

## COMPUTATIONAL MATH, SCIENCE AND ENGINEERING DEPARTMENT

MICHIGAN STATE  
UNIVERSITY

Threads 2

```

5312 0000488B 15470B00 004889D6 4889C7E8 6E050000 488B1535 0B000048 89D64889 C7E85C05 0000488D 35790600 00488B05 140B0000 4889C7E8 7C050000 BE080000
5376 004889C7 E85D0500 00488B15 000B0000 4889D648 89C7E827 050000E8 10050000 4889C348 8D354A06 0000488B 05070A00 004889C7 E83F0500 004889D6 4889C7E8
5440 1C050000 488B15C5 0A000048 89D64889 C7E8EC04 0000E8CF 04000048 89C3488D 351E0600 00488B05 9C0A0000 4889C7E8 04050000 4889D648 89C7E8E1 04000048
5504 8B158A0A 00004889 D64889C7 E8B10400 00488B15 780A0000 4889D648 89C7E89F 04000048 8D35E705 0000488B 05570A00 004889C7 E8BF0400 000B0800 00004889
5568 C7E8A004 0000488B 15430A00 004889D6 4889C7E8 6A040000 488B1531 0A000048 89D64889 C7E85804 0000488D 35B70500 00488B05 100A0000 4889C7E8 78040000
5632 BE040000 004889C7 E8590400 00488B15 FC090000 4889D648 89C7E823 040000E8 F4030000 660F7EC3 488D3588 05000048 8B05D209 00004889 C7E83A04 0000660F
5696 6EC34889 C7E8A004 0000488B 15BF0900 004889D6 4889C7E8 E6030000 E8B10300 00660F7E C3488D35 5B050000 488B0595 09000048 89C7E8FD 03000066 0F6EC348
5760 89C7E8CD 03000048 8B158209 00004889 D64889C7 E8A90300 00488D35 36050000 488B0561 09000048 89C7E8C9 030000BE 18000000 4889C7E8 9E030000 488B154D
5824 09000048 89D64889 C7E87403 0000488B 153B0900 004889D6 4889C7E8 62030000 488D3509 05000048 8B051A09 00004889 C7E88203 0000BE08 00000048 89C7E863
5888 03000048 8B150609 00004889 D64889C7 E82D0300 00E8E602 00006648 0F7EC348 8D35DA04 C000488B 05D80800 004889C7 E8430300 0066480F 6EC34889 C7E80603
5952 0000488B 15C70800 004889D6 4889C7E8 EE020000 E8A10200 0066480F 7EC3488D 35AC0400 00488B05 9C080000 4889C7E8 04030000 66480F6E C34889C7 E8C70200
6016 00488B15 83080000 4889D648 89C7E8AF 02000048 8D358704 0000488B 05670800 004889C7 E8CF0200 00BE3500 00004889 C7E8A402 0000488B 15530800 004889D6
6080 4889C7E8 7A020000 488B1541 08000048 89D64889 C7E86302 0000488D 355B0400 00488B05 20080000 4889C7E8 88020000 BE100000 004889C7 E8690200 00488B15
6144 0C080000 488B0648 89C7E827 050000E8 F8010900 D64889C7 E8353348 00488B05 05E30700 004889C7 E8480200 000B06D0 DB3C2448 89C7E813 02000048 8B15CE07
6208 00004889 D64889C7 E8353348 0000488B 153B0900 004889D6 4889C7E8 62030000 488D3509 05000048 8B051A09 00004889 C7E88203 0000BE08 00000048 89C7E863
6272 4889D648 89C7E8 01000048 8B150609 00004889 D64889C7 E82D0300 00E8E602 00006648 0F7EC348 8D35DA04 C000488B 05D80800 004889C7 E8430300 0066480F
6336 0000488B 15C70800 004889D6 4889C7E8 EE020000 E8A10200 0066480F 7EC3488D 35AC0400 00488B05 9C080000 4889C7E8 04030000 66480F6E C34889C7 E8C70200
6400 07000048 8B05F606 00004889 C7E86C01 0000C9C3 554889E5 BEFFF000 00BF0100 0000E8A5 FFFFFF5D C3554889 E5B80080 FFFF5DC3 554889E5 B8FF7F00 005DC355
6464 4889E5B8 00000080 5DC35548 89E5B8FF FFFF7F5D C3554889 E5488B00 00000000 0000805D C3554889 E5488BFF FFFF7F5D C3554889 E58B0589 01000066
6528 0F6EC05D C3554889 E58B057D 01000066 0F6EC05D C3554889 E5488B00 00000000 00100066 480F6EC0 5DC35548 89E5488B FFFF7F5D C3554889 E58B0589 01000066
6592 4889E548 B8000000 00000000 00BA0100 00004889 45F08955 F8DB6D0F 5DC35548 89E548C7 C0FFFFF7 FFBABE7F 00004889 45F08955 F8DB6D0F 5DC35548 89E548C7 C0FFFFF7
6656 FF252A06 0000FF25 2C060000 FF252E06 0000FF25 30060000 FF253206 0000FF25 34060000 FF253606 0000FF25 38060000 FF253A06 0000FF25 3C060000 FF253E06
6720 0000FF25 40060000 FF254206 0000FF25 44060000 FF254606 0000FF25 48060000 FF254A06 0000FF25 4C060000 FF254E06 0000FF25 50060000 FF255206 0000FF25
7424 0B000000 4C8D1D95 05000041 53FF2585 05000090 68000000 00E9E2FF 4FF66819 000000E9 DCF7FFFF 682B0000 00E902FF FFFF683D 000000E9 08F02FFF
6848 00E9BEFF FFFF6861 000000E9 B4FFFFF7 68730000 00E9AAFF FFFF6885 000000E9 A0FFFFF7 68970000 00E996FF FFFF68B7 000000E9 8CFFFFF7 68F70000 00E982FF
6912 FFFF0000 00000000 FFFF77F7 FFFF77FF 53697A65 206F6620 73686F72 743A0053 6D616C6C 65737420 73686F72 743A004C 61726765 73742073 686F7274 3A005369
6976 7A05206F 6620696E 743A0053 6D616C6C 65737420 696E743A 004C6172 767567374 3A005369 66206C6F 66E73A00 65737420 686F7274 3A005369 6D616C6C
7040 673A004C 61726765 7374206C 6F6E673A 0053697A 65206F66 206C6F6E 67206C6F 6E672069 6E743A00 53697A65 206F6620 666C6F61 743A0053 6D616C6C
7104 666C6F61 743A004C 61726765 73742066 6C6F6174 3A004469 67697473 20696E20 6D617469 737612C2 20666C6F 61743A00 53697A65 206F6620 646F7562 6C653A00
7168 536D616C 6C657374 20646F75 626C653A 004C6172 67657374 20646F75 626C653A 00446967 69747320 696E206D 61746973 73612C20 646F7562 6C653A00 53697A65
7232 206F6620 6C6F6E67 20646F75 626C653A 00536D61 6C6C6573 74206C6F 6E67206A 6F75626C 653A004C 61726765 7374206C 6F6E6720 646F7562 6C653A00 00000000
7296 44696769 74732069 6E206D61 74697373 612C206C 6F6E6720 646F7562 6C653A00 14000000 00000000 017A5200 01781001 100C0708 90010000 34000000 1C000000
7360 69FCFFFF FFFFFFFF 0B000000 00000000 00040100 00000E10 86020403 000000C7 06040600 00000C07 08000000 00000000 00000000 54000000 3CFCFFFF FFFFFFFF
7424 0B000000 00000000 00040100 00000E10 86020403 000000C7 06040600 00000C07 08000000 00000000 00000000 00000000 00000000 00000000 00000000
7488 00040100 00000E10 86020403 000000C7 08000000 00000000 00000000 00000000 00000000 00000000 00000000 00040100 00000E10 000000C7
7552 86020403 000000C7 08000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00040100 00000E10 86020403 000000C7
7616 06040B00 00000C07 08000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00040100 00000E10 86020403 000000C7
7680 08000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00040100 00000E10 86020403 000000C7
7744 34000000 A4010000 3DFBFFFF FFFFFFFF 10000000 00000000 00040100 00000E10 86020403 000000C7 08000000 00000000 00000000 00000000 00000000 00000000
7808 15FBFFFF FFFFFFFF 15000000 00000000 00040100 00000E10 86020403 000000C7 08000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
7872 15000000 00000000 00040100 00000E10 86020403 000000C7 08000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
7936 00040100 00000E10 86020403 000000C7 08000000 00000000 00000000 00000000 00000000 00000000 00000000 00040100 00000E10 86020403 000000C7
8000 86020403 000000C7 08000000 00040100 00000E10 86020403 000000C7 08000000 00000000 00000000 00000000 00040100 00000E10 86020403 000000C7
8064 06040B00 00000C07 08000000 00040100 00000E10 86020403 000000C7 08000000 00000000 00000000 00000000 00040100 00000E10 86020403 000000C7
8128 08000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00040100 00000E10 86020403 000000C7
8192 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
8256 BF190000 01000000 75190000 01000000 85190000 01000000 4A190000 01000000 3F190000 01000000 65190000 01000000 55190000 01000000 34190000 01000000
8320 29190000 01000000 9E1A0000 01000000 A31A0000 01000000 B21A0000 01000000 BC1A0000 01000000 C11A0000 01000000 D01A0000 01000000 E01A0000 01000000
8384 DA1A0000 01000000 E41A0000 01000000 EE1A0000 01000000 F81A0000 01000000 14190000 01000000 00000000 00000000 00000000 00000000 00000000 00000000
8448 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
8512 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
8576 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

```

CMSE 822, FS21, W.F. Punch

## pass args to thread func

In calling a thread, **everything** is passed to the thread as a copy.

- this is a good thing, as we want copies local in a thread to avoid any conflict
- if you want to pass by reference, you need to do `std::ref`



## 2.1

```
#include<iostream>
using std::cout; using std::endl;
#include<thread>
using std::thread;
#include<string>
using std::string; using std::to_string;

void thread_fun(long start, long stop, string &outs){
    int even=0, odd=0;
    for(int i=start; i<stop; i++)
        if (i%2 == 0)
            even++;
        else
            odd++;
    outs = "Even:" + to_string(even) + ", Odd:" + to_string(odd);
}

int main (){
    string s;
    thread t(thread_fun, 100, 1000, std::ref(s));
    t.join();
    cout << s << endl;
}
```

string by ref

need to tell thread  
this is a reference

## move semantics

You cannot copy a thread object (much like you cannot copy an ostream).

However, threads do allow for move semantics

- you can move one thread of execution to another thread object
- you can move a thread out of a function.




**2.2**

```
void thread_fun(long l, double d, string s, long &num){  
    long val;2.2  
    val = (all_of(s.begin(), s.end(), isdigit)) ? stol(s) : 0;  
    num = l + d + val;  
}
```


```
thread construct_thread(long l, double d, string s, long &num){  
    thread t(thread_fun, l, d, s, ref(num));  
    return t;  
}
```

construct a  
thread and  
return it



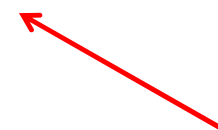
```
int main () {  
    long l;  
    thread t1(thread_fun, 1, 3.14, "1234", ref(l));  
    thread t2 = std::move(t1);  
    t2.join();  
    cout << "Long:"<<l<<endl;
```

move thread  
to another  
thread obj



```
    l = 0;  
    thread t3 = construct_thread(10, 35.678, "5678", ref(l));  
    t3.join();  
    cout << "Long:"<<l<<endl;  
}
```

thread return is a move  
to another thread object



# timing

You can use time (C++11 time) at various places in the thread libraries and their support:

- \*\_for takes a duration (how long)
- \*\_until takes a time point (wait until some timepoint in the future)



# this\_thread

`this_thread` is a namespace that identifies the presently running thread

- `sleep_for(duration)`
- `sleep_until(timepoint)`
- `get_id(unique thread it)`
- `yield()`



Operation	Effect
<code>this_thread::get_id()</code>	Yields the ID of the current thread
<code>this_thread::sleep_for(<i>dur</i>)</code>	Blocks the thread for duration <i>dur</i>
<code>this_thread::sleep_until(<i>tp</i>)</code>	Blocks the thread until timepoint <i>tp</i>
<code>this_thread::yield()</code>	Hint to reschedule to the next thread

*Table 18.5. Thread-Specific Operations of Namespace `std::this_thread`*



## 2.3

```
#include<thread>
using std::thread;
#include<chrono>
using std::chrono::duration;
using std::chrono::seconds;
#include<iostream>
using std::cout; using std::endl;
```

```
bool stop = false;
```

```
void thread_fun(){
    cout << "I am:"<<std::this_thread::get_id()<<endl;
    auto wait_for = seconds(2);
    while (!stop){
        cout << "**Inside thread...\n";
        std::this_thread::sleep_for(wait_for);
    }
}
```

```
int main() {
    thread t = thread(thread_fun);
    getchar(); // wait for user to press enter:
    stop = true; // stop thread:
    t.join();    // wait for thread to finish
}
```

thread  
unique id



timeout  
value



## checking a global variable

Notice that in the previous code, there is a global variable `stop` that all the functions can "watch"

The thread is allowed to run until the user enters a character. At that point the global changes and the thread will notice (after it's timeout) and quit.

Note where the `.join()` is



**2.4**

```

...
#include<thread>
using std::this_thread::sleep_for;
using std::this_thread::get_id;

bool stop=false;
void thread_fun(int seed){
    default_random_engine dre(seed);
    uniform_int_distribution<int> dist(500,2000);
    int rand_val;
    auto wait_for = milliseconds(1);
    while (!stop){
        rand_val = dist(dre);
        wait_for = milliseconds(rand_val);
        cout << "Thread:"<<get_id()<<" slept for:"
            << dec << wait_for.count()<<endl;
        sleep_for(wait_for);
    }
}

```

shortcut the  
this\_thread  
namespace

global var

```

int main () {
    thread t1(thread_fun, 1);
    thread t2(thread_fun, 2);
    thread t3(thread_fun, 3);
    thread t4(thread_fun, 4);
    getchar(); // wait
    stop = true; // stop thread:
    t1.join();
    t2.join();
    t3.join();
    t4.join();
}

```

all threads stop  
when global is  
set

Operation	Effect
<i>thread</i> <i>t</i>	Default constructor; creates a <i>nonjoinable</i> thread object
<i>thread</i> <i>t</i> ( <i>f</i> ,...)	Creates a thread object, representing <i>f</i> started as thread (with additional args), or throws <code>std::system_error</code>
<i>thread</i> <i>t</i> ( <i>rv</i> )	Move constructor; creates a new thread object, which gets the state of <i>rv</i> , and makes <i>rv</i> <i>nonjoinable</i>
<i>t</i> .~ <i>thread</i> ()	Destroys *this; calls <code>std::terminate()</code> if the object is <i>joinable</i>
<i>t</i> = <i>rv</i>	Move assignment; move assigns the state of <i>rv</i> to <i>t</i> or calls <code>std::terminate()</code> if <i>t</i> is <i>joinable</i>
<i>t</i> .joinable()	Yields true if <i>t</i> has an associated thread (is <i>joinable</i> )
<i>t</i> .join()	Waits for the associated thread to finish (throws <code>std::system_error</code> if the thread is not <i>joinable</i> ) and makes the object <i>nonjoinable</i>
<i>t</i> .detach()	Releases the association of <i>t</i> to its thread while the thread continues (throws <code>std::system_error</code> if the thread is not <i>joinable</i> ) and makes the object <i>nonjoinable</i>
<i>t</i> .get_id()	Returns a unique <code>std::thread::id</code> if <i>joinable</i> or <code>std::thread::id()</code> if not
<i>t</i> .native_handle()	Returns a platform-specific type <code>native_handle_type</code> for nonportable extensions

Table 18.4. Operations of Objects of Class `thread`