CSE 333
Section 09

Booooooost & Threads

Logistics:

Due Monday (06/01):

Exercise 17 - pthreads

Due next Thursday (06/04):

HW4 (Can use two late days)

How is HW4 going? Any questions?

BOOOOOOOOST

BOOST

Boost is a free C++ library that provides support for various tasks in C++

- **Note:** Boost does NOT follow the Google style guide!!!

Boost adds many string algorithms that you may have seen in Java

- Include with `#include <boost/algorithm/string.hpp>`

We are showcasing a few we think could be useful for HW4, but more can be found here:

- https://www.boost.org/doc/libs/1_60_0/doc/html/string_algo.html

trim

```
void boost::trim(string& input);
```

- Removes all leading and trailing whitespace from the string
- `input` is an input *and* output parameter (non-const reference)

```
string s("    HI    ");
boost::algorithm::trim(s);

// results in s == "HI"
```

replace_all

```
void boost::replace_all(string& input, const string& search,
                                       const string& format);
```

- Replaces all instances of `search` inside `input` with `format`

```
string s("ynrnrt");
boost::algorithm::replace_all(s, "nr", "e");

// results in s == "yeet"
```

replace_all

```
void boost::replace_all(string& input, const string& search,
                                       const string& format);
```

- Replaces all instances of `search` inside `input` with `format`

```
string s("queue?");
boost::algorithm::replace_all(s, "que", "q");

// results in s == "que?"
```

replace_all() guarantees that 'format' will be in the final result if-and-only-if 'search' existed.

replace_all() makes a *single* pass over input.

split

```
void boost::split(vector<string>& output,
                  const string& input,
                  boost::PredicateT match_on,
                  boost::token_compress_mode_type compress);
```

- Split the string by the characters in `match_on`

```
boost::PredicateT boost::is_any_of(const string& tokens);
```

- Returns predicate that matches on any of the characters in tokens

split Examples

```
vector<string> tokens;

string s("I-am--split");

boost::split(tokens, s, boost::is_any_of("-"),
                        boost::token_compress_on);
// results in tokens == ["I", "am", "split"]

boost::split(tokens, s, boost::is_any_of("-"),
                        boost::token_compress_off);
// results in tokens == ["I", "am", "", “split"]
```

Boost - Extra practice

Write a function that takes in a string that contains words separated by whitespace and returns a vector that contains all of the words in that string, in the same order as they show up, but with no duplicates. Ignore all leading and trailing whitespace in the input string.

Example:
`RemoveDuplicates(" Hi I'm sorry  jon sorry hi hihi  hi hi     ")`
should return the vector `["Hi", "I'm", "sorry", "jon", "hi", "hihi"]`

```
vector<string> RemoveDuplicates(const string& input){
      string copy(input);
      boost::algorithm::trim(copy);
      std::vector<string> components;
      boost::split(components, copy, boost::is_any_of(" \t\n"),

    boost::token_compress_on);
      std::vector<string> result;
      for (uint i = 0; i < components.size(); ++i) {
        bool unique = true;
        for (uint j = 0; j < i && unique; ++j) {
          unique &= !(components[i] == components[j]);
        }
        if (unique) {
          result.push_back(components[i]);
        }
      }
}     return result;
```

Threads

“Computers are really dumb. They can only do a few things like shuffling around numbers, but they do them really really fast so that they appear smart.”

-Hal Perkins

Threads are just a way of making computers appear to do multitasking, *regardless of whether they are running one or more* CPU*s*

Threads

- Contained within a process.
- Multiple threads can exist within the same process.
  - Every process starts with one thread of execution, but it can spawn more.
- Threads in a single process share one address space
  - Instructions (code)
  - Static (global) data
  - Dynamic (heap) data
  - Environment variables, open files, sockets, etc.
- Easy communication (put something in shared memory)
- Synchronization often uses locks (like mutexes)

POSIX threads (pthreads)

- The POSIX standard provides APIs for creating and manipulating threads.
- Part of the standard C/C++ libraries, declared in pthread.h
- Core pthread functions:
  - pthread_create - “Go do this {function}”
  - pthread_exit - “I’m done with my task!”
  - pthread_join - “I’ll wait for you to report back your result”
  - pthread_cancel - “I changed my mind, you can stop now”
  - pthread_detach - “You’re free now, go forth and prosper”

pthread_create

```
#include <pthread.h>
int pthread_create( pthread_t *thread,
                    const pthread_attr_t *attr,
                    void *(*start_routine) (void *),
                    void *arg );
```

- `pthread_create` creates a new thread and calls start_routine with arg as its parameter.
- `pthread_create` arguments:
  - **thread:** Pointer to a unique identifier for the new thread. (output parameter)
  - **attr:** An attribute object that may be used to set thread attributes. Use NULL for the default values.
  - **start_routine:** The C routine that the thread will execute once it is created.
  - **arg**: A single argument that may be passed to *start_routine*. It must be passed by reference as a pointer cast of type void. NULL may be used if no argument is to be passed.
- Compile and link with –pthread.

Threads - Quick Check

```
MyClass onTheStack;
pthread_t child;
pthread_create(&child, nullptr, foo, &onTheStack);
```

onTheStack is on the parent thread's stack. However, each thread has its own stack! Can we still access onTheStack from the child? Why or why not?

Yes! All threads share an address space

Terminating Threads

- There are several ways in which a thread may be terminated:
  - The thread returns normally from its starting routine; Its work is done.
  - The thread makes a call to the `pthread_exit` subroutine - whether its work is done or not.
  - The thread is canceled by another thread via the `pthread_cancel` routine.
  - The entire process is terminated due to making a call to either the `exec()` or `exit()`.
  - If `main()` finishes first, without calling `pthread_exit` explicitly itself.

pthread_exit

```
void pthread_exit(void *retval);
```

- Allows the user to terminate a thread and to specify an optional termination status parameter, *`retval`*.
- In subroutines that execute to completion normally, you can often dispense with calling `pthread_exit()`.
- Calling pthread_exit() from main():
  - If `main()` finishes before the threads it spawned and does not call `pthread_exit()` explicitly, all the threads it created will terminate.
  - To allow other threads to continue execution, the main thread should terminate by calling `pthread_exit()` rather than `exit()`.

Demo simple_pthreads.cc

Threads - Gotchas

- Resources (heap-allocated storage, file descriptors, etc)
  - Often shared between multiple threads
  - Must be allocated / deallocated *exactly once*
  - Don’t use deallocated resources from other threads

```
buf = new int[BUFSIZE];
...
if (!handleRequest(buf, req, len)) {
        delete[] buf; // buf was allocated in this thread
        close(fd); // is somebody else going to try to use
fd???
        pthread_exit(nullptr);
}
```

pthread_join

```
int pthread_join(pthread_t thread, void **retval);
```

- Synchronization between threads.
- `pthread_join` blocks the calling thread until the specified thread terminates and then the calling thread joins the terminated thread.
- Only threads that are created as joinable can be joined; a thread created as detached can never be joined. (Refer `pthread_create`)
- The target thread's termination return status can be obtained if it was specified in the target thread's call to `pthread_exit()`.

Demo: *pthreads.cc*

Locking - mutex

```
int pthread_mutex_init(pthread_mutex_t *mutex,
                       const pthread_mutexattr_t *attr);
```

- Initializes the mutex lock pointed to by `mutex` with lock attributes specified by `attr`.
- `Attr` can be `null`.

```
int pthread_mutex_lock(pthread_mutex_t *mutex);
```

- Grabs the lock

```
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

- Releases the lock

```
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

- Destroys the lock

Threads – Locking

- Locking is hard.
  - Too much, and performance is *worse than sequential*
  - Too little, and threads clash - *often unexpected results*
  - Not careful, and deadlock freezes your program forever!

```
pthread_mutex_lock(&lock);
if (!do_computation(resource)) {
        printf("Error doing computation\n");
        return false;  // !!!
}
pthread_mutex_unlock(&lock);
return true;
```

Demo total.cc & total_locking.cc

More examples of pthreads usage

- From sequential to concurrent merge sort
    1. merge_sort.cc
    2. c4_merge_sort.cc
    3. concurrent_merge_sort.cc