Operating Systems Design (CS 423)

Elsa L Gunter
2112 SC, UIUC
http://www.cs.illinois.edu/class/cs423/

Based on slides by Roy Campbell, Sam King, and Andrew S Tanenbaum

swapcontext

- Unix support for switching threads
- Like Linux context switching function
  - Hides many of the details
  - Moves context running on CPU to mem add or context at mem add to CPU or combination
  - What does `swapcontext` store?
  - We will talk about how to use `swapcontext` next time

swapcontext

- Unix support for switching threads
- Like Linux context switching function
  - Hides many of the details
  - Moves context running on CPU to mem add or context at mem add to CPU or combination
  - What does `swapcontext` store?
    Stores registers (including PC), stack pointer
  - We will talk about how to use `swapcontext` next time

Example of thread switching

- Thread 1

```
print "start thread 1";
yield();
print "end thread 1";
```

- Thread 2

```
print "start thread 2";
yield();
print "end thread 2";
```

- Yield

```
print "start yield (thread %d)";
switch to next thread (swapcontext);
print "end yield (current thread %d);
```


Thread Output

- Thread 1

```
start thread 1
start yield (thread 1)

end yield (thread 1)
end thread 1
```

- Thread 2

```
start thread 2
start yield (thread 2)

end yield (thread 2)
end thread 2
```


Thread switching in Linux

- PCB == TCB conceptually
- Thread switching is the same as Process switching except that the address space stays the same
- To make switching work in Linux, any thread that switches **must** do so through same one switching function

Thread switching in Linux

- When executing in kernel, executing on behalf of a thread
- Kernel stack key to this abstraction on x86
  - Contains local state (stack) and process struct
  - E.g., current pointer (recall it from MP1)
- Other architectures use different techniques

```
switch_to(task_struct *prev_p,
          task_struct *next_p)
```


Swapcontext()

```
swapcontext(ucontext_t *oucp,
            ucontext_t *uco);
```

- Saves the current contexts, takes the given context and makes it the current context
- oucp pointer to data structure to store current context
- Ucp pointer to new context to install

Example

- Void schedule(){

```
    struct TCB *next = NULL
    struct TCB *prev = NULL
// Code here: Figure out next and prev
    current = next
    swapcontext(prev->ctx, next->ctx);
    readyQueue.push(prev);
}
```

What is wrong with this?

x86 assembly overview (32 bit)

- Movl src, dst
- Pushl reg
- Pushfl – push eflags
- Jump imm
- Popl reg
- Popfl – pop eflags
- Eax – gp reg
- Ebx – gp reg
- Ecx – gp reg
- Edx – gp reg
- …
- Ebp – frame pointer
- Eip – PC
- Esp – stack pointer

Thread switching in Linux

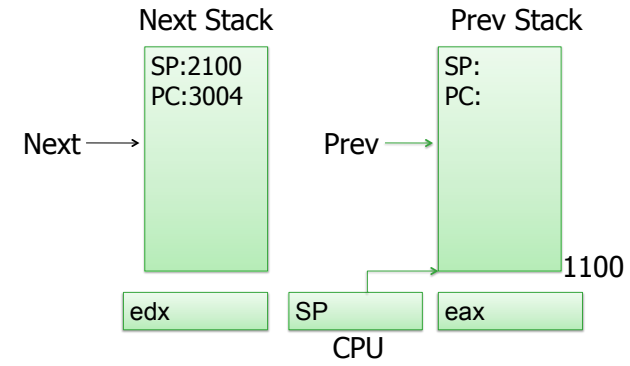
```
// save values of prev and next, prev next
// are PCB (similar to TCB)
Movl prev, %eax
Movl next, %edx
// save context of eflags and ebp
Pushfl Pushl %ebp
// save context of stack pointer in prev PCB
Movl %esp, 484(%eax)
// switch to next processes stack
Movl 484(%edx), %esp
```


Thread switching in Linux

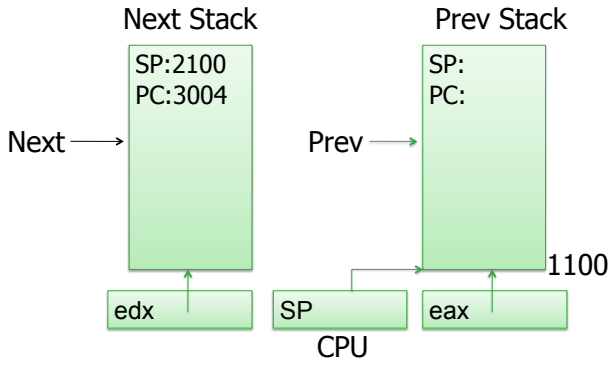
```
// save add of $1f in pc element of prev PCB
Movl $1f, 480(%eax)
// push value of pc for next PCB on stack
Pushl 480(%edx)
// *jump* (not call) to C function __switch_to
Jmp __switch_to
// when __switch_to returns, pops PC off of stack
// as return address. This PC was $1f
1:  popl %ebp
popfl
```

Thread switching in Linux

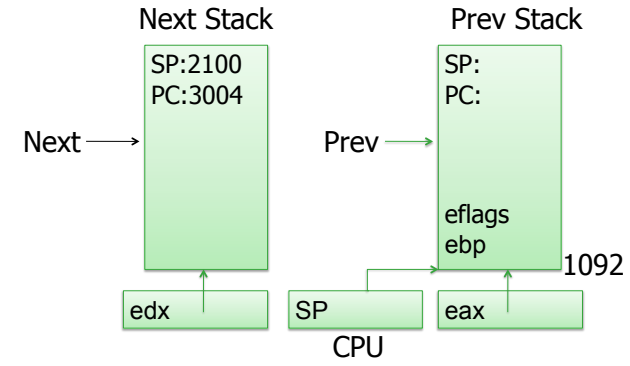
- Draw the stack for each step of the switch function shown here



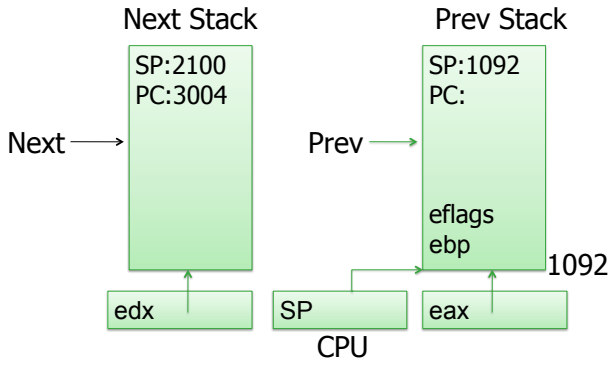
Movl prev, %eax Movl next, %edx



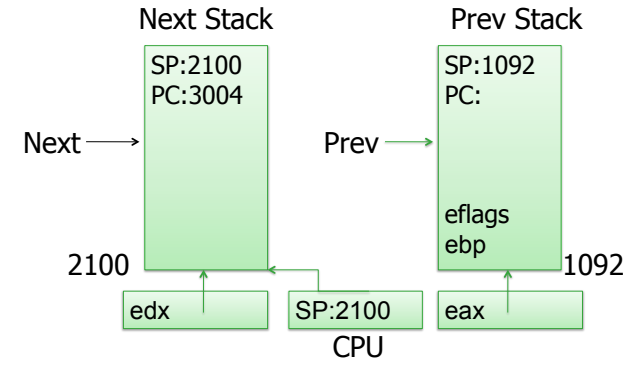
Pushfl Pushl %ebp



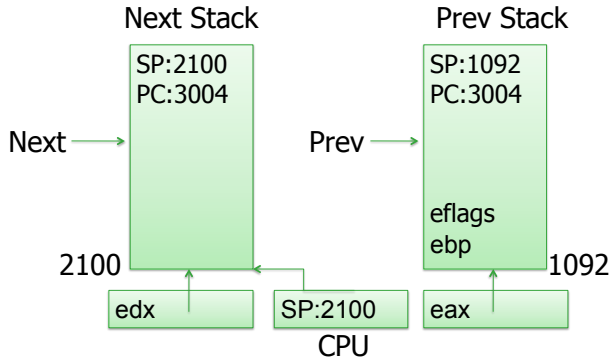
Movl %esp, 484(%eax)



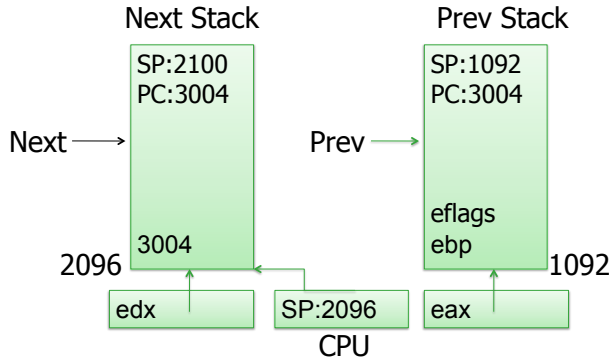
Movl 484(%edx), %esp



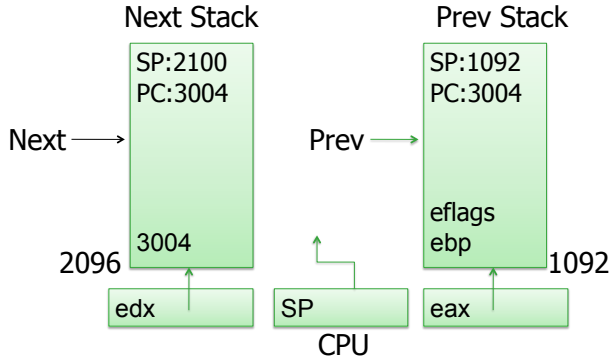
Movl $1f, 480(%eax)



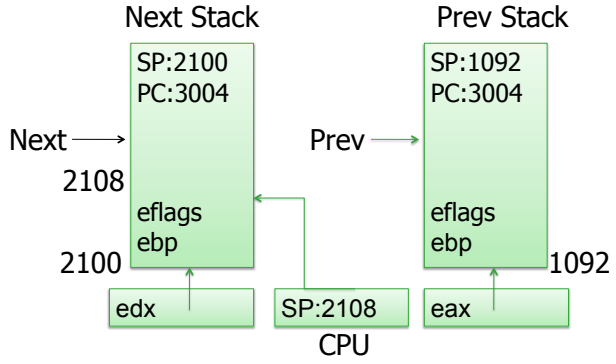
Pushl 480(%edx)



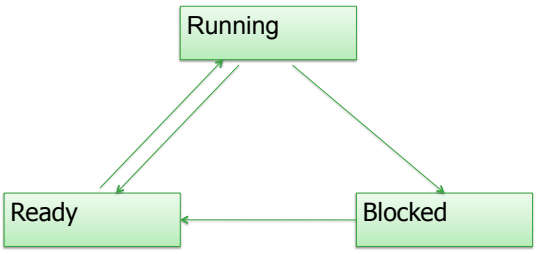
Jmp __switch_to



popl %ebp popfl



Thread States



Creating a new thread

- Overall: create state for thread and add it to the ready queue
- When saving a thread to its thread control block, we remembered its current state
- We can **construct** the state of new thread as if it had been running and got switched out

Creating a new thread

- Steps:
  - Allocate and initialize new thread control block
  - Allocate and init new stack
  - Add to ready queue
- From the scheduler's perspective, what is the different between a new and an old thread?
- What state does a new thread get init to?