ECE 391 Discussion Week 7

Announcements & Reminders

- MP2.2 due next Monday (Oct 15) at 6:00pm
  - Demo assignments will remain the same as MP2.1
  - ASK your grader for MP2.1 regrade (half points for any points lost in functionality) – this is your ONLY chance for a regrade
- Please submit your MP3 group information by Tonight(Oct 10)
  - Group name
  - Group member netids (there must be EXACTLY 4 people in your group)
- Exam1 Regrade is ready for pick up.

Octrees

- Algorithm to display images with a multitude of colors on devices that can only display a limited number of colors (color quantization)
- .photo files
  - Each pixel is 16-bits: RRRRRGGGGGGBBBBB
  - 1st 64/256 VGA palette colors already set up and used by game objects
  - Other 192 colors are for you to represent the room photos
- Use arrays, not a pointer-based data structure
- Use 64 colors for the 2nd level nodes and the remaining 128 to represent the nodes in the 4th level
- Don't leave "holes"!

Octrees (continued)

1. Count the number of pixels in each node at level 4 of your octree
2. Sort the level 4 nodes based on the count and select the most frequent 128
    a. Need to keep track of the original order. How?
3. Calculate the averages for red, green, and blue separately for the most frequent 128 level 4 nodes and assign them to the palette
    a. Note that red and blue are 5 bits while green is 6 bits!
    b. You should be able to figure out the VGA index from here
4. Repeat 1-3 for level 2 nodes
    a. Remember to remove the contribution of any pixels assigned to the level 4 nodes
    b. There's a more efficient way than just simply repeating steps 1-3 again
5. Finally, reassign the colors to each pixel of the room photo

TUX Controller Driver

- Enable control (read/write) of LED displays
- Report button presses and releases – be careful not to switch buttons!
- Handle device reset (save/restore the LED state!)
- Enable game play using the TUX
- Implement Ioctls
  - Do NOT implement TUX_READ_LED
- Testing
  - printk()
  - input.c

```
/* set to 1 and compile this file by itself to test functionality */
#define TEST_INPUT_DRIVER 0

/* set to 1 to use tux controller; otherwise, uses keyboard input */
#define USE_TUX_CONTROLLER 0
```

Synchronization

- Keyboard and Tux controller should both work at the same time
  - Update the same variables
  - Shared variables should be synced
- Look at status_thread() and show_status() to see how to properly integrate the TUX with the game

TUX Driver – Userspace Integration

Definitions

- Thread
  - An independent stream of instructions scheduled to run by the operating system
  - Exists within a process and uses the process resources
  - Scheduled within a process, depending on scheduling policy
  - Can be scheduled simultaneously across processors/cores
- pthreads
  - Threading implementation adhering to POSIX standards

Using pthreads

- Creation

```
int pthread_create(thread_id, attributes, thread_function, args);
```

- Termination

```
void pthread_cancel(thread_id);
```

- Refer to [1] for other functions – join, exit…
  Not required for this MP

Thread functions

```
void* foo(void * arg){

    while (1) {
        // execute code here
    }

}
```

- Return type is generic pointer. Can be casted to any variable type.
- Argument is generic pointer. Any object can be passed in (and casted)

Synchronizing threads

- Use a semaphore/mutex!

```
static pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

void* foo(void * arg){

    while (1) {
        pthread_mutex_lock(&lock);
        // critical section
        if (data_available) {
            // do something
        }
        pthread_mutex_unlock(&lock);
    }

}
```

BAD!!

Synchronizing threads

- Condition variables – Put thread to sleep until wake up signal received

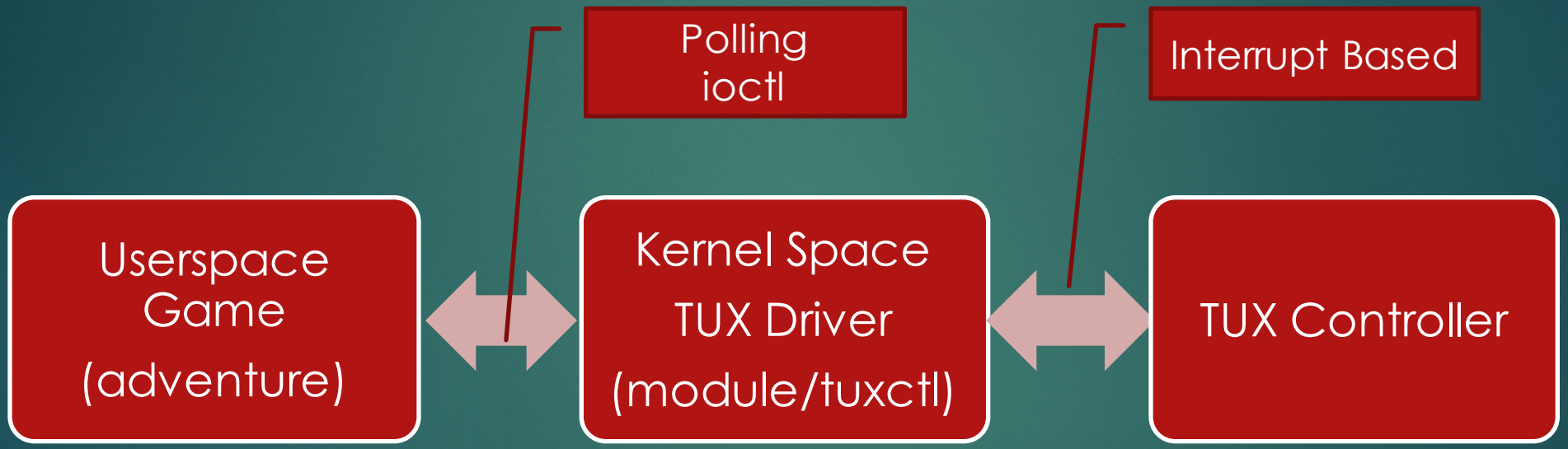
```
static pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
static pthread_cond_t  cv = PTHREAD_COND_INITIALIZER;

void* foo(void * arg){

    while (1) {
        pthread_mutex_lock(&lock);
        while (!data_available) {
            pthread_cond_wait(&cv, &lock);
        }
        // data_available, do something
        pthread_mutex_unlock(&lock);
    }
}
```

- When does the sleeping thread wakeup?
  - When signaling thread gives up lock
  - *pthread_cond_wait* reacquires lock before proceeding with thread

TUX Driver


Polling
ioctl
Interrupt Based
Userspace
Game
(adventure)
Kernel Space
TUX Driver
(module/tuxctl)
TUX Controller

Adding TUX support to Adventure

- Main game loop

```
game_loop() {

    while (1) {
        // display new image/room
        // update status message
        // sleep for 1.5s
        // check keyboard command
        // update game state
    }

}
```

Adding TUX support to Adventure

- Approach 1 – Use main thread

```
game_loop() {

    while (1) {
        // display new image/room
        // update status message
        // sleep for 1.5s

        ioctl(fd, TUX_BUTTONS, &buttons); // poll driver
        switch (buttons) { // update game state }

        // check keyboard command
        // update game state
    }

}
```

Adding TUX support to Adventure

- Approach 2 – Use new thread

```
game_loop() {} // no changes to game_loop

static pthread_t tux_tid;

void* tux_thread(void * arg){
    int buttons = 0;
    while (1) {
        ioctl(fd, TUX_BUTTONS, &buttons); // poll driver
        switch (buttons) { // update game state }
    }
}

main() {
     …
     pthread_create(tux_tid, NULL, tux_thread, NULL);
     …
}
```

Adding TUX support to Adventure

- Approach 3 – Use condition variables

```
game_loop() {
  while (1) {
    …
    // poll driver
    ioctl(fd, TUX_BUTTONS, &buttons);
    // determine if button pressed
    pthread_mutex_lock(&lock);
    if (buttons_pressed){
        pthread_cond_signal(&cv);
    }
    pthread_mutex_unlock(&lock);
    …
  }
}
```

```
void* tux_thread(void * arg){

  while (1) {
    pthread_mutex_lock(&lock);
    while (!buttons_pressed){
      pthread_cond_wait(&cv, &lock);
    }

    switch (buttons) {
      // update game state
    }
    pthread_mutex_unlock(&lock);
  }
  return NULL;
}
```

Exam Questions?

Reference

- [1] - https://computing.llnl.gov/tutorials/pthreads/