

ECE 2400 Computer Systems Programming

Fall 2017

Topic 16: C++ Threads

School of Electrical and Computer Engineering
Cornell University

revision: 2017-11-15-09-46

1	Concurrency via Threads	3
2	Synchronization	7

Processing an Array

```
function processArray(arr) {  
    let result = [];  
    for (let i = 0; i < arr.length; i++) {  
        if (arr[i] % 2 === 0) {  
            result.push(`Element ${i} is even`);  
        } else {  
            result.push(`Element ${i} is odd`);  
        }  
    }  
    return result;  
}  
  
const array = [1, 2, 3, 4, 5, 6, 7, 8, 9];  
processArray(array);
```

Graphical User Interface

```
function createUI() {  
    const container = document.createElement('div');  
    const title = document.createElement('h1');  
    title.textContent = 'My App';  
    const button = document.createElement('button');  
    button.textContent = 'Click me!';  
    button.addEventListener('click', () => {  
        alert('Button was clicked!');  
    });  
    container.appendChild(title);  
    container.appendChild(button);  
    return container;  
}  
  
createUI();
```

Sorting an Array

```
function sortArray(arr) {  
    let sortedArr = arr.sort((a, b) => a - b);  
    return sortedArr;  
}  
  
const array = [1, 2, 3, 4, 5, 6, 7, 8, 9];  
sortArray(array);
```

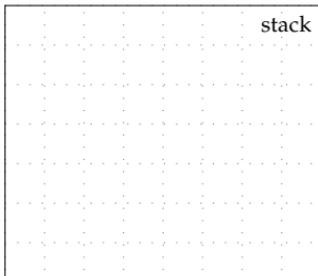
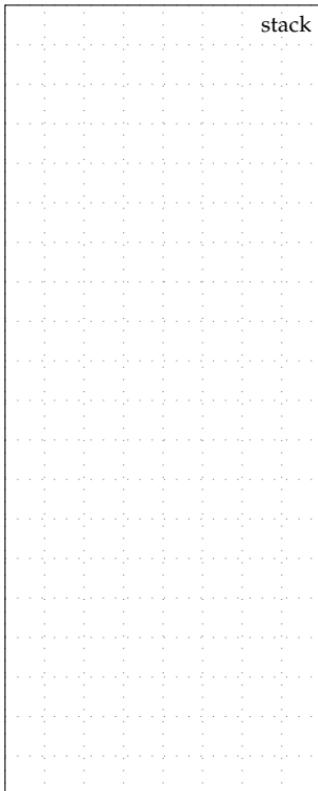
Database Transactions

```
function insertUser(user) {  
    const db = window.localStorage;  
    db.setItem(`user-${user.id}`, JSON.stringify(user));  
}  
  
function getUser(id) {  
    const db = window.localStorage;  
    const userString = db.getItem(`user-${id}`);  
    if (!userString) {  
        return null;  
    }  
    return JSON.parse(userString);  
}  
  
insertUser({  
    id: 1,  
    name: 'John Doe',  
    email: 'john.doe@example.com'  
});  
  
getUser(1);
```

1. Concurrency via Threads

```
1 #include <thread>
2
3 void avg( int* z_p, int x, int y )
4 {
5     int sum = x + y;
6     *z_p = sum / 2;
7 }
8
9 int main( void )
10 {
11     int a;
12     std::thread t( avg, &a, 5, 10 );
13
14     int b;
15     avg( &b, 10, 15 );
16
17     t.join();
18
19 }
```

<https://repl.it/@cbatten/4>



Parallel Vector-Vector Add

```
1 #include <thread>
2
3 void vvadd( int dest[], int src0[], int src1[] ,
4             size_t lo, size_t hi )
5 {
6     for ( size_t i = lo; i < hi; i++ )
7         dest[i] = src0[i] + src1[i];
8 }
9
10 int main( void )
11 {
12     const int size = N;
13     int src0[size] = { ... };
14     int src1[size] = { ... };
15     int dest[size];
16
17     size_t middle = size/2;
18     std::thread t( vvadd, dest, src0, src1, 0, middle );
19
20     vvadd( dest, src0, src1, middle, size );
21
22     t.join();
23     return 0;
24 }
```

<https://repl.it/@cbatten/3>

Parallel Merge/Quick Sort

```
1 #include <thread>
2
3 int main( void )
4 {
5     const int size = N;
6     int arr[size] = { ... };
7
8     size_t mid1 = 1*(size/4);
9     size_t mid2 = 2*(size/4);
10    size_t mid3 = 3*(size/4);
11
12    // Sort each partition in parallel
13
14    std::thread t0( qsort, arr, 0,      mid1 );
15    std::thread t1( qsort, arr, mid1, mid2 );
16    std::thread t2( qsort, arr, mid2, mid3 );
17
18    qsort( arr, mid3, mid4 );
19
20    t0.join(); t1.join(); t2.join();
21
22    // Serial merge
23
24    merge( arr, 0,      mid1, mid2 );
25    merge( arr, mid2, mid3, size );
26    merge( arr, 0,      mid2, size );
27    return 0;
28 }
```

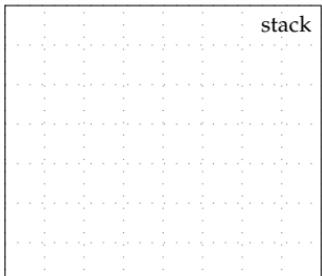
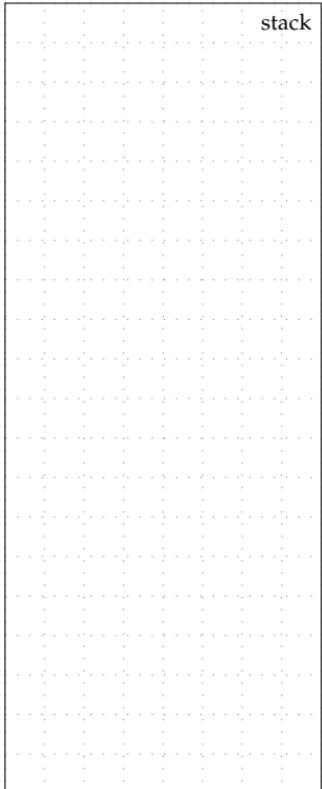
Complexity Analysis

What is the execution time complexity as a function of N (size of array) and P (number of processors) using big-O notation?

2. Synchronization

```
1 #include <thread>
2
3 void incr( int* x_p )
4 {
5     int y = *x_p;
6     int z = y + 1;
7     *x_p = z;
8 }
9
10 int main( void )
11 {
12     int a;
13     std::thread t( incr, &a );
14
15     incr( &a );
16
17     t.join();
18     return 0;
19 }
```

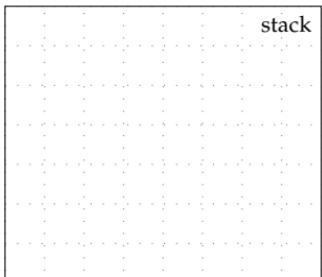
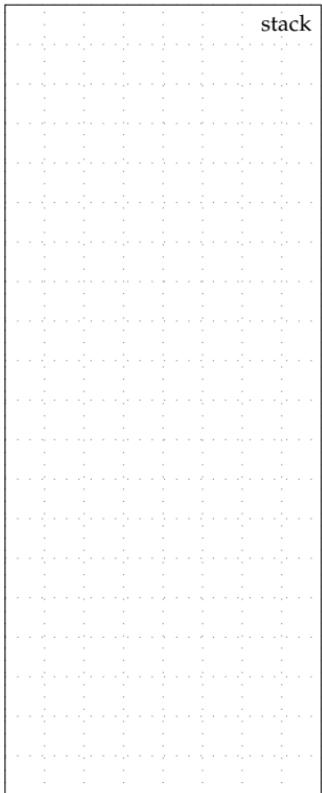
<https://repl.it/@cbatten/6>



2. Synchronization

```
1 #include <thread>
2
3 void incr( int* x_p )
4 {
5     (*x_p)++;
6 }
7
8 int main( void )
9 {
10    int a;
11    std::thread t( incr, &a );
12
13    incr( &a );
14
15    t.join();
16
17 }
```

<https://godbolt.org/g/zXLFXE>



Using atomic operations

```
1 #include <thread>
2 #include <atomic>
3
4 void incr( std::atomic<int>* x_p )
5 {
6     (*x_p)++;
7 }
8
9 int main( void )
10 {
11     std::atomic<int> a;
12     std::thread t( incr, &a );
13
14     incr( &a );
15
16     t.join();
17     return 0;
18 }
```

<https://repl.it/@cbatten/9>

<https://godbolt.org/g/bBeRzh>

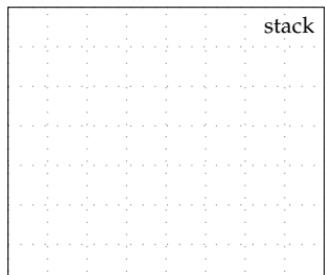
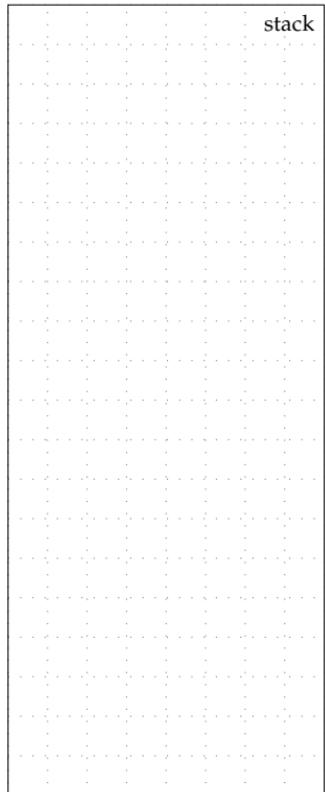
std::atomic<T> member functions

1 operator++	1 fetch_add
2 operator--	2 fetch_sub
3 operator+=	3 fetch_and
4 operator-=	4 fetch_or
5 operator&=	5 fetch_xor
6 operator =	
7 operator^=	

2. Synchronization

```
1 void incr( int* x_p,
2             std::atomic<int>* y_p )
3 {
4     // acquire lock
5     while ( y_p->fetch_or(1) == 1 )
6     { }
7
8     *x_p = foo(*x_p);
9
10    // release lock
11    *y_p = 0;
12 }
13
14 int main( void )
15 {
16     int a = 0;
17     std::atomic<int> b;
18
19     std::thread t( incr, &a, &b );
20
21     incr( &a, &b );
22
23     t.join();
24     return 0;
25 }
```

<https://repl.it/@cbatten/10>



Encapsulate lock into a mutex

```
1  class Mutex
2  {
3      public:
4          Mutex() : m_lock(0) { }
5
6          void lock()
7          {
8              while ( m_lock.fetch_or(1) == 1 ) { }
9          }
10
11         void unlock() { m_lock = 0; }
12     private:
13         std::atomic<int> m_lock;
14     };
15
16     void incr( int* x_p, Mutex* m_p )
17     {
18         m_p->lock();
19         std::cout << "start ..." << std::endl;
20         *x_p = foo(*x_p);
21         std::cout << "stop ..." << std::endl;
22         m_p->unlock();
23     }
24
25     int main( void )
26     {
27         int a = 0;
28         Mutex m;
29         std::thread t( incr, &a, &m );
30         incr( &a, &m );
31         t.join();
32         return 0;
33     }
```

<https://repl.it/@cbatten/11>

RAII: Resource Acquisition is Initialization

- What if we forget to unlock mutex?
- What if there is an exception?
- RAII ties resources to object lifetime

```
1  class LockGuard
2  {
3      public:
4
5      LockGuard( Mutex* m )
6          : m_mutex_p(m)
7      {
8          m_mutex_p->lock();
9      }
10
11     ~LockGuard()
12     {
13         m_mutex_p->unlock();
14     }
15
16     private:
17     Mutex* m_mutex_p;
18 };
19
20 void incr( int* x_p, Mutex* m_p )
21 {
22     LockGuard guard(m_p);
23
24     std::cout << "start ..." << std::endl;
25     *x_p = foo(*x_p);
26     std::cout << "stop ..." << std::endl;
27 }
```

<https://repl.it/@cbatten/12>