Section 4

Threading and Project 2

(Many slides taken from Sec. 3 Winter 2006)

Project 0 stats

- Mean: 3.2 / 4
- Median: 3.35 / 4
- Stdev: 0.69

Project 2 will be up today

- Start EARLY!
  - It's long
  - Read the assignment carefully
  - Read it again
- Use the same groups as for project 1
  - If you want to change, tell me soon!

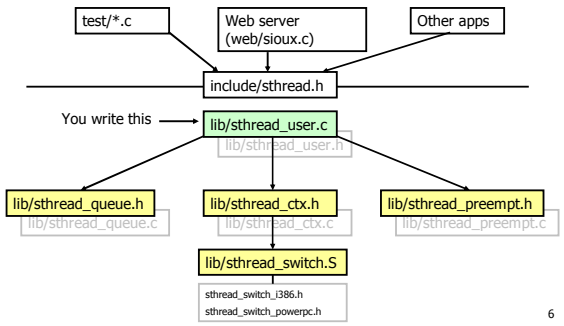
Project 2

- You have to:
  - Part a:
    - Implement a user thread library
    - Implement synchronization primitives
    - Solve a synchronization problem
  - Part b:
    - Add Preemption
    - Implement a multithreaded web server
    - Get some results and write a (small) report
- Part a and b due separately (TBD)
  - Whole project 2 will be due on Nov 10

Simplethreads

- We give you:
  - Skeleton functions for thread interface
  - Machine-specific code
    - Support for creating new stacks
    - Support for saving regs/switching stacks
  - A generic queue
    - When do you need one?
  - Very simple test programs
    - You should write more, and include them in the turnin
  - Singlethreaded web server

Simplethreads Code Structure



Thread Operations

- What functions do we need?
- What should the TCB look like?

Thread Operations

- `void sthread_init()`
  - Initialize the whole system
- `sthread_t sthread_create(func start_func, void *arg)`
  - Create a new thread and make it runnable
- `void sthread_yield()`
  - Give up the CPU
- `void sthread_exit(void *ret)`
  - Exit current thread
- Structure of the TCB:

```
struct _thread {
    sthread_ctx_t *saved_ctx;
    ……..

}
```


Sample multithreaded program

```
int main(int argc, char **argv) {
   int i;

   sthread_init();
   for(i=0; i<3; i++)
      if (sthread_create(thread_start, (void*)i) == NULL) {
          printf("sthread_create failed\n");
          exit(1);
      }

   sthread_yield();
   printf("back in main\n");
   return 0;
}

void *thread_start(void *arg) {
  printf("In thread_start, arg = %d\n", (int)arg);
  return 0;
}
```

- **Output? (assume no preemption)**

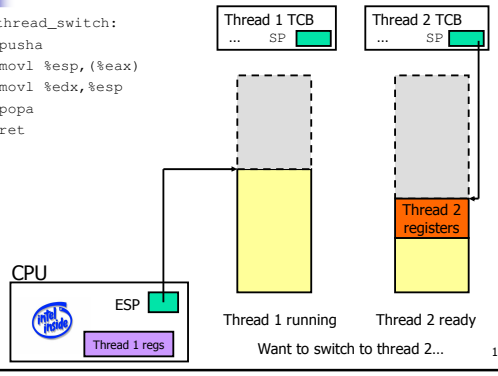
Managing Contexts (given)

- Thread context = thread stack + stack pointer
- sthread_new_ctx(func_to_run)
  - creates a new thread context that can be switched to
- sthread_free_ctx(some_old_ctx)
  - Deletes the supplied context

➢ sthread_switch(oldctx, newctx)
  - Puts current context into oldctx
  - Takes newctx and makes it current

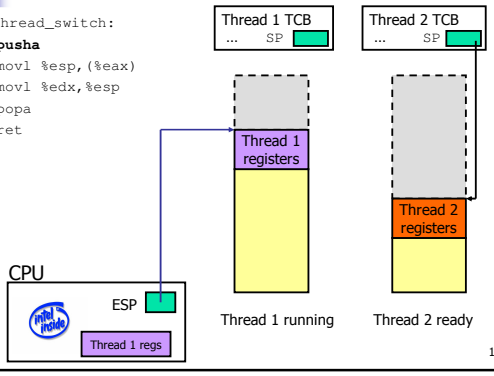
How sthread_switch works

```
Xsthread_switch:
   pusha
   movl %esp,(%eax)
   movl %edx,%esp
   popa
   ret
```



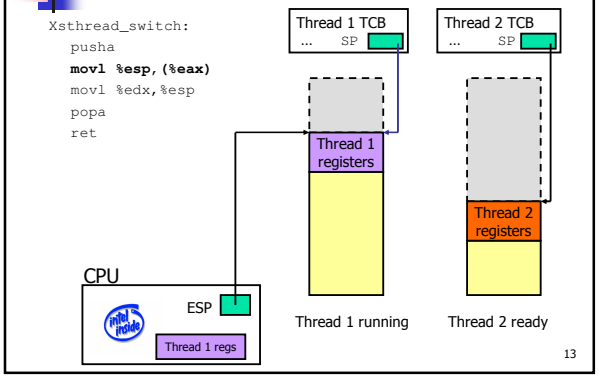
Push old context

```
Xsthread_switch:
   pusha
   movl %esp,(%eax)
   movl %edx,%esp
   popa
   ret
```



Save old stack pointer

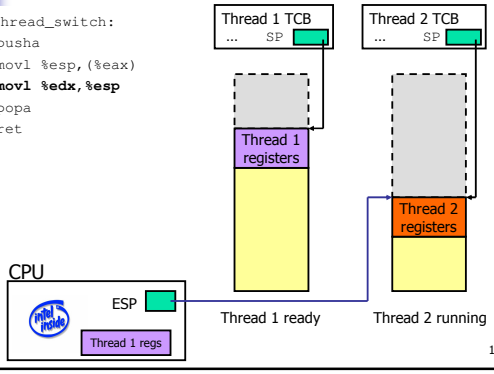
```
Xsthread_switch:
   pusha
   movl %esp,(%eax)
   movl %edx,%esp
   popa
   ret
```



13

Change stack pointers

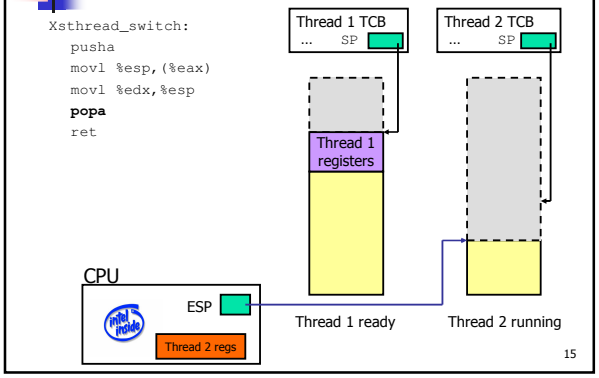
```
Xsthread_switch:
   pusha
   movl %esp,(%eax)
   movl %edx,%esp
   popa
   ret
```



14

Pop off new context

```
Xsthread_switch:
   pusha
   movl %esp,(%eax)
   movl %edx,%esp
   popa
   ret
```

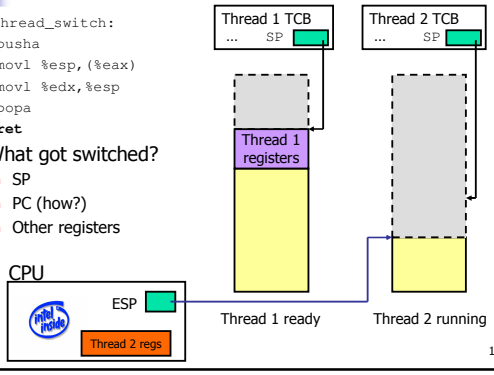


15

Done; return

```
Xsthread_switch:
   pusha
   movl %esp,(%eax)
   movl %edx,%esp
   popa
   ret
```

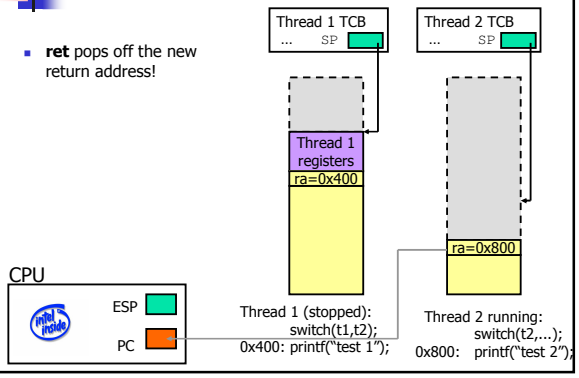
- What got switched?
  - SP
  - PC (how?)
  - Other registers



16

Adjusting the PC

- **ret** pops off the new return address!



Synchronization primitives: Mutex

- sthread_mutex_t sthread_mutex_init()
- void sthread_mutex_free(sthread_mutex_t lock)
- void sthread_mutex_lock(sthread_mutex_t lock)
  - Returned thread is guaranteed to acquire lock
- void sthread_mutex_unlock(sthread_mutex_t lock)
  - Release lock
- See sthread.h

18

Synchronization primitives: Condition variables

- sthread_cond_t sthread_cond_init()
- void sthread_cond_free(sthread_cond_t cond)
- void sthread_cond_signal(sthread_cond_t cond)
  - Wake-up one waiting thread, if any
- void sthread_cond_broadcast(sthread_cond_t cond)
  - Wake-up all waiting threads, if any
- void sthread_cond_wait(sthread_cond_t cond, sthread_mutex_t lock)
  - Wait for given condition variable
  - Returning thread is guaranteed to hold the lock

19

Things to think about

- How do you create a thread?
  - How do you pass arguments to the thread's start function?
  - (sthread_new_ctx() doesn't call function w/ arguments)
- How do you deal with the initial (main) thread?
- When and how do you reclaim resources for a terminated thread?
  - Can a thread free its stack itself?
- Where does sthread_switch return?
- Who and when should call sthread_switch?
- How do you block a thread?
- What should be in struct _sthread_mutex|cond?

20

Sthread is similar to pthread

- Pthread (POSIX threads) is a preemptive, kernel-level thread library
- You can compare your implementation against pthreads
  - ./configure --with-pthreads

21