Low-Level Thread Dispatching on the x86

- Dispatching
- Dispatching and Exception Handling
- RECAP: Exception Handling in the x86
- How to use Exception Handling mechanisms for Dispatching
- RECAP: The Thread Control Block (TCB) in class **Thread**
- Dispatching in 4 Steps: Start, Store, Locate, Load
- Creating a Thread
- The Execution Lifecycle of a Thread

Low-Level Thread Dispatching

Low-level Dispatch

```
Thread::dispatch_to(Thread * _thread);
```

Store state of current thread execution and load state of target thread onto CPU.

Dispatching in 3 steps:

1. Save state of current thread
2. Load state of new thread
3. Continue execution of new thread

Low-Level Thread Dispatching

Dispatching in 3 steps:

1. Save state of current thread
2. Load state of new thread
3. Continue execution of new thread

Problems:

- How do we save the state of the current thread?
- How to we load the state of the new thread?
- How do we continue execution of new thread?!

Low-Level Thread Dispatching

Problems:

- How do we save the state of the current thread?
- How to we load the state of the new state?
- How do we continue execution of new thread?!

Q: Where else do we have the same problem?

A: When we handle exceptions.

Solution: Handle Thread Dispatching (somewhat) in the same manner!

RECAP: Exception Handling

```
global _isr0
global _isr1
global _isr2

...

global _isr8

…

global _isr30
global _isr31
```

```
;  0: Divide By Zero Exception
_isr0:
    push byte 0     ; push dummy
    push byte 0     ; push interrupt no
    jmp isr_common_stub
```

```
;  8: Double Fault Exception (Error Code!)
_isr8:
    ; don't push dummy!
    push byte 8       ; push interrupt no
    jmp isr_common_stub
```

```
extern _dispatch_exception
; The common ISR stub.
isr_common_stub:
  pusha
  push ds
  push es
  push fs
  push gs
  mov eax, esp
  push eax
  mov eax, _dispatch_exc
  call eax
  pop eax
  pop gs
  pop fs
  pop es
  pop ds
  popa
  dd esp, 8 ; pop code and int no
  ret
```

```
/* This defines what the stack looks like when the exception/interrupt
   reaches the exception dispatcher. */
typedef struct regs
{
    unsigned int gs, fs, es, ds,      /* pushed in common stub */
    unsigned int edi, esi, ebp, esp,
             ebx, edx, ecx, eax;      /* pushed by pusha */
    unsigned int int_no, err_code;    /* pushed in _isrXX */
    unsigned int eip, cs, eflags;     /* pushed by processor */
} REGS;
```

RECAP: Saving and Loading State

```
global _isr0
global _isr1
global _isr2

...

global _isr8

…

global _isr30
global _isr31
```

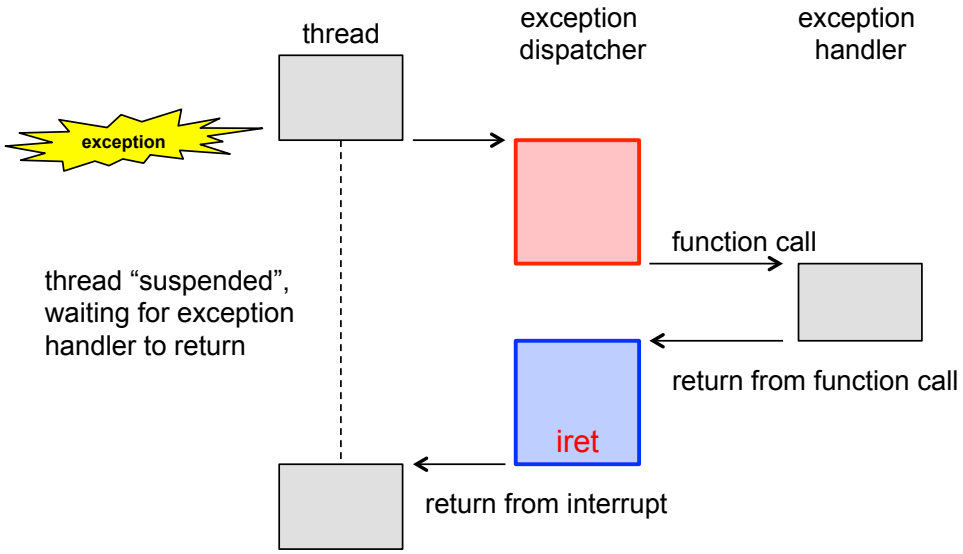
```
;  0: Divide By Zero Exception
_isr0:
    push byte 0     ; push dummy
    push byte 0     ; push interrupt no
    jmp isr_common_stub
```

```
extern _dispatch_exception
; The common ISR stub.
isr_common_stub:
  pusha
  push ds
  push es
  push fs
  push gs
  mov eax, esp
  push eax
  mov eax, _dispatch_exc
  call eax ; call exc dispatcher
  pop eax
  pop gs
  pop fs
  pop es
  pop ds
  popa
  add esp, 8 ; pop code and int no
  iret
```

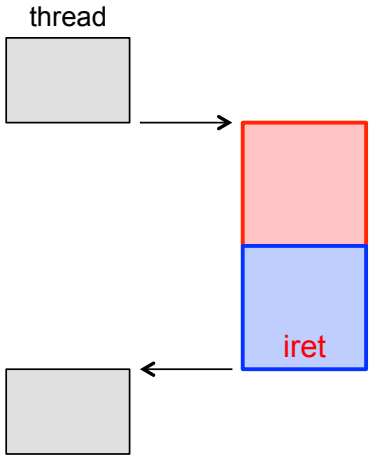
Here we save the state

Here we load the state

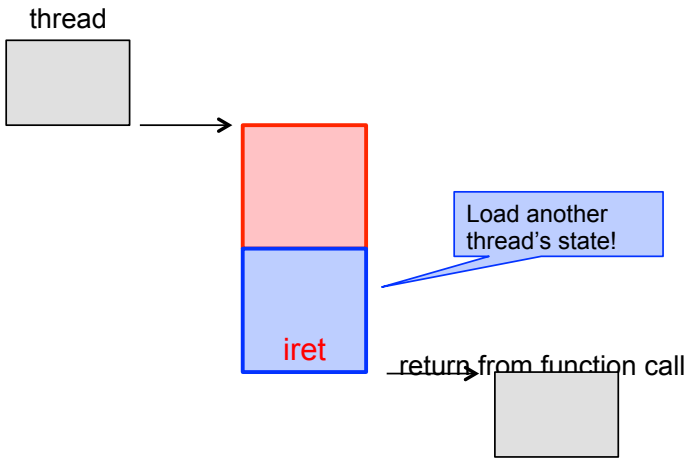
Saving and Restoring the State



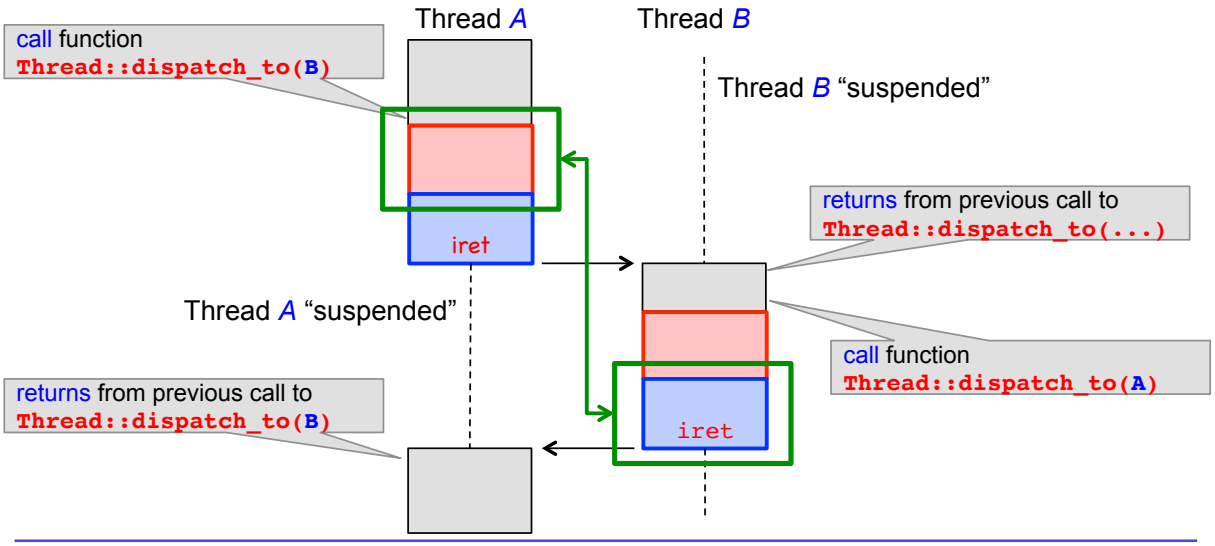
Saving and Restoring the State



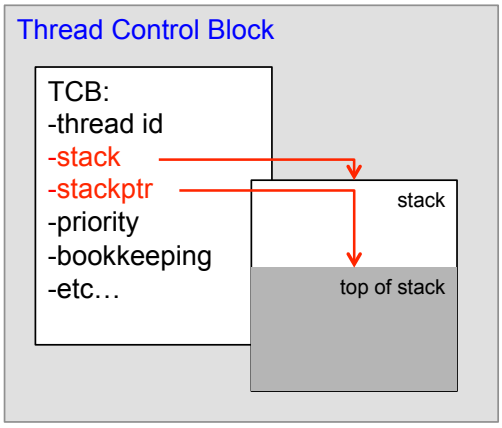
Saving and Restoring the State



Dispatching, Example



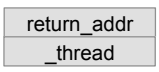
RECAP: Structure of a Thread




```
class Thread {

private:
  char      * stack_ptr;
  /* The current stack pointer for the thread.
     Keep it at offset 0, since the thread
     dispatcher relies on this  location! */
  int         thread_id;
  /* thread identifier.
     Assigned upon creation. */
  char      * stack;
  /* pointer to the stack of the thread.*/
  unsigned int stack_size;
  /* size of the stack (in byte) */
  int         priority;
  /* Maybe the scheduler wants to use priorities. */
  etc...
```

Dispatching Step-by-Step: 0. Start

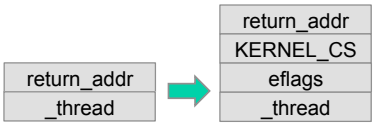
```
void Thread::dispatch_to(Thread * _thread) {
    threads_low_switch_to(_thread);
}
```



Dispatching Step-by-Step: 1. Fix Stack

```
void Thread::dispatch_to(Thread * _thread) {
    threads_low_switch_to(_thread);
}
```

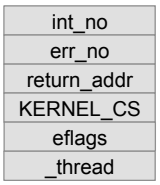
```
_threads_low_switch_to:
  ; STEP 1: fix stack
  push      eax             ; save eax
  mov       eax, [esp+4]    ; get return address
  mov       [esp-4], eax    ; move return addr
                            ; down 8 bytes from orig loc
  add       esp, 8          ; move stack ptr up
  pushfd                    ; put eflags where return address was
  mov       eax, [esp-4]    ; restore saved value of eax
  push      dword KERNEL_CS ; push cs selector
  sub       esp, 4          ; point stack ptr
                            ; at return address

  ; STEP 2: ...
  ...
```



Dispatching Step-by-Step: 2. Save State

```
void Thread::dispatch_to(Thread * _thread) {
    threads_low_switch_to(_thread);
}
```

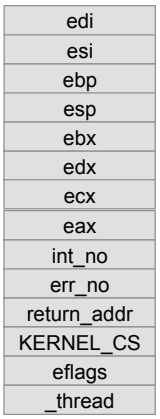
```
_threads_low_switch_to:
  ; STEP 1: fix stack
  ...
  ; STEP 2: save state
  ; Push fake error code and interrupt number
  push      dword 0
  push      dword 0
```



Dispatching Step-by-Step: 2. Save State

```
void Thread::dispatch_to(Thread * _thread) {
    threads_low_switch_to(_thread);
}
```

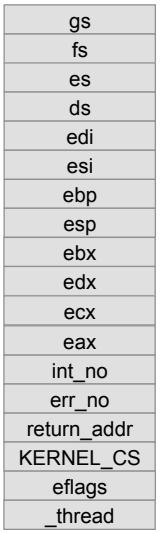
```
_threads_low_switch_to:
  ; STEP 1: fix stack
  ...
  ; STEP 2: save state
  ; Push fake error code and interrupt number
  push      dword 0
  push      dword 0
  ; Save general purpose registers.
  pushad
```



Dispatching Step-by-Step: 2. Save State

```
void Thread::dispatch_to(Thread * _thread) {
    threads_low_switch_to(_thread);
}
```

```
_threads_low_switch_to:
  ; STEP 1: fix stack
  ...

  ; STEP 2: save state
  ; Push fake error code and interrupt number
  push      dword 0
  push      dword 0
  ; Save general purpose registers.
  pushad
  push      ds
  push      es
  push      fs
  push      gs
```

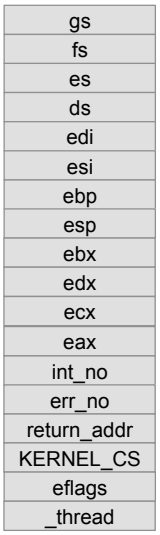


Dispatching Step-by-Step: 2. Save State

```
void Thread::dispatch_to(Thread * _th
    threads_low_switch_to(_thread);
}
```

```
_threads_low_switch_to:
  ; STEP 1: fix stack
  ...
  ; STEP 2: save state
  ; Push fake error code and interrupt
  push      dword 0
  push      dword 0
  ; Save general purpose registers.
  pushad
  push      ds
  push      es
  push      fs
  push      gs
  ; Save stack pointer in the thread control block
  ; (at offset 0).
  mov       eax, [_current_thread]
  mov       [eax+0], esp
```

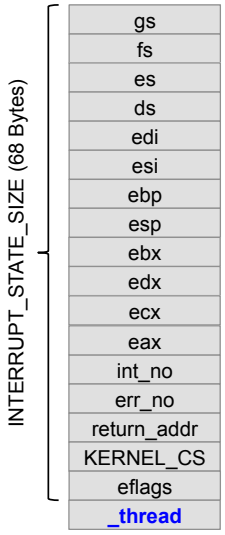
```
class Thread {

private:
  char      * stack_ptr;
  /* The current stack pointer for the
     thread.
     Keep it at offset 0, since the thread
     dispatcher relies on this  location! */
  int         thread_id;
  /* thread identifier.
     Assigned upon creation. */
  char      * stack;
  /* pointer to the stack of the thread.*/
  unsigned int stack_size;
  /* size of the stack (in byte) */
  int         priority;
  /* Maybe the scheduler wants to use
     priorities. */
  etc...
```



Dispatching Step-by-Step: 3. Locate State

```
void Thread::dispatch_to(Thread * _thread) {
    threads_low_switch_to(_thread);
}
```

```
_threads_low_switch_to:
  ; STEP 1: fix stack
  ...
  ; STEP 2: save state
  ...

  ; STEP 3: locate state of new thread
  mov       eax, dword [esp+INTERRUPT_STATE_SIZE]
```

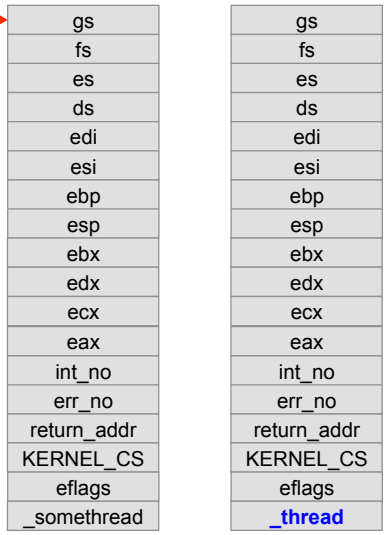


Dispatching Step-by-Step: 3. Locate State

```
void Thread::dispatch_to(Thread * _thread) {
    threads_low_switch_to(_thread);
}
```

```
_threads_low_switch_to:
  ; STEP 1: fix stack
  ...
  ; STEP 2: save state
  ...

  ; STEP 3: locate state of new thread
  mov       eax, dword [esp+INTERRUPT_STATE_SIZE]
  ; Make the new thread current, and switch to its stack.
  mov       [_current_thread], eax
  mov       esp, [eax+0]
```

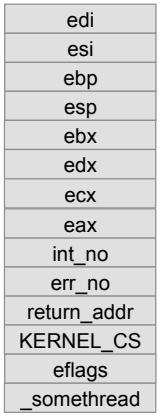
```
class Thread {                                  new thread

private:
  char      * stack_ptr;
  /* The current stack pointer for the thread.
     Keep it at offset 0, since the thread
     dispatcher relies on this  location! */
etc...
```



Dispatching Step-by-Step: 4. Load State

```
void Thread::dispatch_to(Thread * _thread) {
    threads_low_switch_to(_thread);
}
```

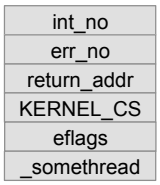
```
_threads_low_switch_to:
  ; STEP 1: fix stack
  ...
  ; STEP 2: save state
  ...
  ; STEP 3: locate state of new thread
  ...
  ; STEP 4: load state of new thread
  pop       gs
  pop       fs
  pop       es
  pop       ds
```



Dispatching Step-by-Step: 4. Load State

```
void Thread::dispatch_to(Thread * _thread) {
    threads_low_switch_to(_thread);
}
```

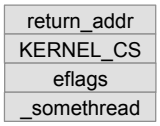
```
_threads_low_switch_to:
  ; STEP 1: fix stack
  ...
  ; STEP 2: save state
  ...
  ; STEP 3: locate state of new thread
  ...
  ; STEP 4: load state of new thread
  pop       gs
  pop       fs
  pop       es
  pop       ds
  popad
```



Dispatching Step-by-Step: 4. Load State

```
void Thread::dispatch_to(Thread * _thread) {
    threads_low_switch_to(_thread);
}
```

```
_threads_low_switch_to:
  ; STEP 1: fix stack
  ...
  ; STEP 2: save state
  ...
  ; STEP 3: locate state of new thread
  ...
  ; STEP 4: load state of new thread
  pop       gs
  pop       fs
  pop       es
  pop       ds
  popad
  add       esp, 8      ; skip int num and error code
```

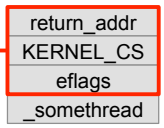


Dispatching Step-by-Step: 4. Load State

```
void Thread::dispatch_to(Thread * _thread) {
    threads_low_switch_to(_thread);
}
```

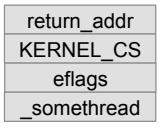
```
_threads_low_switch_to:
  ; STEP 1: fix stack
  ...
  ; STEP 2: save state
  ...
  ; STEP 3: locate state of new thread
  ...
  ; STEP 4: load state of new thread
  pop       gs
  pop       fs
  pop       es
  pop       ds
  popad
  add       esp, 8      ; skip int num and error code
  ; We'll return to the place where the thread was
  ; executing last.
  iret
```

```
void Thread::dispatch_to(Thread * _thread) {
    threads_low_switch_to(_thread);
}
```



Dispatching Step-by-Step: 4. Load State

```
void Thread::dispatch_to(Thread * _thread) {
    threads_low_switch_to(_thread);
}
```



Dispatching Step-by-Step: 4. Load State

```
void Thread::dispatch_to(Thread * _thread) {
    threads_low_switch_to(_thread);
}
```

_somethread

Dispatching Step-by-Step: 4. Load State

_somethread

Dispatching Step-by-Step: 4. Load State

Creating a Thread

```
void Thread::setup_context(Thread_Function _tfunction){
    /* Address of thread shutdown function */
    push((unsigned long) &tshutdown);
}
```

```
inline void Thread::push(unsigned long _val) {
    stack_ptr -= 4;
    *((unsigned long *) stack_ptr) = _val;
}
```

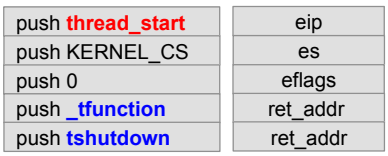
push **tshutdown** | ret_addr

Creating a Thread

```
void Thread::setup_context(Thread_Function _tfunction){
    /* Address of thread shutdown function */
    push((unsigned long) &tshutdown);
    /* Push the address of the thread function. */
    push((unsigned long) _tfunction);




}
```

Creating a Thread

```
void Thread::setup_context(Thread_Function _tfunction){
    /* Address of thread shutdown function */
    push((unsigned long) &tshutdown);
    /* Push the address of the thread function. */
    push((unsigned long) _tfunction);
    /* EFLAGS and instruction pointer */
    push(0);
    push(Machine::KERNEL_CS);
    push((unsigned long) &thread_start);









}
```



Creating a Thread

```
void Thread::setup_context(Thread_Function _tfunction){
    /* Address of thread shutdown function */
    push((unsigned long) &tshutdown);
    /* Push the address of the thread function. */
    push((unsigned long) _tfunction);
    /* EFLAGS and instruction pointer */
    push(0);
    push(Machine::KERNEL_CS);
    push((unsigned long) &thread_start);
    /* Fake error code and interrupt number. */
    push(0); push(0);




}
```

push 0	int_no
push 0	err_code
push **thread_start**	eip
push KERNEL_CS	es
push 0	eflags
push **_tfunction**	ret_addr
push **tshutdown**	ret_addr

Creating a Thread

```
void Thread::setup_context(Thread_Function _tfunction){
    /* Address of thread shutdown function */
    push((unsigned long) &tshutdown);
    /* Push the address of the thread function. */
    push((unsigned long) _tfunction);
    /* EFLAGS and instruction pointer */
    push(0);
    push(Machine::KERNEL_CS);
    push((unsigned long) &thread_start);
    /* Fake error code and interrupt number. */
    push(0); push(0);
    /* Initial values for general-purpose registers. */
    push(0); ...; push(0); /* eax ... edi */




}
```

push 0	edi
push 0	esi
push 0	ebp
push 0	esp
push 0	ebx
push 0	edx
push 0	ecx
push 0	eax
push 0	int_no
push 0	err_code
push **thread_start**	eip
push KERNEL_CS	es
push 0	eflags
push **_tfunction**	ret_addr
push **tshutdown**	ret_addr

Creating a Thread

```
void Thread::setup_context(Thread_Function _tfunction){
    /* Address of thread shutdown function */
    push((unsigned long) &tshutdown);
    /* Push the address of the thread function. */
    push((unsigned long) _tfunction);
    /* EFLAGS and instruction pointer */
    push(0);
    push(Machine::KERNEL_CS);
    push((unsigned long) &thread_start);
    /* Fake error code and interrupt number. */
    push(0); push(0);
    /* Initial values for general-purpose registers. */
    push(0); ...; push(0); /* eax ... edi */
    /* Segment registers */
    push(Machine::KERNEL_DS);  /* ds */
    push(Machine::KERNEL_DS);  /* es */
    push(0); push(0);  /* fs; gs */
}
```

push 0	gs
push 0	fs
push KERNEL_CS	es
push KERNEL_CS	ds
push 0	edi
push 0	esi
push 0	ebp
push 0	esp
push 0	ebx
push 0	edx
push 0	ecx
push 0	eax
push 0	int_no
push 0	err_code
push **thread_start**	eip
push KERNEL_CS	es
push 0	eflags
push **_tfunction**	ret_addr
push **tshutdown**	ret_addr

Execution Lifecycle of a Thread

```
void Thread::dispatch_to(Thread * _thread) {
    threads_low_switch_to(_thread);
}
```

push 0	gs
push 0	fs
push KERNEL_CS	es
push KERNEL_CS	ds
push 0	edi
push 0	esi
push 0	ebp
push 0	esp
push 0	ebx
push 0	edx
push 0	ecx
push 0	eax
push 0	int_no
push 0	err_code
push **thread_start**	eip
push KERNEL_CS	es
push 0	eflags
push **t_fun_1**	ret_addr
push **tshutdown**	ret_addr

Execution Lifecycle of a Thread

```
void Thread::dispatch_to(Thread * _thread) {
    threads_low_switch_to(_thread);
}
```

push **thread_start**	eip
push KERNEL_CS	es
push 0	eflags
push **t_fun_1**	ret_addr
push **tshutdown**	ret_addr

Execution Lifecycle of a Thread

```
static void thread_start() {
  /* This function is used to release
     the thread for execution in the ready queue. */
  /* We need to add code, but it is probably
     nothing more than enabling interrupts. */
}
```

push **t_fun_1**	ret_addr
push **tshutdown**	ret_addr

Execution Lifecycle of a Thread

```
void t_fun_1() {
  /* do_something_in_this_thread */
}
```

push **tshutdown** | ret_addr

Execution Lifecycle of a Thread

```
static void tshutdown() {
  /* terminates thread after thread func returns*/
}
```

Execution Lifecycle of a Thread

Low-Level Dispatching on the x86 : Summary

We learned:

- Dispatching can be implemented similarly to exception handlers.
  (Side Note: This approach is very general and portable.)
- Implementation of the Thread Control Block in C++ class **Thread**
- Dispatching in 4 Steps: Start, Store, Locate, Load
- Thread Creation
- The Execution Lifecycle of a Thread