Project 3 (100 points)
Assigned: Tuesday, January 30, 2018
Checkpoint: Wednesday, February 7, 2018
Due: Wednesday, February 14, 2018

Project #3
Threads and Synchronization

Professor Hugh C. Lauer
CS-3013 — Operating Systems

(Slides include copyright materials from *Operating Systems: Three Easy Step*, by Remzi and Andrea Arpaci-Dusseau, from *Modern Operating Systems*, by Andrew S. Tanenbaum, 3rd edition, and from other sources)

Objective

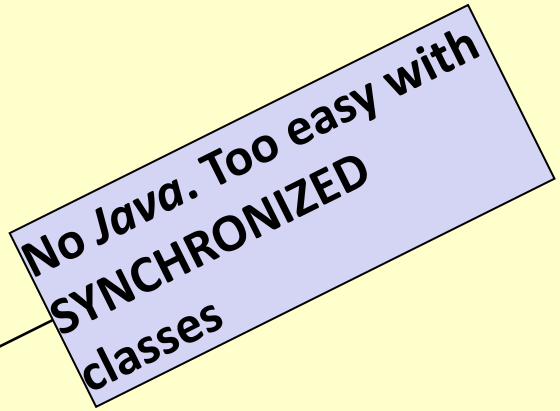
- To learn to use threads and synchronization mechanisms in user space
- To implement a multi-threaded program that aggressively pounds on a shared resource

The problem

- **Model a communal bathroom used by both sexes**
- **Possible states of the bathroom:–**
  - *Vacant*
  - *Occupied by women*
  - *Occupied by men*
- **Rules:–**
  - Anyone may enter if *vacant*
  - A user may enter if occupied by same sex
    - Any number of same sex may be in bathroom at same time.
  - A user must wait until *vacant* if occupied by opposite sex
    - Any number of users may by waiting at same time.

Problem #51, p. 174 of "Modern Operating Systems," 3rd ed., Andrew S. Tanenbaum

Implementation



- **Control module in *C***
  - Using **`pthread`** synchronization functions
  - To model the bathroom itself
- **Multi-threaded Test Program**
  - One thread per simulated person
  - Operate for simulated time
  - Random attempts to access and stay in bathroom

Implementation

- **Control module:–**
  - A `.h` file and a one or more `.c` files
- **Interface:–**

```
enum gender = {male, female};
void Enter(gender g);
void Leave(void);

void Initialize(...);
void Finalize(...);
```

- **Must correctly maintain:–**
  - State
  - # of users in bathroom
  - Gender of users in bathroom

Should be named bathroom.h and bathroom.o

Implementation

- **Control module:–**
  - A `.h` file and a one or more `.c` files
- **Interface:–**

```
enum gender = {male, female};
void Enter(gender g);
void Leave(void);

void Initialize(...);
void Finalize(...);
```

Waits until *vacant* if occupied by opposite gender

Sets sta

the last user to leave

Notifies *all* waiting threads that they can proceed

- **Must correctly maintain:–**
  - State
  - # of users in bathroom
  - Gender of users in bathroom

Multiple threads trying to access or change at same time!

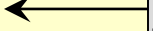
Multi-threaded test program

- **`main()` function is *master thread***
  - Interprets **`argc`** and **`argv`**
  - Spawns *n* "user" threads representing individuals
    - Randomly sets *gender, loop count* of each user thread
    - Specifies mean *arrival time* and mean *stay time*
  - Waits for user threads to finish
  - Prints summary
  - Exits cleanly

With no outstanding user threads!

User thread

- **Loops *loop count* times**
  - A random number
- **For each iteration:–**
  - Generate random *arrival* and *stay* times based on means
  - Sleep *arrival* units of time
  - Invoke **`Enter(gender)`**
    - May wait a long time if occupied by opposite gender
  - Sleep *stay* units of time
  - Invoke **`Leave()`**
  - Collect statistics
- **Print statistics (min, max, average wait time to enter)**
- **Exit cleanly**



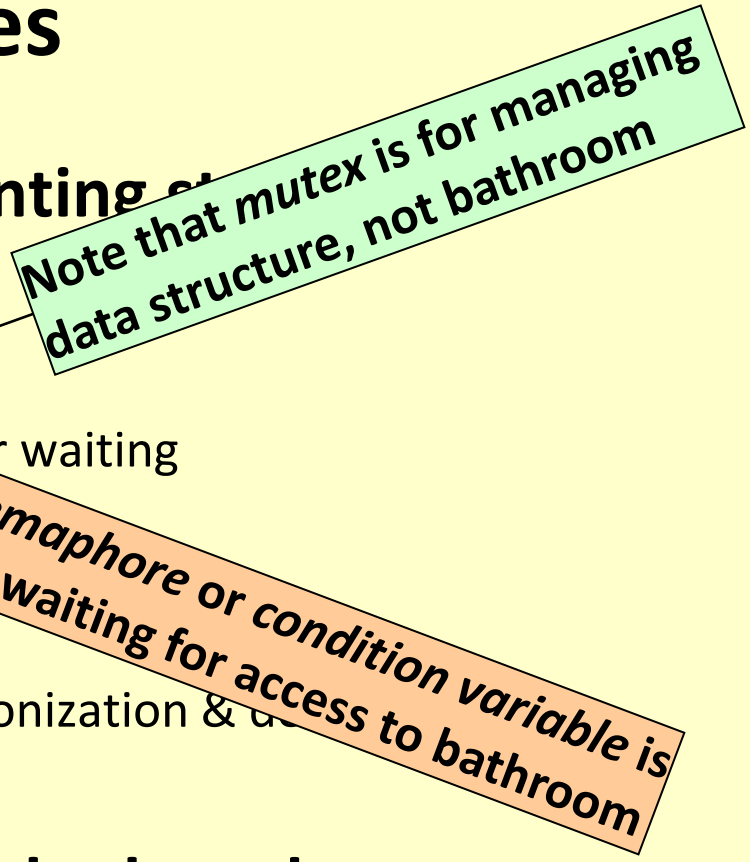
So that master thread knows that user thread is done

As simple as that!

With a few minor challenges!

Synchronization challenges

- **Shared data structure representing st… bathroom, count, etc.**
  - *Mutex* for protecting shared access
  - *Semaphore* or *condition variable* for waiting
- **Clean exit from user thread**
  - **`pthread_join`** or barrier synchronization & d…
- **Printing to `stdout` from multiple threads**
  - Without race conditions or getting mixed up.



Other challenges

- **Random number generation**
  - Loop count (per thread)
  - Arrival interval, stay time per iteration of thread
- **Normal distribution**
  - Box-Muller transform — uses two random #'s
  - Use ½ "standard deviation" for width
- **Simulating units of time**
  - #include <usleep.h>
  - Suggest 1or 10 milliseconds for one time unit.

Write-up

- **Explain *invariant* of shared data structure**
  - Show that **`Enter()`** and **`Leave()`** preserve invariant
  - Or show states & state transition diagram
- **Explain master thread and user thread relationship**
  - Creation and termination
- **Analysis of test cases**
  - Three with different parameters

Individual or 2-person team project …

- **You may discuss algorithms, strategies, etc., with each other and other teams**
- **You *should* share `bathroom.h` file for bathroom interface**
- **You are *strongly encouraged* to help each other & swap test programs**
  - I.e., run your **`bathroom.o`** with your friend's master program
  - Run your master program with your friend's **`bathroom.o`**
- **Must write individual/team code**
  - Own style, own words, own documentation, etc.

Due Date

- **Project due on Wednesday, February 14, 2018, 11:59 PM**
  - Checkpoint Wednesday, February 7, 2018, 11:59 PM
- **Submit via *InstructAssist***
  - *Project3* — **Project3_studentname.zip** or **Project3_teamname.zip**
  - Zip all files together!
- **Report to instructor or TAs any difficulties**
- **This project is worth 100 points!**

More on multi-threaded test program

- **Command line format specified in handout**
  - So graders can run your program with same arguments
- **Random number generator**
  - Uniform distribution:– `rand()`, `drand48()`
    - Issues of reproducibility in multi-threaded environment
  - Normal distribution:– Box-Muller algorithm
    - See handout
- **Each thread:–**
  - Random number of iterations
  - Loop:– wait, try to enter bathroom, get into bathroom, stay, leave bathroom, repeat
  - Keep statistics

Seed!

Individual threads

- **Each iteration:–**
  - Wait random time (normal distribution)
  - Call **`Enter()`**
  - Once in bathroom, stay random time (normal distribution)
  - Call **`Leave()`**
- **When done, print**
  - Thread number
  - Gender and number of iterations
  - Minimum, average, and maximum wait times to enter
- **Exit cleanly**

Many threads

- **Hundreds (or more)**
  - Optional:– use **`attr`** argument of **`pthread_create`** to set stack size
  - Run in Zoo Lab or other machine with at least eight processors
    - i.e., four cores, two threads per core
  - Try for as many processors as possible
- **Exit cleanly**
  - Main thread prints final summary after all threads have exited

Questions?