CSE 451: Operating Systems

Section 6
Project 2b; Midterm Review

Project 2b

* Parts 4, 5 and 6 of project 2
* Due at 11:59pm, Wednesday November 17

Part 4: web server

* web/sioux.c – singlethreaded web server
  * Read in command line args, run the web server loop

Part 4: web server

* web/sioux_run.c – the web server loop
  * Open a socket to listen for connections (listen(2))
  * Wait for a connection (accept(2))
  * Handle connection:
    * Parse the HTTP request
    * Find and read the requested file
    * Send the file back
    * Close the connection
Thread pools

![Thread pool diagram](http://en.wikipedia.org/wiki/Thread_pool_pattern)


What you need to do

- Make the web server multithreaded
- Create a thread pool
  - Suggestion: create separate thread_pool.h, thread_pool.c
- Wait for a connection
- Find an available thread to handle the request
  - Request waits (where?) if all threads busy
- Once the request is handed to a thread, it uses the same processing code as before

Hints

- Each connection is identified by a socket file descriptor returned by `accept(2)`
- File descriptor (fd) is just an int
- Threads should sleep while waiting for a new connection
- Condition variables are perfect for this

Hints

- Don’t forget to protect any global variables
- Use mutexes and CVs from part 2
- Develop and test with pthreads initially
- Use only the pthread.h interface
- Mostly modify sioux_run.c, and your own files
Part 5: preemption

What we give you:
- Timer interrupts
- Function to turn interrupts on and off
- Synchronization primitives
  - `atomic_test_and_set`, `atomic_clear`
- x86 architecture only

What you have to do:
- Add code that will run every time a timer interrupt is generated
- Add synchronization to your part 1 and part 2 implementations so that everything works with preemptive thread scheduling
- Can be done independently of part 4

```c
/* Start preemption - func will be called
   * every period microseconds
   */
void sthread_preemption_init
  (sthread_ctx_start_func_t func,
   int period);

/* Turns interrupts on (LOW) or off (HIGH)
   * Returns the last state of the
   * interrupts
   */
int splx(int splval);
```

```c
/* atomic_test_and_set - using the native
   * compare and exchange on the Intel x86.
   * Example usage:
   *   lock_t lock;
   *   while(atomic_test_and_set(&lock))
   *     {} // spin
   *   __critical_section__
   *   atomic_clear(&lock);
   */
int atomic_test_and_set(lock_t *l);
void atomic_clear(lock_t *l);
```
Signals

* Used to notify processes of events, asynchronously
* Every process has a *signal handler* table
* When a signal is sent to a process, OS interrupts that process and calls the handler registered for that signal

Signal manipulation

* A process can:
  * Override the default signal handlers using `sigaction(2)`
  * Block / unblock signals with `sigprocmask(2)`
  * Send a signal via `kill(2)`
* Signals:
  * SIGINT (CTRL-C), SIGQUIT (CTRL-\), SIGKILL, SIGFPE, SIGALRM, SIGSEGV...

What you need to do

* Add a call to `sthread_preemption_init()` as the last line in your `sthread_user_init()` function
  * `sthread_preemption_init()` takes a pointer to a function that will be called on each timer interrupt
  * This function should cause thread scheduler to switch to a different thread!

What you need to do

* Add synchronization to *critical sections* in thread management routines
  * Think: what would happen if the code was interrupted at this point?
    * Would it resume later with no problems?
    * Could the interrupting code mess with any variables that this code is currently using?
  * Don’t have to worry about simplethreads code that you didn’t write (i.e. `sthread_switch`): already done for you
Interrupt disabling

Non-thread-safe

/* returns next thread */

sthread_t

sthread_user_next() {
    sthread_t next;
    next = sthread_dequeue(ready_q);
    if (next == NULL)
        exit(0);
    return next;
}

Thread-safe

/* returns next thread on the ready queue */

sthread_t

sthread_user_next() {
    sthread_t next;
    int old = splx(HIGH);
    next = sthread_dequeue(ready_q);
    splx(old);
    if (next == NULL)
        exit(0);
    return next;
}

Atomic locking

* So what is atomic_test_and_set() for?
    * Primarily to implement higher-level synchronization primitives (mutexes, CVs)

* One way to think about preemption-safe thread library:
    * Disable/enable interrupts in “library” context
    * Use atomic locking in “application” context

Race conditions and testing

* How can you test your preemption code?
* How can you know that you’ve found all of the critical sections?

Part 6: report

* Covers all parts of project 2
* Discuss your design decisions

* Performance evaluation:
    * Measure throughput and response time of your web server using web benchmarking tool
    * Vary the number of threads and number of “clients”
    * Present results in graphical form
    * Explain results: expected or not?
Project 2 questions?

Midterm

* Concepts to know:

The kernel

* Kernel mode vs user mode
  * How these modes differ conceptually and from the CPU's point of view
  * How we switch between the two
* Interrupts

System calls

* What they are
* What they do
* How they do it
* What hardware is involved
* Who uses them and when
Processes and threads

- Kernel processes, kernel threads, and user threads
- How these differ from one another
- Context switching
- Process and thread states
- fork, exec, wait

Scheduling

- Different scheduling algorithms and their tradeoffs
- Average response time, various “laws”
- Starvation
- Cooperative vs. preemptive scheduling

Synchronization

- Critical sections
- Locks and atomic instructions
- Mutexes, semaphores, and condition variables
- Monitors
- Ways to detect / avoid deadlock

Memory management

- Paging
- Segmentation
- Address translation
- Page tables
- Page replacement
Tips

* Focus on lecture slides
* Review textbook, section slides and project writeups to emphasize key concepts and fill in gaps
* On Friday:
  * Arrive early
  * Focus on key points
  * Work quickly; finish easy problems first