CMPUT 379 Lab

ETLC E1003: Tuesday, 5:00 – 7:50 PM.

Tianyu Zhang, Peiran Yao

CAB 311: Thursday, 2:00 – 4:50 PM.

Max Ellis, Aidan Bush

Last Week...

- Threading examples
- Fine and coarse grained locks

Today’s Lab

- Semaphore
- FAQ

More example on Condition Variable

```
// increment counter a few times
// wake up watch_count thread when reaching COUNT_LIMIT
void *inc_count(void *t) {
    int my_id = *(int*)t;
    for (int i=0; i < NUM_INC; ++i) {
        pthread_mutex_lock(&count_mutex);
        count++;
        // check the value of count and signal waiting thread when
        // condition is reached. This occurs while mutex is locked
        if (count == COUNT_LIMIT) {
            pthread_cond_signal(&count_cond);
            printf("inc_count: thread %d, count = %d Threshold reached.\n", my_id, count);
        }
        printf("inc_count: thread %d, count = %d, unlocking mutex\n", my_id, count);
        pthread_mutex_unlock(&count_mutex);
    sleep(1); // do some "work" so threads can alternate on mutex lock
    }
return nullptr;
}
```

More example on Condition Variable

```
void *watch_count(void *t) { // wait until signalled, then add 125
    int my_id = *(int*)t;
    printf("Starting watch_count: thread %d\n", my_id);
    // Lock mutex and wait for signal. pthread_cond_wait will unlock
    // mutex while it waits. Also, if COUNT_LIMIT is reached before
    // this function is run by the waiting thread, the loop will be
    // skipped to prevent pthread_cond_wait from never returning
    pthread_mutex_lock(&count_mutex);
    while (count < COUNT_LIMIT) {
        pthread_cond_wait(&count_cond, &count_mutex);
        // check whether we actually received a signal
        if (count >= COUNT_LIMIT) {
            printf("watch_count: thread %d signal received.\n", my_id);
            count += 125;
            printf("watch_count: thread %d count now = %d.\n", my_id, count);
        }
    }
    pthread_mutex_unlock(&count_mutex);
    return 0;
}
```

More example on Condition Variable

```
int main () {
    pthread_t *threads = new pthread_t[NUM_THREADS];
    int *ids = new int[NUM_THREADS];
    // initialize mutex and condition variable objects
    pthread_mutex_init(&count_mutex, 0);
    pthread_cond_init(&count_cond, 0);
    ids[0] = 0;
    pthread_create(&threads[i], nullptr, watch_count, (void *)&ids[0]);
    for (int i=1; i < NUM_THREADS; ++i) {
        ids[i] = i;
        pthread_create(&threads[i], nullptr, inc_count, (void *)&ids[i]);
    }
    // wait for all threads to complete
    for (int i=0; i < NUM_THREADS; ++i)
        pthread_join(threads[i], nullptr);
    printf("Main(): Waited on %d threads. Done.\n", NUM_THREADS);
    // clean up and exit
    pthread_mutex_destroy(&count_mutex);
    pthread_cond_destroy(&count_cond);
    delete [] threads;
    delete [] ids;
    return 0;
}
```

More example on Condition Variable

```
inc_count: thread 1, count = 1, unlocking mutex
Starting watch_count: thread 0
inc_count: thread 2, count = 2, unlocking mutex
inc_count: thread 1, count = 3, unlocking mutex
inc_count: thread 2, count = 4, unlocking mutex
inc_count: thread 1, count = 5, unlocking mutex
inc_count: thread 2, count = 6, unlocking mutex
inc_count: thread 1, count = 7, unlocking mutex
inc_count: thread 2, count = 8, unlocking mutex
inc_count: thread 1, count = 9, unlocking mutex
inc_count: thread 2, count = 10, unlocking mutex
inc_count: thread 1, count = 11, unlocking mutex
```

```
inc_count: thread 2, count = 12 Threshold reached.
inc_count: thread 2, count = 12, unlocking mutex
watch_count: thread 0 signal received.
watch_count: thread 0 count now = 137.
inc_count: thread 1, count = 138, unlocking mutex
inc_count: thread 2, count = 139, unlocking mutex
inc_count: thread 1, count = 140, unlocking mutex
inc_count: thread 2, count = 141, unlocking mutex
inc_count: thread 1, count = 142, unlocking mutex
inc_count: thread 2, count = 143, unlocking mutex
inc_count: thread 1, count = 144, unlocking mutex
inc_count: thread 2, count = 145, unlocking mutex
Main(): Waited on 3 threads. Done.
```

Semaphore

Semaphore

- Combines the functionalities of a mutex and a condition variable
- Can be shared across processes

Semaphore - features

- A combination of an unsigned integer **x**, a lock, and a condition variable
- **int sem_init(sem_t** *sem**, int** pshared**, unsigned int** value**)**;

```
x = value;
```

- **int sem_wait(sem_t** *sem**);**

```
if (x == 0)    {
    wait();    }
x--;
```

- **int sem_post(sem_t** *sem**);**

```
x++;
```

x is automatically protected by locking, and wait() is automatically managed with mechanisms similar to condition variable

Semaphore - replacing mutex

- Set the initial value to 1
- Lock: **sem_wait()**
- Unlock: **sem_post()**

```
void *child(void *arg) {
    int i;
    for (i = 0; i < 10000000; i++) {
    sem_wait(&mutex);
    counter++;
    sem_post(&mutex);
    }
    return NULL;
}
```

```
int main(int argc, char *argv[]) {
    sem_init(&mutex, 0, 1);
    pthread_t c1, c2;
    pthread_create(&c1, NULL, child, NULL);
    pthread_create(&c2, NULL, child, NULL);
    pthread_join(c1, NULL);
    pthread_join(c2, NULL);
    printf("result: %d (should be
20000000)\n", counter);
    return 0;
}
```

Semaphore - replacing condition variable

- Set the initial value to 0
- Wait: **sem_wait()**
- Signal: **sem_post()**

```c
void *child(void *arg) {
    sleep(2);
    printf("child\n");
    sem_post(&s); // signal here: child is done
    return NULL;
}
```

```c
int main(int argc, char *argv[]) {
    sem_init(&s, 0, 0);
    printf("parent: begin\n");
    pthread_t c;
    pthread_create(&c, NULL, child, NULL);
    sem_wait(&s); // wait here for child
    printf("parent: end\n");
    return 0;
}
```

Semaphore - throttling

- Control how many threads can access a resource / perform an action at the same time
- Initialize the value to the maximum concurrent accesses allowed

Frequently asked questions

Coding & Debugging

Why can’t my code be compiled?

- Inspect the error message. (Though not easy for C++)
- Are you using the wrong compiler / standard? (Use **gcc** for C and use **g++** for C++, use flags **-std=c++11** / **-std=c++14** / **-std=c99** …. as needed)
- Incompatible types
  - Confusing value types, pointers and references
  - Mixing up STL string (**std::string**), C string (**char***), character (**char**) and array of C strings (**char****)
  - Stick with the language you are familiar with

Coding & Debugging

Why can’t my code be linked?

- Check the command line arguments
  - Add **-c** for creating object files
  - **-o** should directly follow output file
  - One and only one source file should contain **main()**
- Mixing C and C++
  - Use **g++** to compile C code
  - Or, disable name mangling in C++ (https://eclass.srv.ualberta.ca/mod/page/view.php?id=3849191)

Coding & Debugging

Why do I encounter segmentation faults?

- Your code / the external function you called is accessing invalid memory
- Use **gdb** or **valgrind** to locate the cause

```
$ gcc dragonshell.c -g      (Add -g to compiler options)
$ gdb ./a.out               (Open gdb)
(gdb)  run                  (run your program in gdb)
……                          (some output)
Segmentation fault          (segmentation fault happened)
(gdb) where                 (locate the cause)
```

Coding & Debugging

Why is my code not working as expected?

- Should I use passing by value / passing by reference?
- Check variable scope
  - Anything between **{** and **}** is a block
  - Variables declared inside a blocked can not be accessed outside
- Check errors of system calls
- Add assertions

```
#include <cassert>
// ......
assert(x >= 0);
```

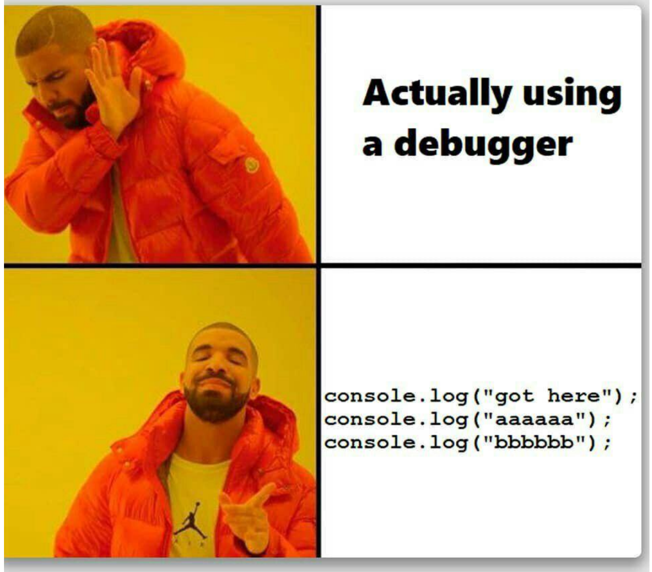
```
Assertion failed: (x < 0),
function main, file assert.c,
line 6.
```

Coding & Debugging

Why is my code not working as expected?

- Track the execution of your program line by line
- Or, print debugging messages 😜

  (although make them meaningful)



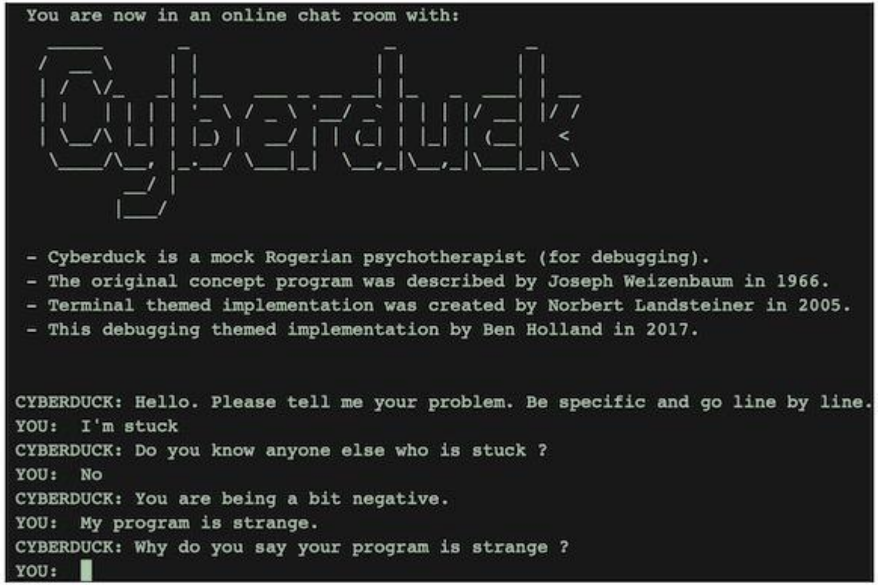
```
console.log("got here");
console.log("aaaaaa");
console.log("bbbbbb");
```

Credit: /u/Sh4moo @ reddit

Coding & Debugging

Why is my code not working as expected?

- Try to explain your code to yourself, your friend, or to a lovely rubber duck
- Find out more: https://rubberduckdebugging.com
- [Talk to a duck online]()
- Don’t make assumptions about how it behaves when reading through code



```
 You are now in an online chat room with:
   _____      _                  _            _
  / ____|    | |                | |          | |
 | |    _   _| |__   ___ _ __ __| |_   _  ___| | __
 | |   | | | | '_ \ / _ \ '__/ _` | | | |/ __| |/ /
 | |___| |_| | |_) |  __/ | | (_| | |_| | (__|   <
  \_____\__, |_.__/ \___|_|  \__,_|\__,_|\___|_|\_\
         __/ |
        |___/

 - Cyberduck is a mock Rogerian psychotherapist (for debugging).
 - The original concept program was described by Joseph Weizenbaum in 1966.
 - Terminal themed implementation was created by Norbert Landsteiner in 2005.
 - This debugging themed implementation by Ben Holland in 2017.


CYBERDUCK: Hello. Please tell me your problem. Be specific and go line by line.
YOU:  I'm stuck
CYBERDUCK: Do you know anyone else who is stuck ?
YOU:  No
CYBERDUCK: You are being a bit negative.
YOU:  My program is strange.
CYBERDUCK: Why do you say your program is strange ?
YOU:
```

Valgrind

Do I need to check for memory leaks?

- Yes, because
- It counts towards your marks
- It helps you resolve your bugs

Valgrind

Why can’t I get rid of “still reachable” leaks?

- You (or some libraries) hold pointers to some memory that could be freed
- C++ STL / runtime will use memory pools to allocate extra memory ahead of time. They are <u>not</u> counted as memory leaks in our assignments. Check the Valgrind report to see where the memory was allocated.

Resources

Where can I find tutorials for beginners on pthread (and other topics)?

- Operating Systems: Three Easy Pieces

- Advanced Programming in a Unix Environment

Assignment 2

Can I modify threadpool.h?

- Yes you can freely change type definitions and include other header files
- If you change function signatures, you have to justify your modifications in the design document
- We won’t test your thread pool. But your MapReduce library must use your own thread pool implementation

Assignment 2

Can I modify mapreduce.h?

- No and you may need to make sure the original **distwc.c** works

Assignment 2

Do I need to handle errors?

- You are encouraged to do so, but it won’t be tested (unless mentioned in the starter code or description)
- If you have a way to handle errors, your may briefly describe them in your design documents

More questions?