ECE 551

Fall 2017 Review Session

Review of Data Structure Big-Oh Behavior

	Random Access	Storage Space	Traversal	Sorted Access	Insertion	Sorted insertion	Min/Max Access	Good Uses Cases
Linked List	O(n)	O(n)	O(n)	O(n)	O(1)	O(n)	O(n) or O(1) – if sorted	Stack, Queue
BST	O(log(n))	O(n)	O(n)	O(log(n))	O(log(n))		O(log(n))	Parsing, Maps, Sets
Array	O(1)	O(n)	O(n)	O(log(n))	O(n) (could be O(1))	O(n)	O(n) or O(1) – if sorted	
Heap	O(n)	O(n)	O(n)	O(n)	O(log(n))		O(1)	Priority Queues, Dijkstras shortest path
Hash Table	O(1)	O(2n)= O(n)	O(n)	O(n)	O(1)	n/a	O(n)	Maps/Sets

Pthreads

Example: 100 random transactions between accounts 5 accounts

Serial Code:

```
int main();{
   int i, acc1, acc2;
   int balance[5] = {100, 100, 100, 100, 100};
   for (i=0; i<100; i++){
      acc1 = rand()%5;
      acc2 = rand()%5;
      balance[acc1] += 10;
      balance[acc2] -= 10;
   }
   return 0;
}
```

Pthread Function

let’s make it parallel – Using 4 threads

```
void* thread_function (void *args){




}
```

Pthread Function

let’s make it parallel – Using 4 threads

```
void* thread_function (void *args){
    int i, acc1, acc2;
    int * balance = args;




}
```

Pthread Function

let’s make it parallel – Using 4 threads

```
void* thread_function (void *args){
   int i, acc1, acc2;
   int * balance = args;
   for (i=0; i<25; i++;){
      acc1 = rand()%5;
      acc2 = rand()%5;
      balance[acc1] += 10;
      balance[acc2] -= 10;
   }
}
```

Calling Pthread Function

let’s make it parallel – Using 4 threads

```
int main();{
    int balance[5] = {100, 100, 100, 100, 100};

    return 0;
}
```

Calling Pthread Function

let’s make it parallel – Using 4 threads

```
int main();{
    int balance[5] = {100, 100, 100, 100, 100};
    pthread_t * tids = (pthread_t*)malloc_safe(4 * sizeof(pthread_t));




    return 0;
}
```

Calling Pthread Function

let's make it parallel – Using 4 threads

```
int main();{
    int balance[5] = {100, 100, 100, 100, 100};
    pthread_t * tids = (pthread_t*)malloc_safe(4 * sizeof(pthread_t));
    for (int i=0, i<4, i++){
        pthread_create(&tids[i], NULL, thread_function, &balance[0]);
    }



    return 0;
}
```

Calling Pthread Function

let's make it parallel – Using 4 threads

```
int main();{
    int balance[5] = {100, 100, 100, 100, 100};
    pthread_t * tids = (pthread_t*)malloc_safe(4 * sizeof(pthread_t));
    for (int i=0, i<4, i++){
        pthread_create(&tids[i], NULL, thread_function, &balance[0]);
    }
    for (int i=0, i<4, i++){
        pthread_join(tids[i], NULL);
    }
    return 0;
}
```

Synchronization

```
pthread_mutex_t lock;
int main();{
    pthread_mutex_init(&lock, NULL)
    int balance[5] = {100, 100, 100, 100, 100};
    pthread_t * tids = (pthread_t*)malloc_safe(4 * sizeof(pthread_t));
    for (int i=0, i<4, i++){
        pthread_create(&tids[i], NULL, thread_function, &balance[0]);
    }
    for (int i=0, i<4, i++){
        pthread_join(tids[i], NULL);
    }
    return 0;
}
```

Pthread Function – Adding Locks

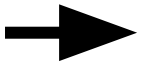
let’s make it parallel – Using 4 threads

```
void* thread_function (void *args){
   int i, acc1, acc2;
   int * balance = args;
   for (i=0; i<25; i++;){
      acc1 = rand()%5;
      acc2 = rand()%5;
      balance[acc1] += 10;
      balance[acc2] -= 10;
   }
}
```

Pthread Function – Adding Locks

let's make it parallel – Using 4 threads

```
void* thread_function (void *args){
   int i, acc1, acc2;
   int * balance = args;
   for (i=0; i<25; i++;){
      acc1 = rand()%5;
      acc2 = rand()%5;
      balance[acc1] += 10;
      balance[acc2] -= 10;
   }
}
```



Critical Section

Pthread Function – Adding Locks

let’s make it parallel – Using 4 threads

```
void* thread_function (void *args){
   int i, acc1, acc2;
   int * balance = args;
   for (i=0; i<25; i++;){
      acc1 = rand()%5;
      acc2 = rand()%5;
      pthread_mutex_lock(&lock);
      balance[acc1] += 10;
      balance[acc2] -= 10;
      pthread_mutex_unlock(&lock);
   }
}
```

Pthread Function – Print balances

```
void* thread_function (void *args){
   int i, acc1, acc2;
   int * balance = args;
   printf("I am one of the threads and the balances are:\n");
   for (i=0, i<5, i++){
      printf("Account_%d: %d\n", i, balance[i]);
   end
   for (i=0; i<25; i++;){
      acc1 = rand()%5;
      acc2 = rand()%5;}
…
```

Pthreads - Barriers

```
pthread_mutex_t lock;
pthread_barrier_t bar;
int main();{
   pthread_mutex_init(&lock, NULL)
   pthread_barrier_init(&bar, NULL, 4);
   …




}
```

```
void* thread_function (void *args){
   int i, acc1, acc2;
   int * balance = args;
   printf("I am one of the threads. The balances are:\n");
   for (i=0, i<5, i++){
      printf("Account_%d: %d\n", i, balance[i]);
   end
   pthread_barrier_wait(&bar);
…



}
```

Pthreads

Name all the synchronization points on the previous example:

Pthreads

Name all the synchronization points on the previous example:

1) Lock

Pthreads

Name all the synchronization points on the previous example:

1) Lock
2) Barrier

Pthreads

Name all the synchronization points on the previous example:

1) Lock
2) Barrier
3) Pthread Join (pthread_join(tids[i], NULL);)

Pthread – Race example

a = 15, b = 10

Thread 1:

```
pthread_mutex_lock(&lock);
a = 3;
pthread_mutex_unlock(&lock);
b += 5;
pthread_barrier_wait(&bar);
printf (“Thread 1: %d, %d\n”, a, b);
```

Thread 2:

```
pthread_mutex_lock(&lock);
a += 5;
pthread_mutex_unlock(&lock);
b = 3;
printf (“Thread 2: %d, %d\n”, a, b);
pthread_barrier_wait(&bar);
```

Is it a possible outcome?

Thread 2: 20, 3

Thread 1: 3, 8

Pthread – Race example

a = 15, b = 10

Thread 1:

```
pthread_mutex_lock(&lock);
a = 3;
pthread_mutex_unlock(&lock);
b += 5;
pthread_barrier_wait(&bar);
printf (“Thread 1: %d, %d\n”, a, b);
```

Thread 2:

```
pthread_mutex_lock(&lock);
a += 5;
pthread_mutex_unlock(&lock);
b = 3;
printf (“Thread 2: %d, %d\n”, a, b);
pthread_barrier_wait(&bar);
```

Is it a possible outcome? YES

Thread 2: 20, 3

Thread 1: 3, 8

Pthread – Race example

a = 15, b = 10

Thread 1:

```
pthread_mutex_lock(&lock);
a = 3;
pthread_mutex_unlock(&lock);
b += 5;
pthread_barrier_wait(&bar);
printf (“Thread 1: %d, %d\n”, a, b);
```

Thread 2:

```
pthread_mutex_lock(&lock);
a += 5;
pthread_mutex_unlock(&lock);
b = 3;
printf (“Thread 2: %d, %d\n”, a, b);
pthread_barrier_wait(&bar);
```

Is it a possible outcome?

Thread 2: 3, 3

Thread 1: 3, 8

Pthread – Race example

a = 15, b = 10

Thread 1:

```
pthread_mutex_lock(&lock);
a = 3;
pthread_mutex_unlock(&lock);
b += 5;
pthread_barrier_wait(&bar);
printf (“Thread 1: %d, %d\n”, a, b);
```

Thread 2:

```
pthread_mutex_lock(&lock);
a += 5;
pthread_mutex_unlock(&lock);
b = 3;
printf (“Thread 2: %d, %d\n”, a, b);
pthread_barrier_wait(&bar);
```

Is it a possible outcome? YES

Thread 2: 3, 3

Thread 1: 3, 8

Pthread – Race example

a = 15, b = 10

Thread 1:

```
pthread_mutex_lock(&lock);
a = 3;
pthread_mutex_unlock(&lock);
b += 5;
pthread_barrier_wait(&bar);
printf (“Thread 1: %d, %d\n”, a, b);
```

Thread 2:

```
pthread_mutex_lock(&lock);
a += 5;
pthread_mutex_unlock(&lock);
b = 3;
printf (“Thread 2: %d, %d\n”, a, b);
pthread_barrier_wait(&bar);
```

Is it a possible outcome?

Thread 2: 20, 15

Thread 1: 20, 15

Pthread – Race example

a = 15, b = 10

Thread 1:

```
pthread_mutex_lock(&lock);
a = 3;
pthread_mutex_unlock(&lock);
b += 5;
pthread_barrier_wait(&bar);
printf (“Thread 1: %d, %d\n”, a, b);
```

Thread 2:

```
pthread_mutex_lock(&lock);
a += 5;
pthread_mutex_unlock(&lock);
b = 3;
printf (“Thread 2: %d, %d\n”, a, b);
pthread_barrier_wait(&bar);
```

Is it a possible outcome? NO

Thread 2: 20, 15

Thread 1: 20, 15

Pthread – Race example

a = 15, b = 10

Thread 1:

```
pthread_mutex_lock(&lock);
a = 3;
pthread_mutex_unlock(&lock);
b += 5;
pthread_barrier_wait(&bar);
printf (“Thread 1: %d, %d\n”, a, b);
```

Thread 2:

```
pthread_mutex_lock(&lock);
a += 5;
pthread_mutex_unlock(&lock);
b = 3;
printf (“Thread 2: %d, %d\n”, a, b);
pthread_barrier_wait(&bar);
```

Is it a possible outcome?

Thread 1: 20, 3

Thread 2: 3, 8

Pthread – Race example

a = 15, b = 10

Thread 1:

```
pthread_mutex_lock(&lock);
a = 3;
pthread_mutex_unlock(&lock);
b += 5;
pthread_barrier_wait(&bar);
printf (“Thread 1: %d, %d\n”, a, b);
```

Thread 2:

```
pthread_mutex_lock(&lock);
a += 5;
pthread_mutex_unlock(&lock);
b = 3;
printf (“Thread 2: %d, %d\n”, a, b);
pthread_barrier_wait(&bar);
```

Is it a possible outcome? NO

Thread 1: 20, 3

Thread 2: 3, 8

Question 18.7

```
#include <iostream>
#include <cstdlib>

class A {
protected:
  int x;
public:
  A(): x(0) {     std::cout <<"A()\n"; }
  A(int _x): x(_x) {  std::cout <<"A("<<x<<")\n"; }
  virtual ~A() { std::cout <<"~A()\n"; }
  int myNum() const { return x; }
  virtual void setNum(int n) {  x = n; }
};

class B : public A {
protected:
  int y;
public:
  B(): y(0) { std::cout <<"B()\n"; }
  B(int _x, int _y): A(_x), y(_y) {
    std::cout <<"B("<<x<<","<<y<<")\n";
  }
  virtual ~B() {  std::cout <<"~B()\n"; }
  virtual int myNum() const { return y; }
  virtual void setNum(int n) { y = n; }
};
```

```
int main(void) {
  B * b1 = new B();
  B * b2 = new B(3, 8);
  A * a1 = b1;
  A * a2 = b2;
  b1->setNum(99);
  a1->setNum(42);
  std::cout << "a1->myNum() = " << a1->myNum() << "\n";
  std::cout << "a2->myNum() = " << a2->myNum() << "\n";
  std::cout << "b1->myNum() = " << b1->myNum() << "\n";
  std::cout << "b2->myNum() = " << b2->myNum() << "\n";
  delete b1;
  delete a2;
  return EXIT_SUCCESS;
}
```

vtables

- A table of function pointers used to call virtual methods based on the *dynamic* type of an object.

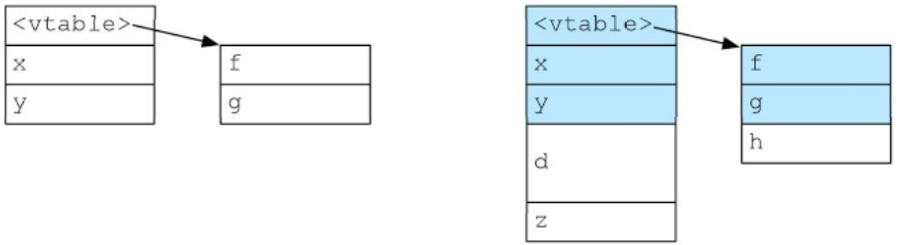


Figure 29.2: Example object layouts with vtables

Drawing object layout 1

```
1   class Foo {
2       void * data;
3       size_t size;
4       virtual void process();
5       virtual void compress();
6   };
7
8   class Bar : public Foo {
9       size_t count;
10      virtual void process();
11      virtual void extract();
12  };
13
```

Let’s look at the *real* vtable!

```
1 Vtable for Foo
2 Foo::_ZTV3Foo: 4 entries
3 0      (int (*)(...))0
4 8      (int (*)(...))(& _ZTI3Foo)
5 16     (int (*)(...))Foo::process
6 24     (int (*)(...))Foo::compress
```

```
14 Vtable for Bar
15 Bar::_ZTV3Bar: 5 entries
16 0      (int (*)(...))0
17 8      (int (*)(...))(& _ZTI3Bar)
18 16     (int (*)(...))Bar::process
19 24     (int (*)(...))Foo::compress
20 32     (int (*)(...))Bar::extract
```

Drawing object layout 2

```
1   class Foo {
2       void * data;
3       virtual void compress();
4   };
5
6   class Bar {
7       size_t count;
8       virtual void extract();
9       virtual void process();
10  };
11
12  class X : public Foo, public Bar {
13      char z;
14      virtual void doX();
15  };
```

Let’s look at the *real* vtable!

```
1 Vtable for Foo
2 Foo::_ZTV3Foo: 3 entries
3 0     (int (*)(...))0
4 8     (int (*)(...))(& _ZTI3Foo)
5 16    (int (*)(...))Foo::compress
```

```
13 Vtable for Bar
14 Bar::_ZTV3Bar: 4 entries
15 0     (int (*)(...))0
16 8     (int (*)(...))(& _ZTI3Bar)
17 16    (int (*)(...))Bar::extract
18 24    (int (*)(...))Bar::process
```

```
26 Vtable for X
27 X::_ZTV1X: 8 entries
28 0     (int (*)(...))0
29 8     (int (*)(...))(& _ZTI1X)
30 16    (int (*)(...))Foo::compress
31 24    (int (*)(...))X::doX
32 32    (int (*)(...))-16
33 40    (int (*)(...))(& _ZTI1X)
34 48    (int (*)(...))Bar::extract
35 56    (int (*)(...))Bar::process
```

Drawing object layout 3

```
1   class Foo {
2       void * data;
3       virtual void compress();
4   };
5
6   class Bar : public Foo {
7       size_t count;
8       virtual void extract();
9       virtual void process();
10  };
11
12  class X : public Foo {
13      char z;
14      virtual void doX();
15  };
16
17  class A : public Bar, public X {
18      int a;
19      virtual void doA();
20  };
```

Drawing object layout 4

```
1   class Foo {
2       void * data;
3       virtual void compress();
4   };
5
6   class Bar : public virtual Foo {
7       size_t count;
8       virtual void extract();
9       virtual void process();
10  };
11
12  class X : public virtual Foo {
13      char z;
14      virtual void doX();
15  };
16
17  class A : public Bar, public X {
18      int a;
19      virtual void doA();
20  };
```