POSIX Threads

• Why Threads?
  – Latency Hiding / Multiprogramming (covered earlier)
  – Ease of Programming (covered now)

• POSIX Threads (Stevens, Chapter 11)
  – Thread Management
  – Thread Safety
  – Thread Attributes

POSIX Threads

• Why Threads?
  – Latency Hiding / Multiprogramming (covered earlier)
  – Ease of Programming (covered now)

• POSIX Threads (R&R, Chapter 12)
  – Thread Management
  – Thread Safety
  – Thread Attributes
Why Threads?

- Many interactive applications run in loops.
- For example, an interactive game.

```c
while (1) {
    /* Read Keyboard */
    /* Recompute Player Position */
    /* Update Display */
}
```


---

Why Threads?

- Many interactive applications run in loops.
- For example, an interactive game.

```c
while (1) {
    /* Synchronize to Highest Frequency */
    /* Read Keyboard */
    /* AND Read Mouse */
    /* Recompute Player Position */
    /* Update Display */
    /* AND emit sounds */
}
```

Why Threads?

- Many interactive applications run in loops.
- For example, an interactive game.
- It ain't over yet!
- What about compute-intensive operations, like AI, video compression?
- How about Signal Handlers?

Suddenly, application is getting complex!

...and much more:


Reading the Mouse

```c
while (1) {
    /* Synchronize to Highest Frequency */
    /* Read Keyboard */
    /* AND Read Mouse */
    /* Recompute Player Position */
    /* Update Display */
    /* AND all other lights */
    /* AND emit sounds */
    /* AND more sounds */
    /* AND move game physically */
}

read_mouse() {
}
```
Reading the Mouse (II)

while (1) {
    /* Synchronize to Highest Frequency */
    /* Read Keyboard */
    /* AND Read Mouse */
    /* Recompute Player Position */
    /* Update Display */
    /* AND all other lights */
    /* AND emit sounds */
    /* AND more sounds */
    /* AND move game physically */
}

Separate Process

int main() {
    handle_mouse();
}

Reading the Mouse (III)

while (1) {
    /* Synchronize to Highest Frequency */
    /* Read Keyboard */
    /* AND Read Mouse */
    /* Recompute Player Position */
    /* Update Display */
    /* AND all other lights */
    /* AND emit sounds */
    /* AND more sounds */
    /* AND move game physically */
}

Separate Thread

handle_mouse() {
The Thread and its Creation

```c
/* The Mouse Input Function */

void * handle_mouse() {
    char buf[BUFSIZE]; ssize_t nbytes;
    for (;;) {
        if ((nbytes = read_from_mouse(buf, BUFSIZE)) <= 0)
            break;
        dosomething_with(buf, nbytes);
    }
    return NULL;
}
```

```c
#include <pthread.h>

int error;
pthread_t tid;

if (error = pthread_create(&tid, NULL, handle_mouse, NULL))
    perror("Failed to create read_mouse thread");
```

while (1) {
    /* Synchronize to Highest Frequency */
    /* Read Keyboard */
    /* AND Read */
    /* Recompute P1 */
    /* Update Display */
    /* AND all */
    /* AND emit */
    /* AND more */
    /* AND move game physically */
    /* Read Mouse */
    /* AND Read Mouse */
    /* Recompute Player Position */
    /* Update Display */
    /* AND all other lights */
    /* AND emit sounds */
    /* AND more sounds */
    /* AND move game physically */
}
```
Thread Management

- `pthread_cancel`: (terminate another thread)
- `pthread_create`: (create a thread)
- `pthread_detach`: (have thread release res’s)
- `pthread_equal`: (two thread id’s equal?)
- `pthread_exit`: (exit a thread)
- `pthread_kill`: (send a signal to a thread)
- `pthread_join`: (wait for a thread)
- `pthread_self`: (what is my id?)

```c
int pthread_create(pthread_t * thread, const pthread_attr_t * attr, void *(*start_routine)(void *), void * arg)

fd = open("my.dat", O_RDONLY);
if (error = pthread_create(&t_id, NULL, processfd, &fd))
    fprintf(stderr, "Failed create thread: %s\n", strerror(error));
```
# Thread Attributes

| Attribute objects          | pthread_attr_destroy  
|                           | pthread_attr_init      |
| state                     | pthread_attr_getdetachstate  
|                           | pthread_attr_setdetachstate  |
| stack                     | pthread_attr_getguardsize  
|                           | pthread_attr_setguardsize  
|                           | pthread_attr_getstack      
|                           | pthread_attr_setstack      |
| scheduling                | pthread_attr_getinheritedsched  
|                           | pthread_attr_setinheritedsched  
|                           | pthread_attr_getschedparam  
|                           | pthread_attr_setschedparam  
|                           | pthread_attr_getschedpolicy  
|                           | pthread_attr_setschedpolicy  
|                           | pthread_attr_getscope       
|                           | pthread_attr_setscope       |

- **Detached** threads release resources when terminate.
- **Attached** states hold on to resources until parent thread calls `pthread_join`.

---

## Thread Attributes: State

| Attribute objects | pthread_attr_destroy  
|-------------------|-----------------------
|                   | pthread_attr_init      |
| state             | pthread_attr_getdetachstate  
|                   | pthread_attr_setdetachstate  |
| stack             | pthread_attr_getguardsize  
|                   | pthread_attr_setguardsize  
|                   | pthread_attr_getstack      
|                   | pthread_attr_setstack      |
| scheduling        | pthread_attr_getschedparam  
|                   | pthread_attr_setschedparam  
|                   | pthread_attr_getschedpolicy  
|                   | pthread_attr_setschedpolicy  
|                   | pthread_attr_getscope       
|                   | pthread_attr_setscope       |
### Thread Attributes: Stack

<table>
<thead>
<tr>
<th>Attribute Objects</th>
<th>pthread_attr_destroy</th>
<th>pthread_attr_init</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>pthread_attr_getdetachstate</td>
<td>pthread_attr_setdetachstate</td>
</tr>
<tr>
<td>Stack</td>
<td>pthread_attr_getguardsize</td>
<td>pthread_attr_setguardsize</td>
</tr>
<tr>
<td></td>
<td>pthread_attr_getstack</td>
<td>pthread_attr_setstack</td>
</tr>
</tbody>
</table>

- **setstack** defines location and size of stack.
- **setguardsize** allocates additional memory. If the thread overflows into this extra memory, an error is generated.

### Thread Attributes: Scheduling

<table>
<thead>
<tr>
<th>Attribute Objects</th>
<th>pthread_attr_destroy</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td></td>
</tr>
<tr>
<td>Stack</td>
<td></td>
</tr>
</tbody>
</table>

- **PTHREAD_INHERIT_SCHED** defines that scheduling parameters are inherited from parent thread. (as opposed to **PTHREAD_EXPLICIT_SCHED**).
- Scheduling policies: SCHED_FIFO, SCHED_RR, SCHED_SPORADIC, SCHED_OTHER, ...
- **contention scope** defines whether process competes at process level or at system level for resources.