Reminders

- Start project 2!
  - It's long
  - Read the assignment carefully
  - Read it again
- Make sure your groups are correct

- Today:
  - Project 2 intro
  - CVS

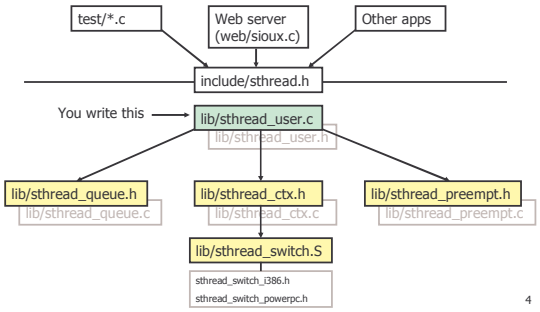
Project 2

- 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 due: in two weeks
  - Part b due: 9 days after part a

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
  - Singlethreaded web server

Simplethreads Code Structure



Thread Operations

- Which ones do we need?

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

- What about the **TCB**?

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

}
```

- **Others?**

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)**

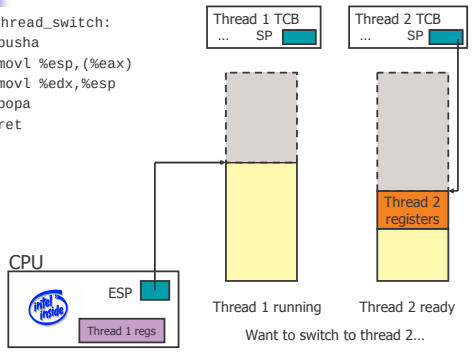
Managing Contexts (given)

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

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

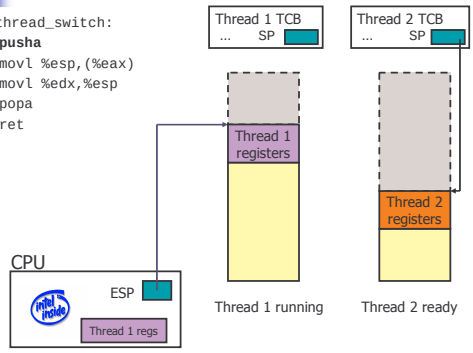
How sthread_switch works

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



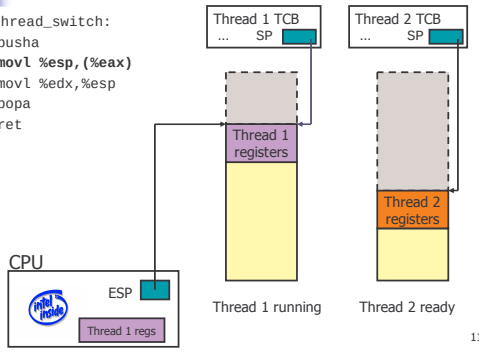
Push old context

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



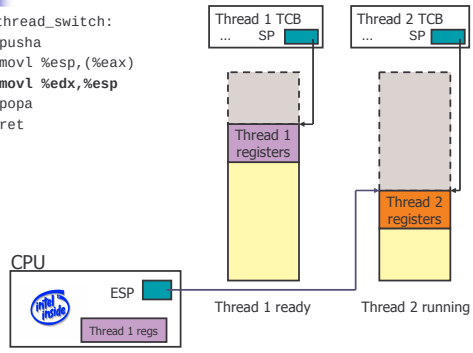
Save old stack pointer

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



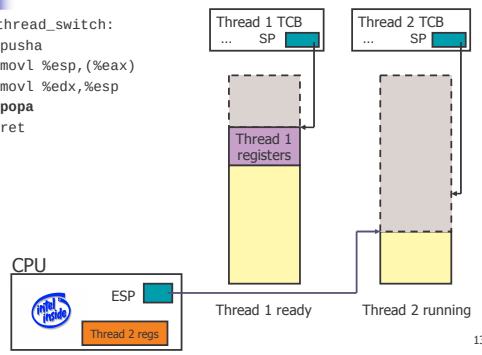
Change stack pointers

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



Pop off new context

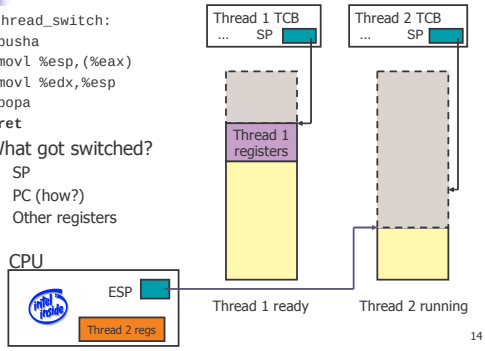
```
Xsthread_switch:
   pusha
   movl %esp,(%eax)
   movl %edx,%esp
   popa
   ret
```



Done; return

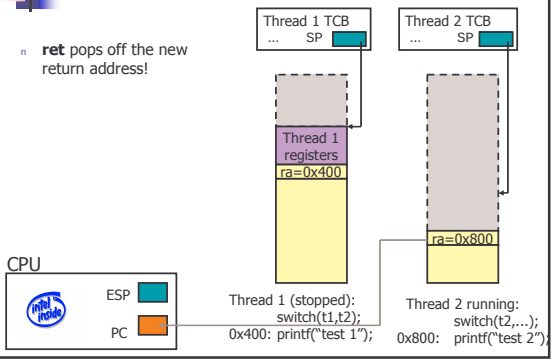
```
Xsthread_switch:
   pusha
   movl %esp,(%eax)
   movl %edx,%esp
   popa
   ret
```

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



Adjusting the PC

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



Things to think about

- Who will call sthread_switch?
- Where does sthread_switch return?
- How do we delete a thread?
  - Can a thread free its stack itself?
- Starting up a thread
  - When you create a new stack with sthread_new_ctx(), you initialize it to run some function foo
  - sthread_new_ctx doesn't pass parameters to foo
  - But in sthread_create, you give a function *and* an arg!
  - **Bottom line:** how do you pass arguments to a function with no arguments?

Preemption

- Initially, you will build a non-preemptive thread library.
- In part 4, you add preemption
  - Much more realistic model
- We give you:
  1. Timer interrupts
  2. Primitives to enable/disable interrupts
  3. atomic_test_and_set/atomic_clear for synchronization
- You must:
  - Do something on each timer interrupt. (What?)
  - Synchronize all of your code using 2 and 3 above.
- More on this later…

Last note

- You can compare your implementation against pthreads (which is preemptive kernel-threads).
  - ./configure --with-pthreads

What is CVS

- Version control system for source files
- Multiple users can work on the same file simultaneously

19

Why use CVS

- The other way:
  - Keep every version of code, all with different names:
    - Project2good
    - Project2_10_13_04
    - Project2working
    - Project2_Feb_2_alex
    - Project2_old
  - Send emails back and forth with new changes
  - Merge different versions by hand
- The CVS way:
  - One version, saved in the CVS repository
  - Multiple people can work on the same file concurrently
  - CVS merges the edited versions automatically as you put them back in the repository

20

Setting up CVS

- Set up CVS root
  - `setenv CVSROOT /cse451/groupa/cvs`
  - (bash) `export CVSROOT=/cse451/groupa/cvs`
- Initialize a repository (only one person per group)
  - `cd /cse451/groupa`
  - `mkdir cvs`
  - `cvs init`

21

Setting up CVS (2)

- Add the simplethreads distribution
  - `tar xvfz simplethreads-1.20.tar.gz`
  - `cd simplethreads-1.20`
  - `cvs import –m "initial code" simplethreads SIMPLETHREADS SIMPLETHREADS_1_20`
  - `cd ..`
  - `rm –fr simplethreads-1.20`

22

CVS Commands

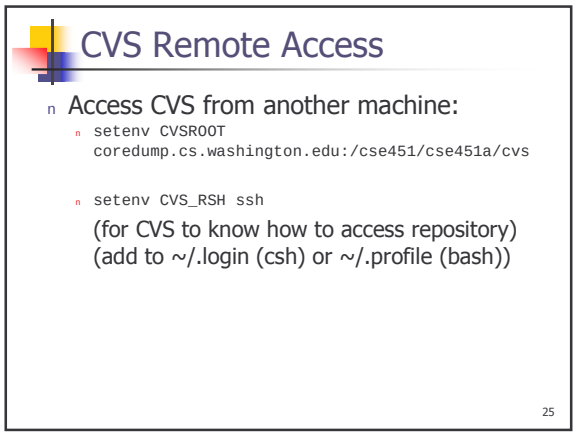
- Check out a project to your home directory:
  - `cd ~`
  - `cvs checkout simplethreads`
- Merge in new changes from repository (update):
  - `cvs update [files…]`
- Save your edited files into the repository:
  - `cvs commit –m "fixed annoying bugs" [files…]`

23

CVS Commands 2

- Add a new file to the repository
  - `cvs add [files…]`
- Check status of a file
  - `cvs status file.c`
- Check differences between your file and one in the repository:
  - `cvs diff file.c`
  - `cvs diff –r 1.1 file.c` (specifies version)
- View log of changes for a file
  - `cvs log file.c`
- More info:
  - http://www.cvshome.org
  - cvs --help-commands

24


CVS Remote Access
Access CVS from another machine:
setenv CVSROOT coredump.cs.washington.edu:/cse451/cse451a/cvs
setenv CVS_RSH ssh
(for CVS to know how to access repository)
(add to ~/.login (csh) or ~/.profile (bash))
25