struct _libc_fpstate
{
  /* 64-bit FXSAVE format.  */
  __uint16_t    __ctx(cwd);
  __uint16_t    __ctx(swd);
  __uint16_t    __ctx(ftw);
  __uint16_t    __ctx(fop);
  __uint64_t    __ctx(rip);
  __uint64_t    __ctx(rdp);
  __uint32_t    __ctx(mxcsr);
  __uint32_t    __ctx(mxcr_mask);
  struct _libc_fpxreg _st[8];
  struct _libc_xmmreg _xmm[16];
  __uint32_t    __glibc_reserved1[24];
};
```

Challenge: Timer Interupt

- 需要添加一个Timer Interupt信号的处理函数
  - Setitimer

```
struct sigaction sched_handler = {
    .sa_handler = &one_schedule, /* set signal handler to call};
sigaction(SIGPROF, &sched_handler, NULL);
time_quantum.it_value.tv_usec = 10000; // 50ms
time_quantum.it_interval.tv_usec = 10000;
setitimer(ITIMER_PROF, &time_quantum, NULL);
```

- 如果一个函数在进行切换时，产生了一次Timer Interupt?

Challenge: Timer Interupt

- 需要一个Spinlock来保护数据
  - 不能使用mutex之类的原语，而要做忙等待.
- C11/gnu c提供atomic函数，可以来实现spinlock

```
atomic_flag _spinlock = ATOMIC_FLAG_INIT;

void spin_lock(atomic_flag *lock) {
  while (atomic_flag_test_and_set_explicit(lock, memory_order_acquire))
    ;
}

void spin_unlock(atomic_flag *lock) {
  atomic_flag_clear_explicit(lock, memory_order_release);
}
```

Challenge: Timer Interupt

- 操作线程相关元数据的地方都要上锁

```
struct uthread *uthread_create(void (*func)(void *), void *arg,const char* thread_name) {
        // …
        spin_lock(&_spinlock);
        push_back(&schedule_thread, &uthread->list);
        spin_unlock(&_spinlock);
        // …
```

```
void schedule() {
    struct uthread *uthread = NULL;

  while (1) {
      spin_lock(&_spinlock);
      uthread = get_thread();
      // check uthread
      uthread->state = THREAD_RUNNING;
      current_thread = uthread;
      spin_unlock(&_spinlock);
      thread_switch(&main_thread->context, &uthread->context);
    }
}
```

Challenge: Timer Interupt

- 还需要关闭signal的响应

```
struct uthread *uthread_create(void (*func)(void *), void *arg,const char* thread_name) {
        sigemptyset(&mask);
        sigaddset(&mask, SIGPROF);
        sigprocmask(SIG_BLOCK, &mask, NULL);
        spin_lock(&_spinlock);
        push_back(&schedule_thread, &uthread->list);
        spin_unlock(&_spinlock);
        sigprocmask(SIG_UNBLOCK, &mask, NULL);

        // …
```

Challenge: Timer Interupt

- 类似yield的步骤

```
static void one_schedule() {
  spin_lock(&_spinlock);
  current_thread->state = THREAD_SUSPENDED;
  push_back(&schedule_thread, &current_thread->list);
  spin_unlock(&_spinlock);
  thread_switch(&current_thread->context, &main_thread->context);
}
```

- 但并不是时间片轮转
  - 时钟中断不是按一个用户态程序执行的时间片算的，而是按整个程序算的.

Challenge : m-n

- 我们只实现了一个1 kthread :n uthread的模型，如何拓展成m : n的模型呢?

- 可以搭建一个m个pthread的线程池，每个线程池有独立的调度函数。
- 原先的current_thread需要改为一个数组。每个pthread都有对应的tid，通过tid获取查询数组，获取当前pthread对应的上下文。
  - 例如，可以写一个hash表

Challenge: mutex

- 以Mutex为例

```
struct umutex {
  struct uthread *owner;
  atomic_flag lock;
  struct list_head wait_list;
};
```

```
int umutex_lock(struct umutex *mutex)
{

    /* Use "test-and-set" atomic operation to acquire the mutex lock */
    while (atomic_flag_test_and_set_explicit(&mutex->lock, __ATOMIC_ACQUIRE)) {
      push_back(&mutex->wait_list, &current_thread->list);
      thread_switch(&current_thread->context, &main_thread->context);
    }
    mutex->owner = current_thread;

    return 0;
}
```

Challenge: mutex

- 以Mutex为例

```
struct umutex {
  struct uthread *owner;
  atomic_flag lock;
  struct list_head wait_list;
};
```

```
int umutex_unlock(struct umutex *mutex)
{
    struct list_head *next_node = NULL;
    struct uthread * cur = NULL;
    next_node = pop_front(&mutex->wait_list);
    atomic_flag_clear_explicit(&mutex->lock,__ATOMIC_RELEASE);
    if (next_node) {
      cur = GET_TCB(next_node);
      spin_lock(&_spinlock);
      // cur->state = THREAD_RUNNING;
      push_back(&schedule_thread, &cur->list);
      spin_unlock(&_spinlock);
    }
    mutex->owner = NULL;
    return 0;
}
```

参考

- Fiber:sysprog21/fiber: A User Space Threading Library (github.com)