Assignment 2 FAQ

Zhe Wang

Oct, 11th 2006

Two *_entry() in Entry.S

- Kernel_entry()
  - Used to switch between user space process and the kernel
  - In the future, if we have syscall like open(), then only kernel_entry will be used.
- Scheduler_entry()
  - Used to switch between kernel threads/(process that is in kernel)
- What syscall yield() goes through:
  - First goes through kernel_entry to switch from user -> kernel
  - Then goes through scheduler_entry() to switch between processes/threads.

Two yield() and exit()

- There are two copies of the same functions!
  - Syslib.c:

```
void yield(void) {
    SYSCALL(SYSCALL_YIELD);
}
```

  - Scheduler.c (To be implemented by you!)
- One is called within the user space by the processes when they want to enter kernel. (Eg: process1.c)
- The other is linked with the kernel, it is called directly by threads! (Eg: th1.c)
  - Check your Makefile

Why two stacks for process

- User stack
  - Used when the process is running in the user space
- Kernel stack
  - Used after the process enter the kernel
- In practice, for this assignment, only using one kernel_stack will still work. (why?)

Synchronization

- Lock implementation
  - You do not have to implement TAS (or cli) kind of mutual exclusion. (For this assignment, it is non-preemptive)
  - Why do you need to use “While”?
    - Think: what if two threads are blocked waiting for the same lock?
    - Or if some other process acquire the lock even before the unblocked process got a chance to run.

How to handle PCB

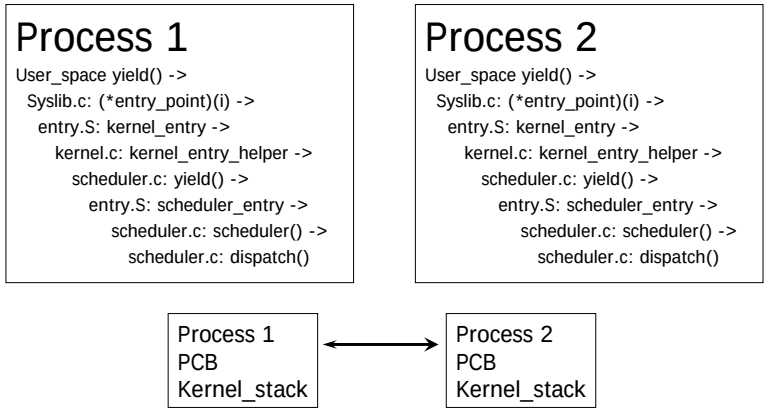
- PCBs are statically allocated in kernel in this assignment (a static array)
- We do not ask you to recycle PCBs in this assignment (to make your life easier). Same applies to stack allocation.
- When process exit(), you can just remove its PCB from the ready queue and “throw” it away.

Context switch!

- Flow of control:

User_space yield() ->
Syslib.c: (*entry_point)(i) ->
entry.S: kernel_entry ->
kernel.c: kernel_entry_helper ->
scheduler.c: yield() ->
entry.S: scheduler_entry ->
scheduler.c: scheduler() ->
scheduler.c: dispatch()
Then WHAT!?

Parallel universe!



First time?

- When the process/thread is switched to for the first time:
  - Kernel_stack does not look like last slide.
  - Need special treatment to “start” the process

Misc

- Inline assembly:

  "jmp *%0"::"q“(addr)

  (If you use “jmp %0”, it will be treated as relative jmp)
- In_kernel (inside pcb_t from last precept)
  - It is a boolean: whether it is process or thread
  - So better name it as: is_thread
- Ignore the jmp code in entry.S (you should replace them with your own code)

Misc (cont.)

- Saving registers onto stack or PCB
- PUSHAD (intel syntax) => PUSHAL (AT&T syntax)
  - Check using AS: 80386 dependent features, at:

  http://www.gnu.org/software/binutils/manual/gas-2.9.1/html_chapter/as_16.html#SEC196
- To access current_running in entry.S: Just refer it by name (compiler will handle it)