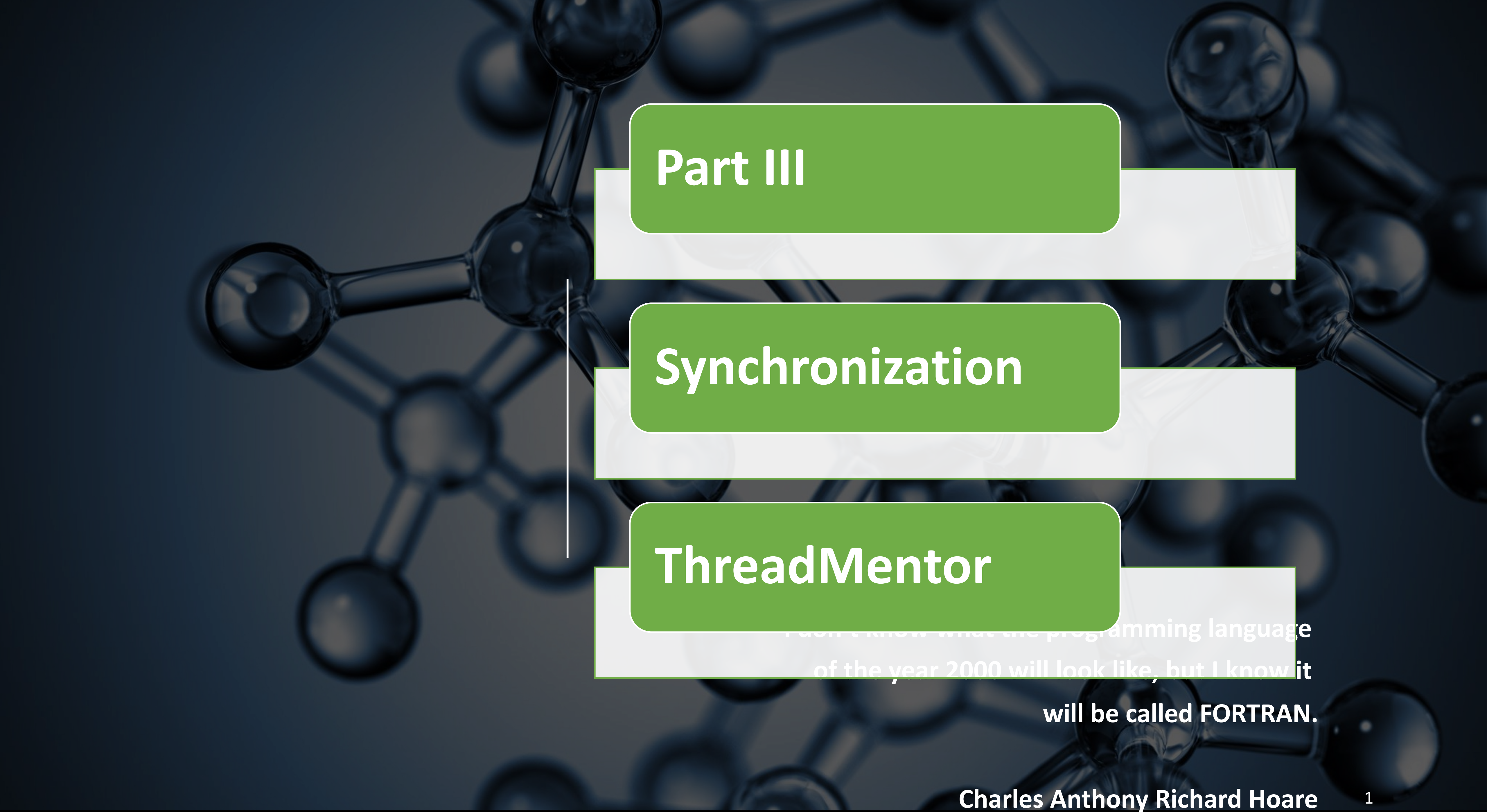


Part III

Synchronization

ThreadMentor

I don't know what the programming language
of the year 2000 will look like, but I know it
will be called FORTRAN.

Charles Anthony Richard Hoare

ThreadMentor

Basics

Example Code

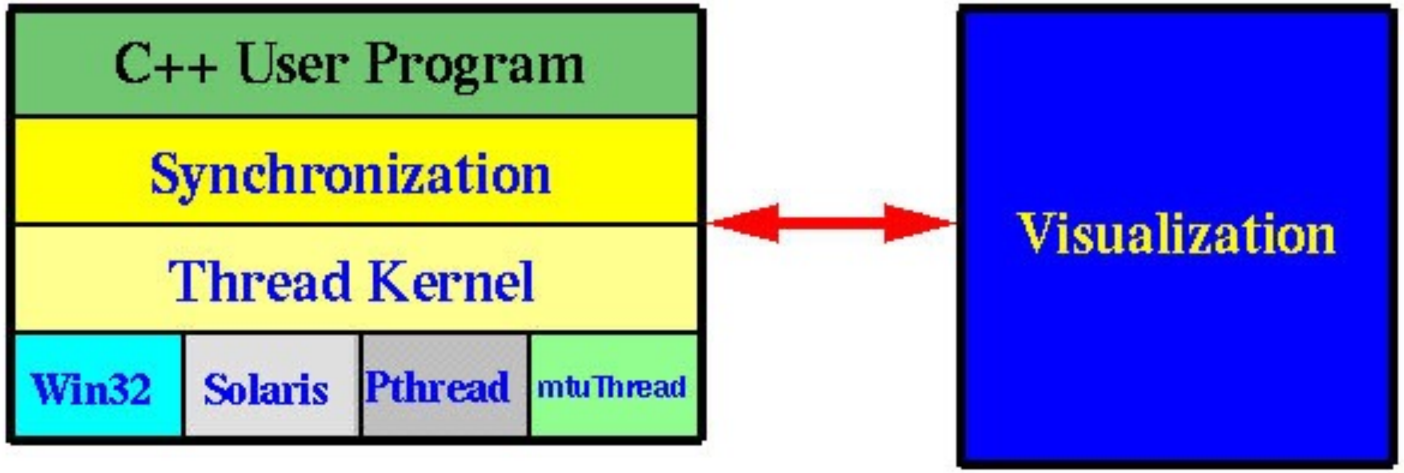
```
~jmayo/public/cs3331/tm/
```

This code, like the overheads, is from the course materials developed by Dr. CK Shene.

ThreadMentor Architecture

- ThreadMentor consists of a class library and a visualization system.
- The class library provides all mechanisms for thread management and synchronization primitives.
- The visualization system helps visualize the dynamic behavior of multithreaded programs.

ThreadMentor Architecture



Basic Thread Management

- Thread creation: creates a new thread
- Thread termination: terminates a thread
- Thread join: waits for the completion of another thread
- Thread yield: yields the execution control to another thread
- Suspend/Resume: suspends or resumes the execution of a thread.

How to Define a Thread?

- A thread should be declared as a derived class of `Thread`.
- All executable code must be in function `ThreadFunc()`.
- A thread may be assigned a name with a constructor.
- Method `Delay()` may be used to delay the thread execution for a random time.

```
#include “ThreadClass.h”
class test : public Thread {
    public:
        test(int i){n=i;};
    private:
        int n;
        void ThreadFunc(int);
};

void test::ThreadFunc() {
    Thread::ThreadFunc();
    for (int i=0; i<10; i++)
        cout << n << i << endl;
    // other stuffs
}
```

Call is necessary for initialization. Side effect of implementation.

may not be thread safe!

```
#include “ThreadClass.h”
class test : public Thread {
   public:
      test(int i){n=i;};
   private:
      int n;
      void ThreadFunc(int);
};

void test::ThreadFunc(int n) {
   Thread::ThreadFunc();
   for (int i=0; i<10; i++)
      cout << n << i << endl;
   // other stuffs
}
```

In the notes, but use of a parameter does not appear to be supported

Create and Run a Thread

- Declare a thread just like declaring an `int` variable.
- Then, use method `Begin()` to run a thread.

```
int main(void)
{
   test* Run[3];
   int   i;
   for (i=0;i<3;i++) {
        Run[i] = new test(i) ;
        Run[i]->Begin() ;
   }
   // other stuffs
}
```

A Few Important Notes

- Before calling method `Begin()`, the created thread **does not** run.
- Function `ThreadFunc()` **never** returns. When it reaches the end or executes a return, it *disappears*!
- Do not use `exit()`, as it terminates the whole system. See next slide.

Terminating a Thread

- Use method `Exit()` of the thread class `Thread`.
- Do not use system call `exit()` as it terminates the whole program.

```
void test::ThreadFunc(int n)
{
   Thread::ThreadFunc() ;

   for (int i=0;i<10;i++)
      cout << n << i << end;
   Exit() ;  // terminates
}
```

Thread Join

- **Sometimes, a thread must wait until the completion of another thread so that the results computed by the latter can be used.**
- **The parent must wait until all of its child threads complete. Otherwise, when the parent exits, all of its child threads exit.**

The `Join()` Method

- Use the `Join()` method of a thread to join with that thread.
- Suppose thread A must wait for thread B's completion. Then, do the following in thread A:

```
B->Join()
```

or

```
B.Join()
```

Thread Join Semantics

Suppose thread A wants to join with thread B, we have two cases:

1. If A reaches the `Join()` call before B exits, A waits until B completes.
2. If B exits before A can reach the `Join()` call, then A continues as if there is no `Join()`.

A Simple Example

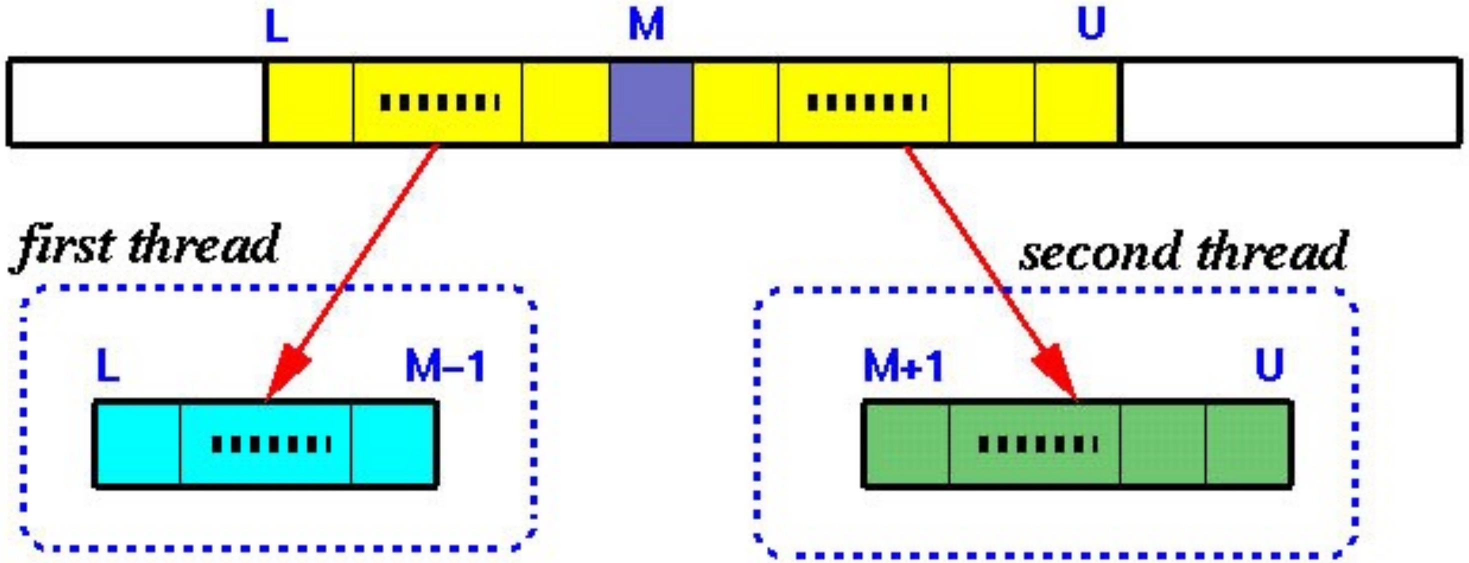
```
#include “ThreadClass.h”
class test : public Thread{
   public:
      test(int i){n = i;};
   private:
      int n;
      void  ThreadFunc();
};
void test::ThreadFunc(int n)
{
   Thread::ThreadFunc();
   for (int i=0; i<10; i++)
    cout << n << i << endl;
   Exit();
}
```

```
#include “ThreadClass.h”

int main(void)
{
  test* Run[3];

  for (int i=0;i<3;i++){
      Run[i] = new test(i);
      Run[i]->Begin();
  }
  for (i = 0; i<3; i++)
      Run[i]->Join() ;
  Exit();
}
```

Quicksort Example

- **In each recursion step, the quicksort cuts the given array segment `a[L:U]` into two with a pivot element `a[M]` such that all elements in `a[L:M-1]` are less than `a[M]` and all elements in `a[M+1:U]` are greater than `a[M]`. Then, `a[L:M-1]` and `a[M+1:U]` are sorted independently and recursively.**
- **Since `a[L:M-1]` and `a[M+1:U]` are sorted independently, we may use a thread for each segment!**

- A thread receives the array segment `a[L:U]` and partitions it into `a[L:M-1]` and `a[M+1:U]`.
- Then, creates a thread to sort `a[L:M-1]` and a second thread to sort `a[M+1:U]`.



Thus, our strategy looks like the following:

1. A thread receives array `a[L:R]`.
2. It finds the pivot element `a[M]`.
3. Creates a child thread and provides it with `a[L:M-1]`.
4. Creates a child thread and provides it with `a[M+1:R]`.
5. Issues two thread `Join()`s waiting for both child threads.

Class Quicksort: Definition

```
class Quicksort : public Thread
{
   public:
      Quicksort(int L, int U, int a[]);
   private:
      int  low;   /* Low index */
      int  up;    /* High index */
      int  *a;    /* Array  */
      void ThreadFunc();
};
```

quicksort.h

Class Quicksort: Implementation

```
Quicksort::Quicksort(int L, int U, int A[])
           :low(L), up(U), a(A){
     ThreadName = // set a thread name;
}
Void Quicksort::ThreadFunc(){
   Thread::ThreadFunc();  // required
   Quicksort  *Left, *Right;
   int         M;
   M = // compute the pivot element;
   Left = new Quicksort(low, M-1, a); Left->Begin();
   Right = new Quicksort(M+1, up, a); Right->Begin();
   Left->Join();
   Right->Join();
   Exit();
}
```

The array has already been initialized external to the QuickSort object.
Getting pointer and indices to it from the constructor when the object is created.

quicksort.cpp

Class Quicksort: Main Program

The main program is easy:

```
int  main(void)
{
   Quicksort   *thread;
   int         a[MAXSIZE], L, U, n;
   // read in array a[] and # of elements n
   L = 0; U = n-1;
   thread = new Quicksort(L, U, a);
   thread->Begin();
   thread->Join();
   Exit();
}
```

quicksort-main.cpp

What If We Have the Following?

```
Quicksort::Quicksort(int L, int U, int A[])
           :low(L), up(U), a(A)
{
     ThreadName = // set a thread name;
}

void Quicksort::ThreadFunc()
{
   Thread::ThreadFunc();
   Quicksort  *Left, *Right;
   int        M;
   M = // compute the pivot element;
   Left = new Quicksort(low, M-1, a);
      Left->Begin();   Left->Join();
   Right = new Quicksort(M+1, up, a);
      Right->Begin(); Right->Join();
   Exit();
}
```

`Join()` are moved to right after `Begin()`. Is this a correct program? Does it fulfill the maximum concurrency requirement?

Compilation with ThreadMentor

- **ThreadMentor** adds all visualization features in its class library implicitly so that you don’t have to do anything in your program to use visualization.
- But, you need to recompile your program properly so that a correct library will be used.
- There are two versions of **ThreadMentor** library: Visual and non-Visual.
- This `Makefile` is in: `~jmayo/public/cs3331/tm/`

Makefile for ThreadMentor

Define some names.
Don’t touch this portion.

visual library

non-visual library

use this only when you work on your home machine
remove this in your submission

```
CC          = c++
FLAGS       = -no-pie
CFLAGS      = -g -O2 -Wno-write-strings -Wno-cpp -w
DFLAGS      = -DPACKAGE=\"threadsystem\" ......
IFLAGS      = -I/local/eit-linux/apps/ThreadMentor/include
TMLIB       = /local/eit-linux/apps/ThreadMentor/Visual/...
TMLIB_NV    = /local/eit-linux/apps/ThreadMentor/NoVisual/...

OBJ_FILE = quicksort.o quicksort-main.o
EXE_FILE = quicksort
```

These two flags eliminate the most common warning messages related to ThreadMentor

this is the executable file

list the .o files here

eliminate **ALL** warning messages
Add this one when you submit

generate executable file with visual

```
${EXE_FILE}: ${OBJ_FILE}
	${CC} ${FLAGS} -o ${EXE_FILE} ${OBJ_FILE} ${TMLIB} -lpthread
```

tab

remove this in your submission

```
quicksort.o: quicksort.cpp
	${CC} ${DFLAGS} ${IFLAGS} ${CFLAGS} -c quicksort.cpp

quicksort-main.o: quicksort-main.cpp
	${CC} ${DFLAGS} ${IFLAGS} ${CFLAGS} -c quicksort-main.cpp

noVisual: ${OBJ_FILE}
	${CC} ${FLAGS} -o ${EXE_FILE} ${OBJ_FILE} ${TMLIB_NV} -lpthread

clean:
	rm -f ${OBJ_FILE} ${EXE_FILE}
```

clean up

generate executable file without visual

- **By default, the above `Makefile` generates executable with visual. The following generates executable `quicksort`:**

```
make
```

- **If you do not want visualization, use the following:**

```
make noVisual
```

- **To clean up the `.o` and executable files, use**

```
make clean
```

- Add the following line to your `.bashrc`, which is in your home directory. Then, logout and login again to make it effective:

```
set path=($path /local/eit-linux/apps/ThreadMentor/bin)
```

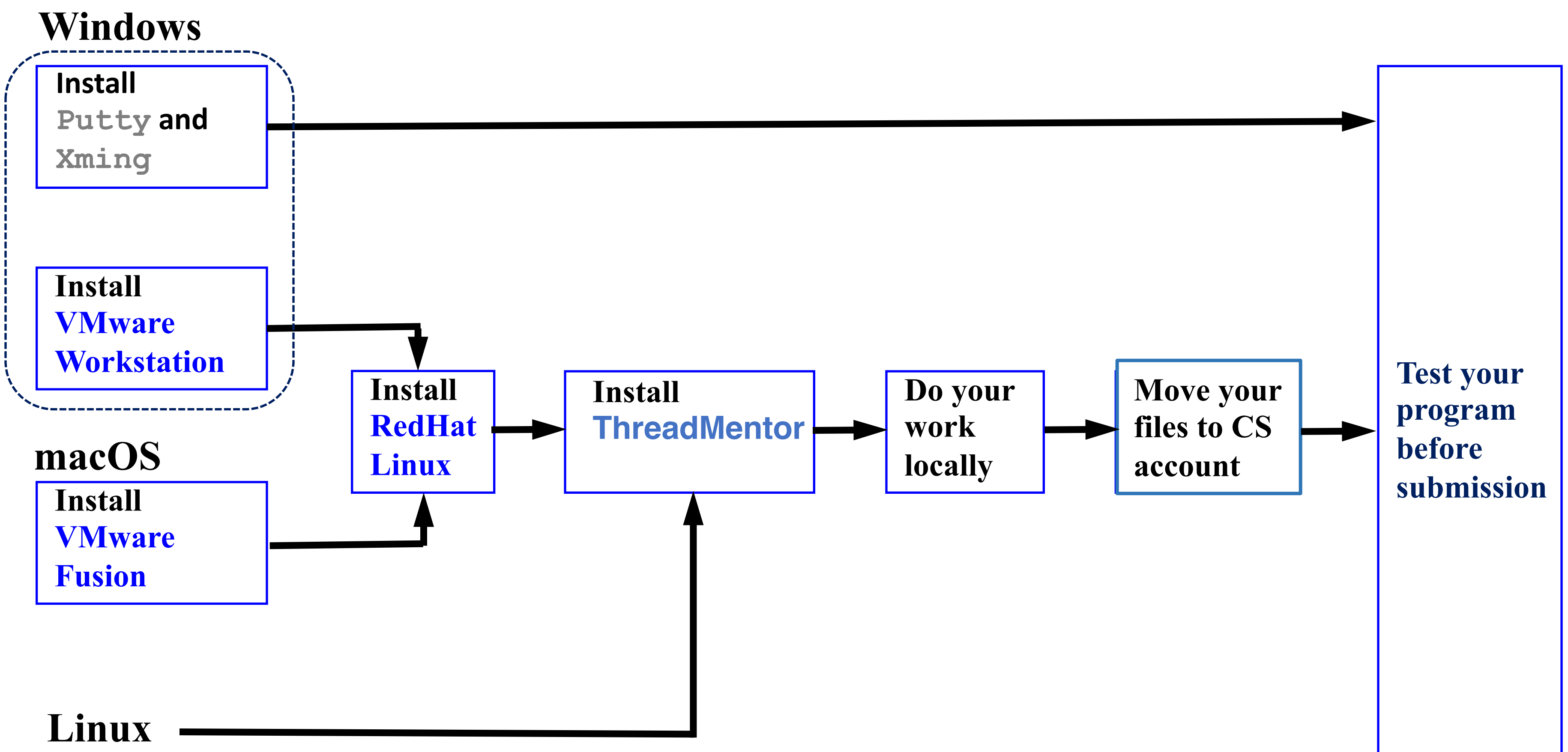
- More **ThreadMentor** examples are available at the **ThreadMentor** tutorial site:

http://www.cs.mtu.edu/~shene/NSF-3/e-Book/index.html

Help!

https://pages.mtu.edu/~shene/NSF-3/e-Book/index.html

Running ThreadMentor @ Home



See the links under the "Programming Information" section

Not applicable to Apple Silicon M1 CPU computers

The End