CS 35L

Software Construction Laboratory

Lecture 7.1

19th February, 2019

Logistics

- Hardware requirement for Week 8
  - Seeed Studio BeagleBone Green Wireless Development Board
- Presentations for Assignment 10
  - https://docs.google.com/spreadsheets/d/1o6r6CKCaB2du3klPflHiquymhBvbn7oP0wkHHMz_q1E/edit?usp=sharing
- Assignment 6 is due on 24th Feb, 2018 at 11:55pm

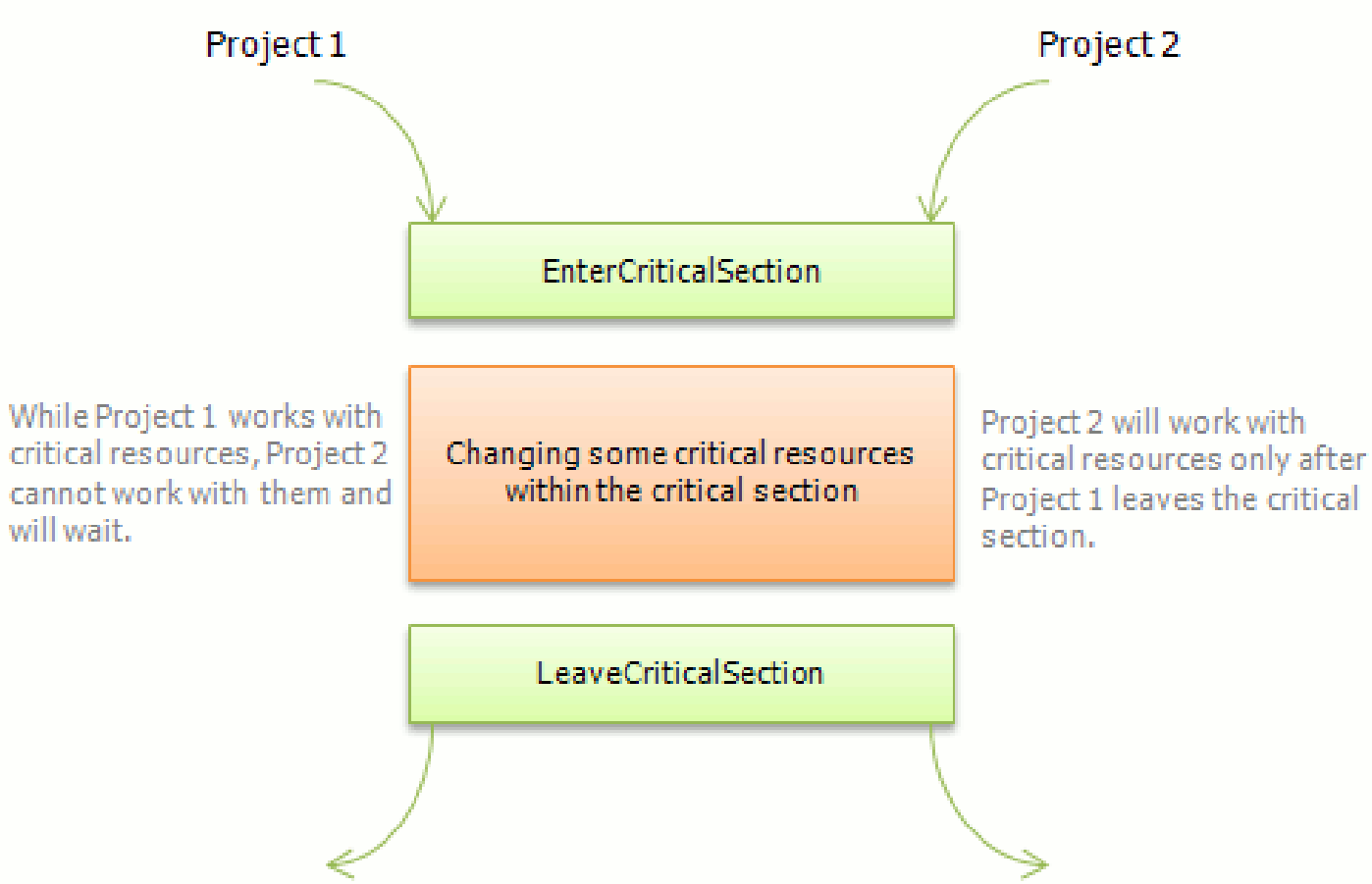
Review - Previous Lab

- Parallelism
- Uniprocessors and Multiprocessors
- Thread
- Multithreading
- Race Condition

Multi Threading

How to deal with Race Condition?

- Critical Section

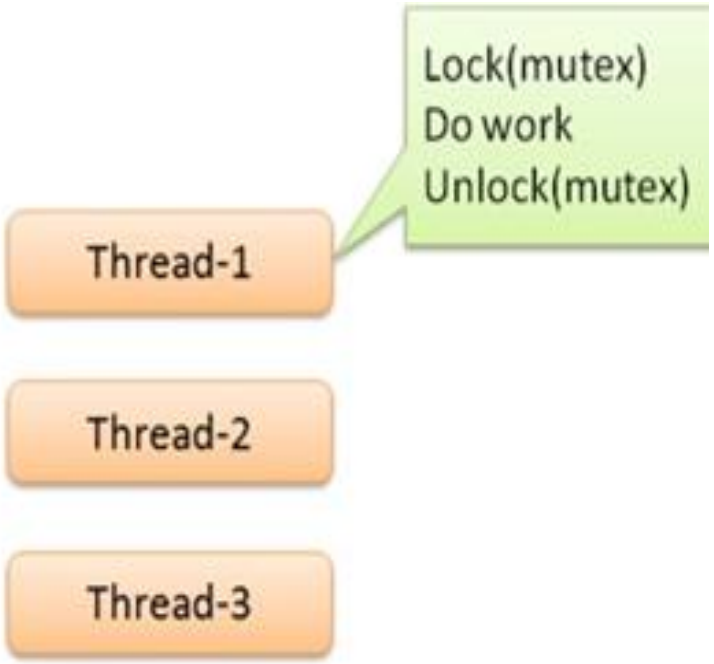


Mutex

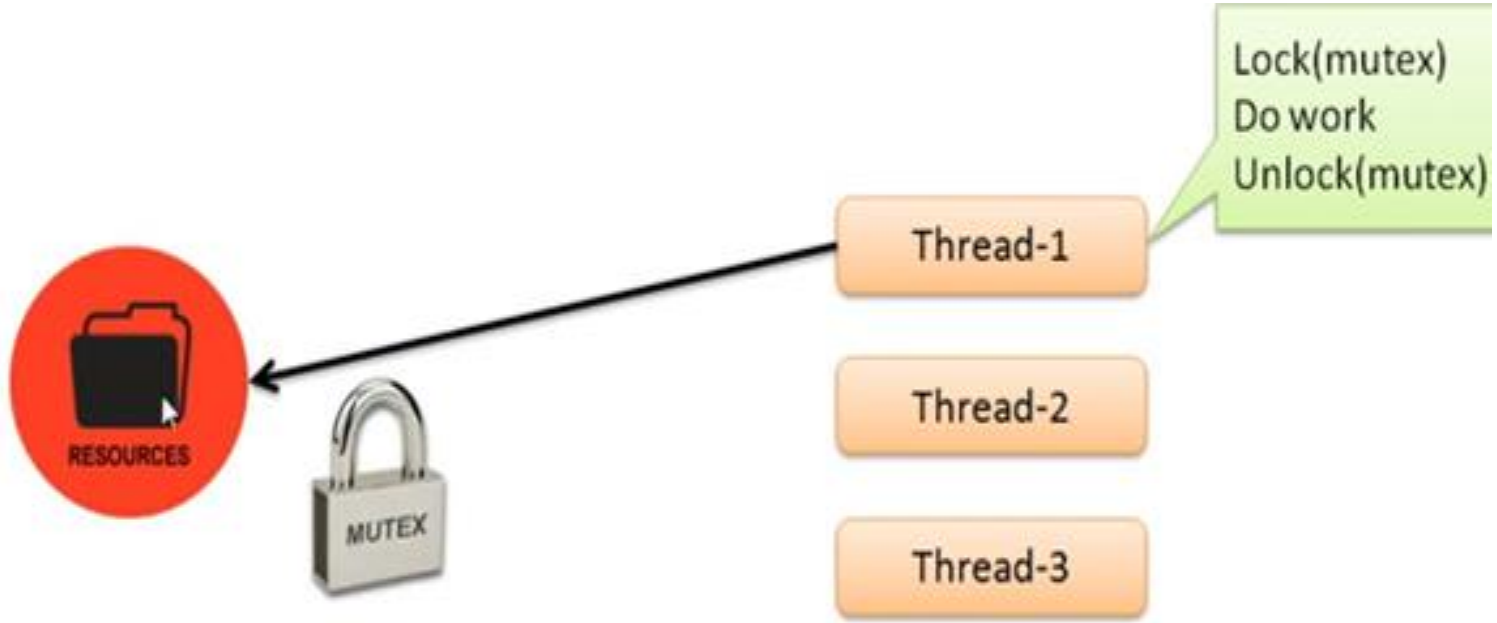
- Mutex is an object which allows only one thread into a critical section
- Mutex is owned by a thread - Ownership
- It forces other threads which attempt to gain access to that section, to wait until the first thread has exited from the section
- Each resource has a mutex

Mutex

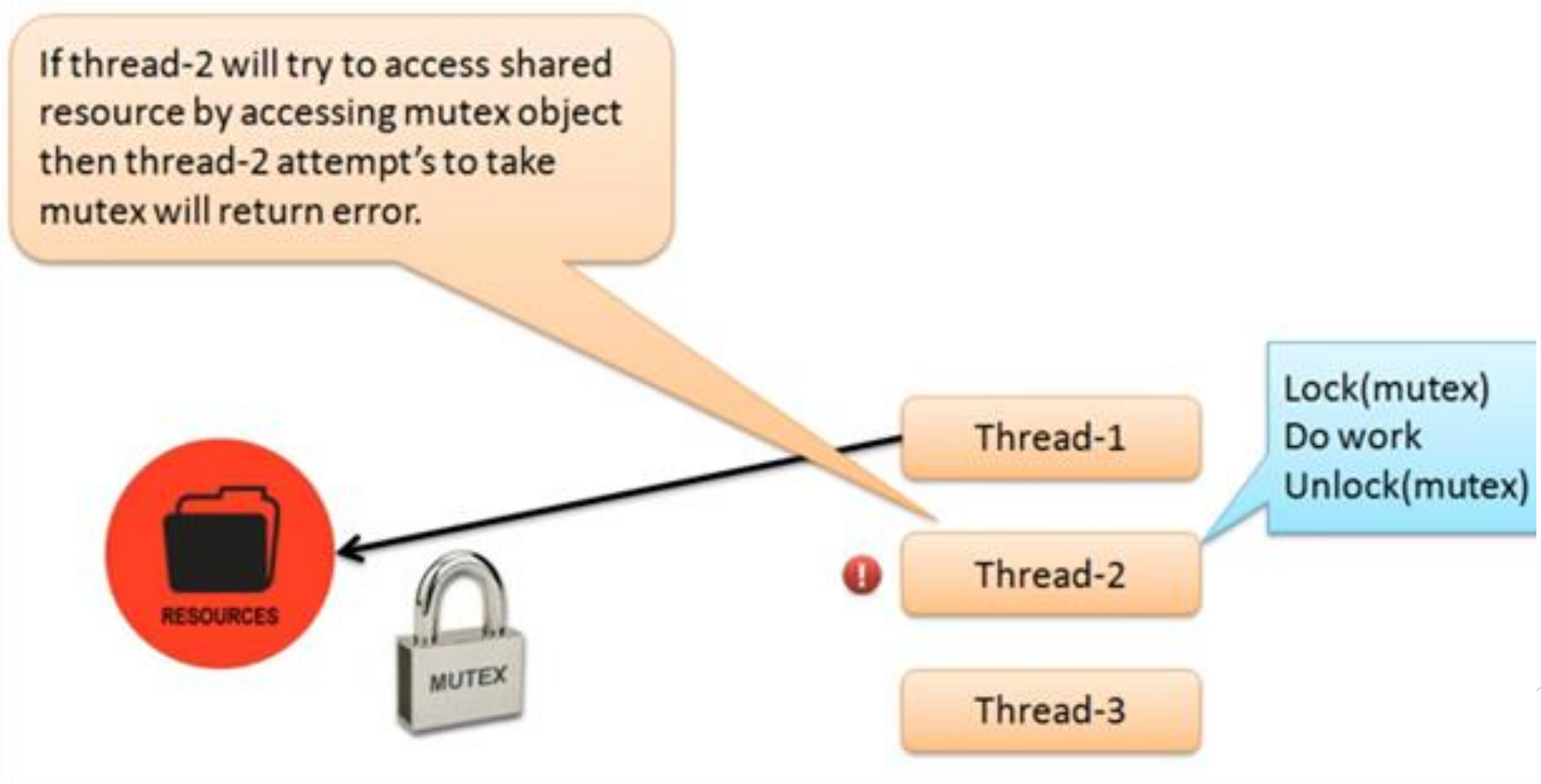

RESOURCES


Lock(mutex)
Do work
Unlock(mutex)
Thread-1
Thread-2
Thread-3

Mutex


Lock(mutex)
Do work
Unlock(mutex)
Thread-1
Thread-2
Thread-3
RESOURCES
MUTEX

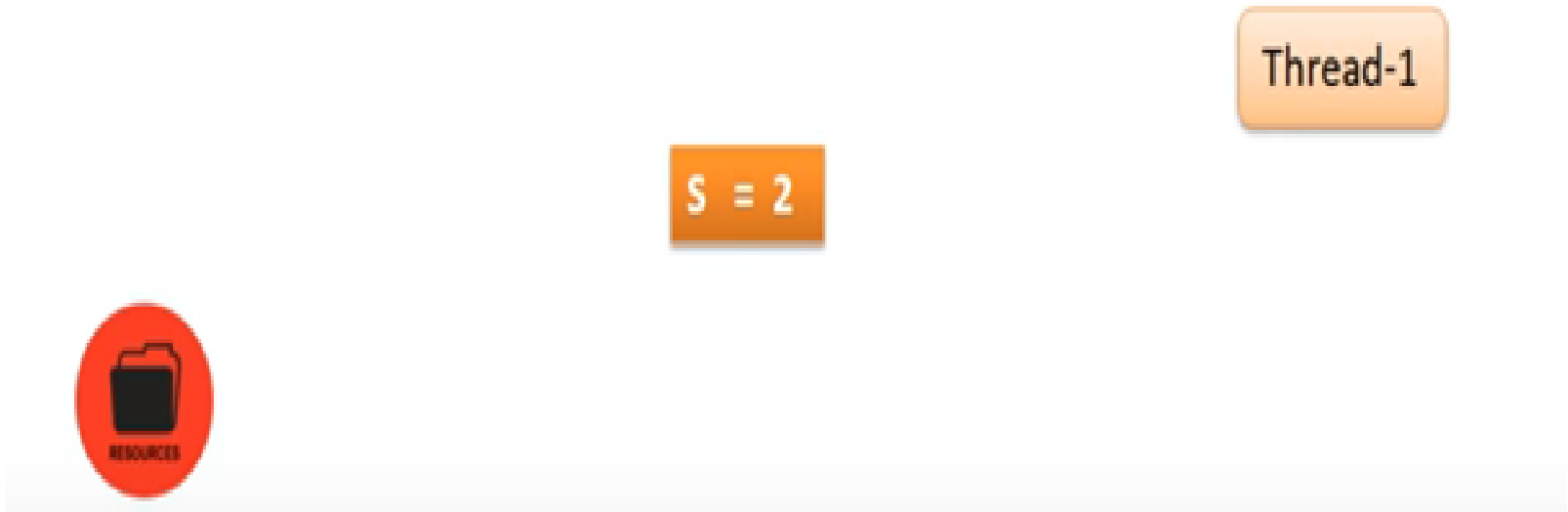
Mutex


If thread-2 will try to access shared resource by accessing mutex object then thread-2 attempt's to take mutex will return error.
Thread-1
Thread-2
Thread-3
Lock(mutex)
Do work
Unlock(mutex)
RESOURCES
MUTEX

Semaphores

- A semaphore is a variable that each process can check and then change.
  - Counting Semaphores and Binary Semaphores
- It's a signaling mechanism
- restricts/allows the number of simultaneous threads of a shared resource upto a maximum number
- threads can request access to a resource (decrements the semaphore)
- threads signal that the have finished using the resource (increments the semaphore)

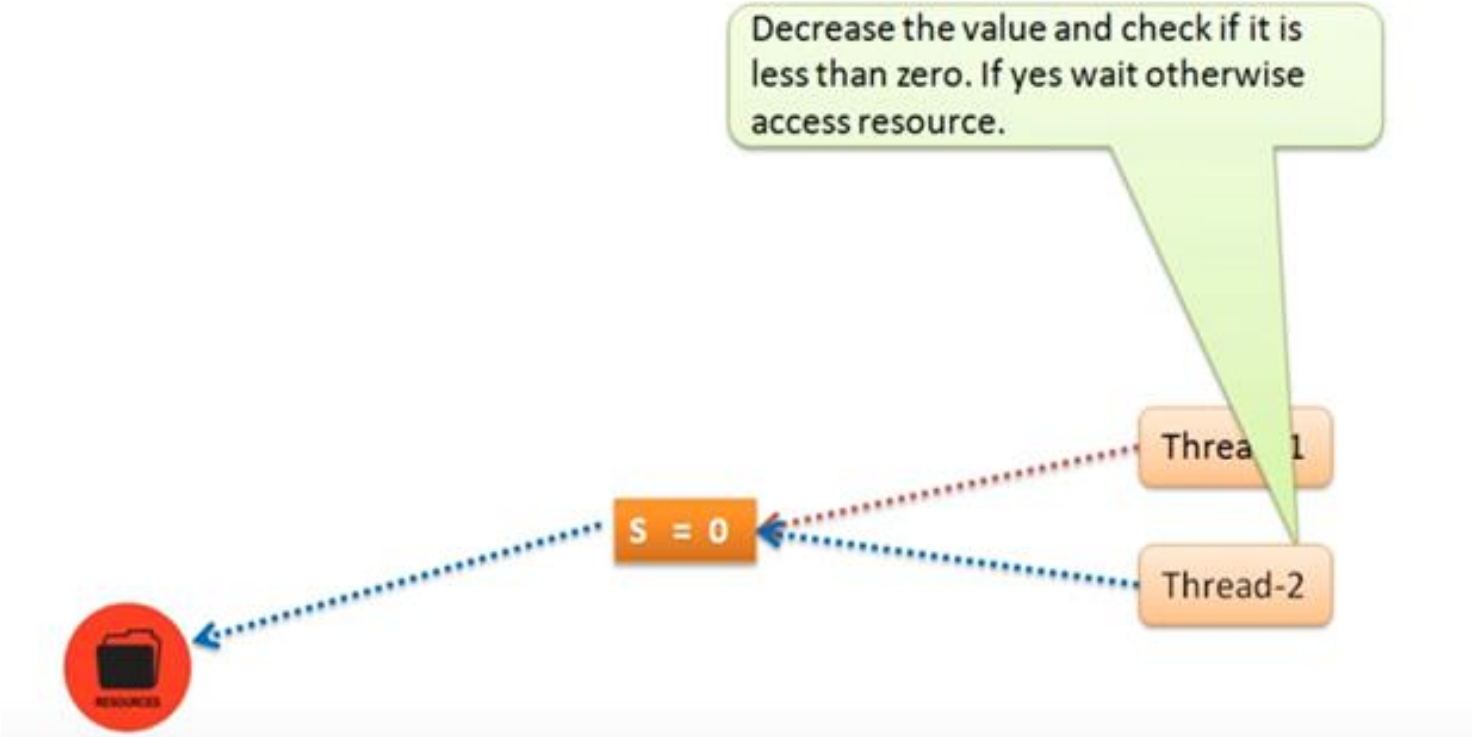
Semaphores



Semaphores

Semaphores

Semaphores


Decrease the value and check if it is less than zero. If yes wait otherwise access resource.
Thread-1
S = 0
Thread-2

Semaphores vs. Mutex

- Semaphore allows multiple program threads to access the finite instance of resources.
- On the other hand, Mutex allows multiple program threads to access a single shared resource but one at a time.

POSIX Threads

- import the pthread library
  - Example: #include<pthread.h>
- Use -pthread while compiling
- Represented by pthread_t (datatype)

Basic pthread functions

There are 5 basic pthread functions:

- pthread_create: creates a new thread within a process
- pthread_exit: terminates the currently running thread

- pthread_join: waits for another thread to terminate
- pthread_equal: compares thread ids to see if they refer to the same thread
- pthread_self: returns the id of the calling thread

Pthread_create

- Creates a new thread and makes it executable
- Can be called any number of times from anywhere within code
- Return value:
  - Success: zero
  - Failure: error number

Parameters

- int pthread_create(pthread_t *tid, const pthread_attr_t *attr, void *(my_function)(void *), void *arg);
- tid: unique identifier for newly created thread
- attr: object that holds thread attributes (priority, stack size, etc.)
  - Pass in NULL for default attributes
- my_function: function that thread will execute once it is created
- arg: a single argument that may be passed to my_function
  - Pass in NULL if no arguments

Pthread_create Example

```
#include <pthread.h> …
void *printMsg(void *thread_num) {
        int t_num = (int) thread_num;
        printf("It's me, thread %d!\n", t_num);
Return NULL;
 }
int main() {
        pthread_t tids[3];
        int t;
        for(t = 0; t < 3; t++) {
                int ret = pthread_create(&tids[t], NULL, printMsg, (void *) t);
                if(ret) {
                        printf("Error creating thread. Error code is %d\n", ret");
                        exit(-1); }
        }
}
```

Possible problem with this code? - If main thread finishes before all threads finish their job -> incorrect results

Pthread_join

- Function: makes originating thread wait for the completion of all its spawned threads' tasks
- Without join, the originating thread would exit as soon as it completes its job
  - A spawned thread can get aborted even if it is in the middle of its chore
- Return value:
  - Success: zero
  - Failure: error number

Parameters

- int pthread_join(pthread_t tid, void **status);
- tid: thread ID of thread to wait on
- status: the exit status of the target thread is stored in the location pointed to by *status
  - Pass in NULL if no status is needed

Assignment 6 - Lab

- Evaluate the performance of multithreaded sort
- Add /usr/local/cs/bin to PATH
  - $ export PATH=/usr/local/cs/bin:$PATH
- Generate a file containing 10M random single-precision floating point numbers, one per line with no white space
  - /dev/urandom: pseudo-random number generator
  - Od –An –t fF –N size < /dev/urandom
  - Research on the above options
- Disk quota exceeded
  - http://www.seasnet.ucla.edu/seasnet-account-quotas/

Assignment 6 - Lab

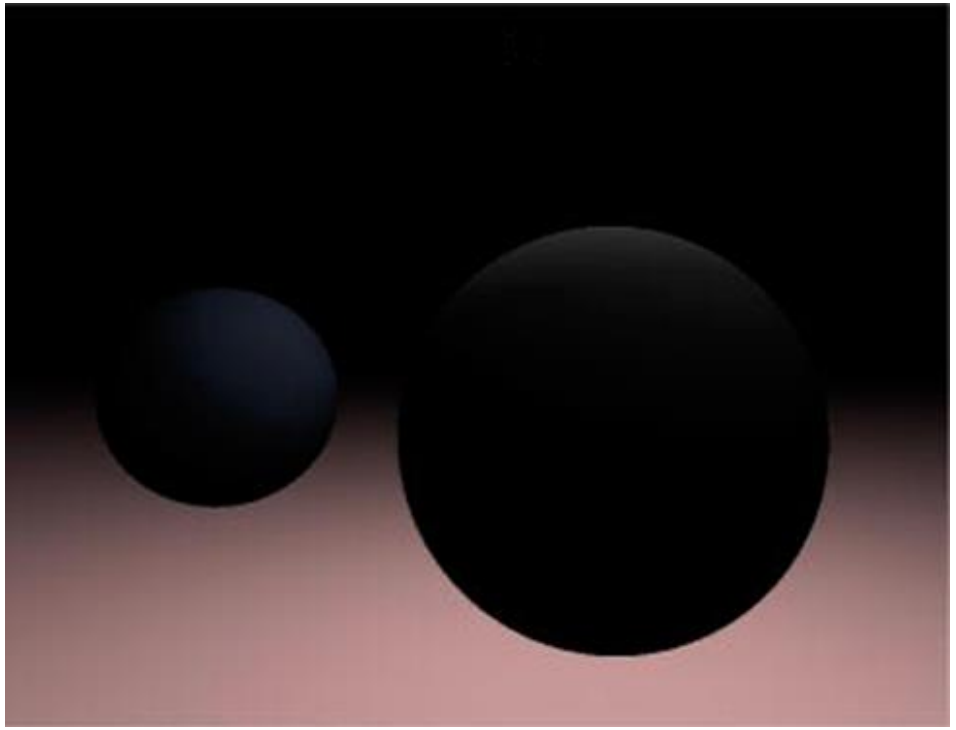
- od
  - write the contents of its input files to standard output in a user-specified format
  - Options
    - -t fF: single precision floating point
    - -N <count>: Format no more than count bytes of input
- sed, tr
  - Remove address, delete spaces, add newlines between each float instead of ‘ ‘

Assignment 6 - Lab

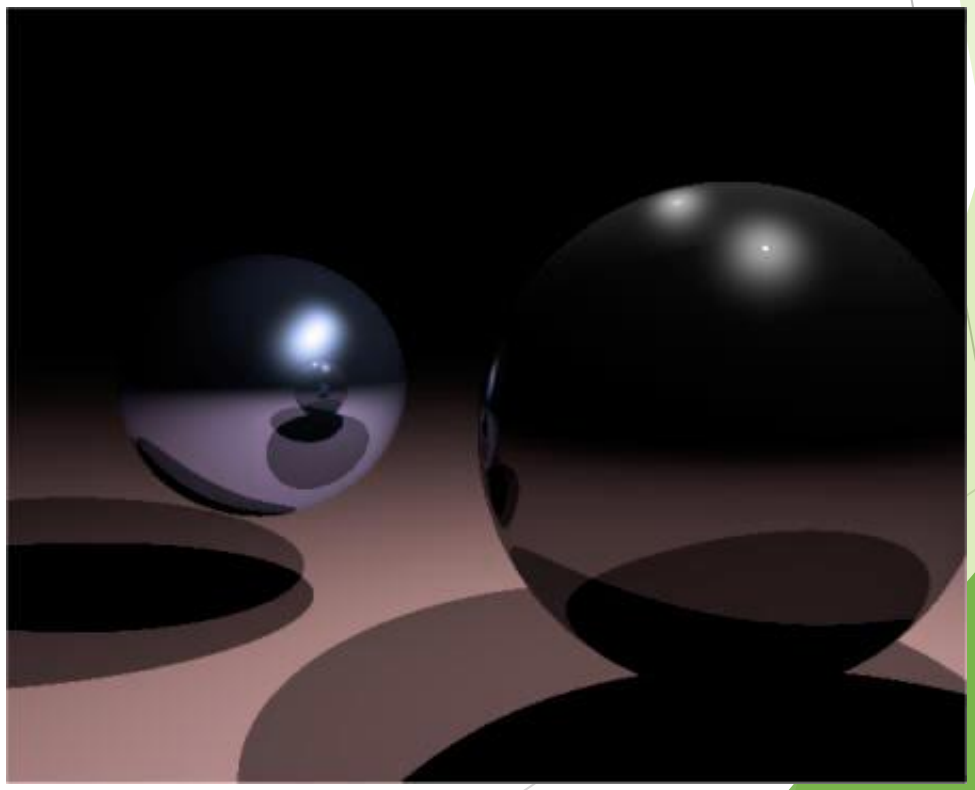
- use time -p to time the command sort -g on the data you generated
- Send output to /dev/null (to dispose unwanted data streams)
- Run sort with the --parallel option and the
  - -g option: compare by general numeric value
  - Use time command to record the real, user and system time when running sort with 1, 2, 4, and 8 threads
    - $ time -p sort -g file_name > /dev/null (1 thread)
    - $ time -p /usr/local/cs/bin/sort -g --parallel=[1, 2, 4, or 8] file_name > /dev/null
  - Record the times and steps in log.txt

Ray Tracing

- An advanced computer graphics technique for rendering 3D images
- Mimics the propagation of light through objects
- Simulates the effects of a single light ray as it’s reflected or absorbed by objects in the images



Without ray tracing



With ray tracing

Computational Resources

- Ray Tracing produces a very high degree of visual realism at a high cost (yields high quality rendering)
- The algorithm is computationally intensive
- Good candidate for multithreading (embarrassingly parallel)
- Threads need not synchronize with each other, because each thread works on a different pixel

Presentations

- Today's Presentation:
  - Robert Minahan
  - Rio Sonoyama
- Next up:
  - Yufei Wang
  - Calvin Chen

Questions?