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

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.

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

- Reference [B.O. Gallmeister, "POSIX.4, Programming for the Real World," O'Reilly&Assoc., Inc.]

Why Threads?

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

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

- Reference [B.O. Gallmeister, "POSIX.4, Programming for the Real World," O'Reilly&Assoc., Inc.]

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?

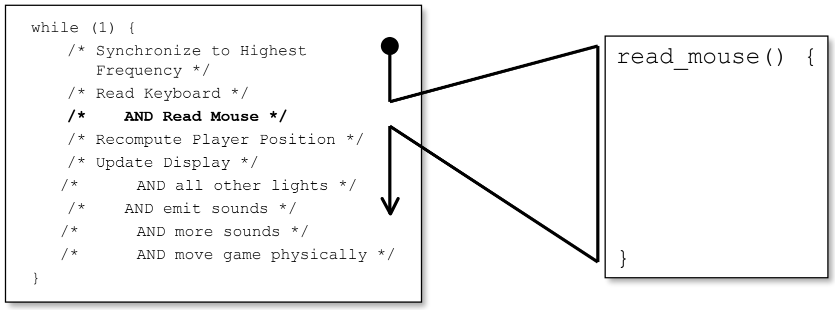
- How about Signal Handlers?

```
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 */
}
```

Suddenly, application is getting complex!

- Reference [B.O. Gallmeister, "POSIX.4, Programming for the Real World," O'Reilly&Assoc., Inc.]

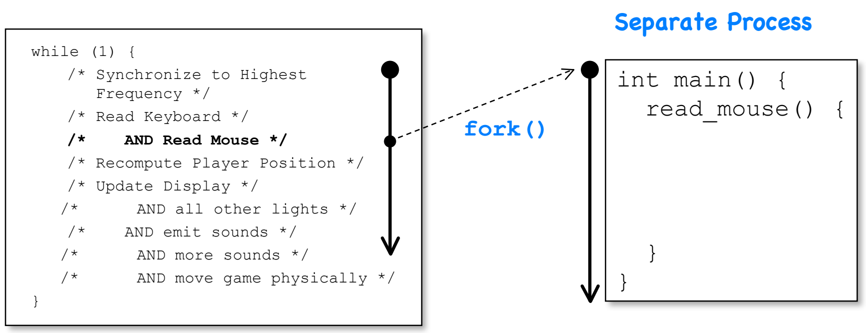
Reading the Mouse



```
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() {

}
```

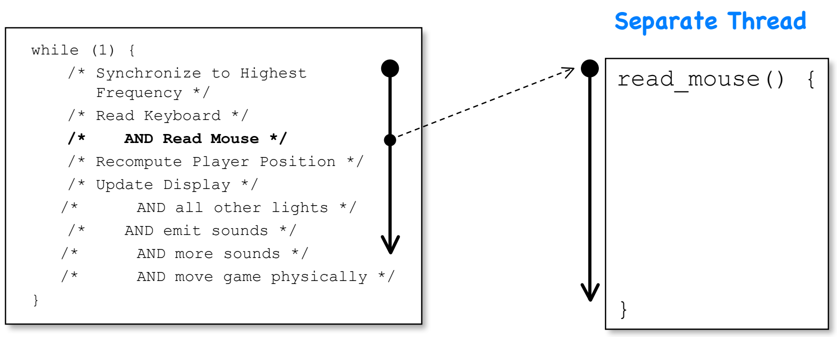
Reading the Mouse (II)



```
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 */
}
```

```
int main() {
   read_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 */
}
```

```
read_mouse() {




}
```

The Thread and its Creation

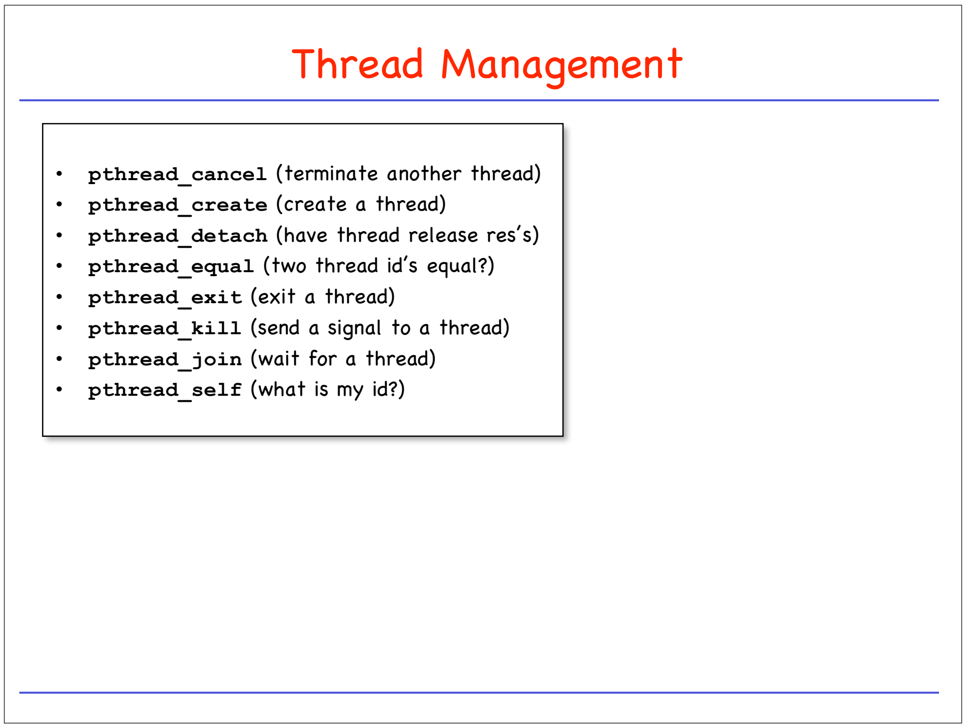
```
while (1) {
    /* Synchronize to 
       Frequency */
    /* Read Keyboard *
    /*    AND Read Mous
    /* Recompute Player
    /* Update Display 
   /*      AND all oth
    /*    AND emit sounds */
   /*      AND more sounds */
   /*      AND move game physically */
}
```

```
/* The Mouse Input Function */

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

The Thread and its Creation

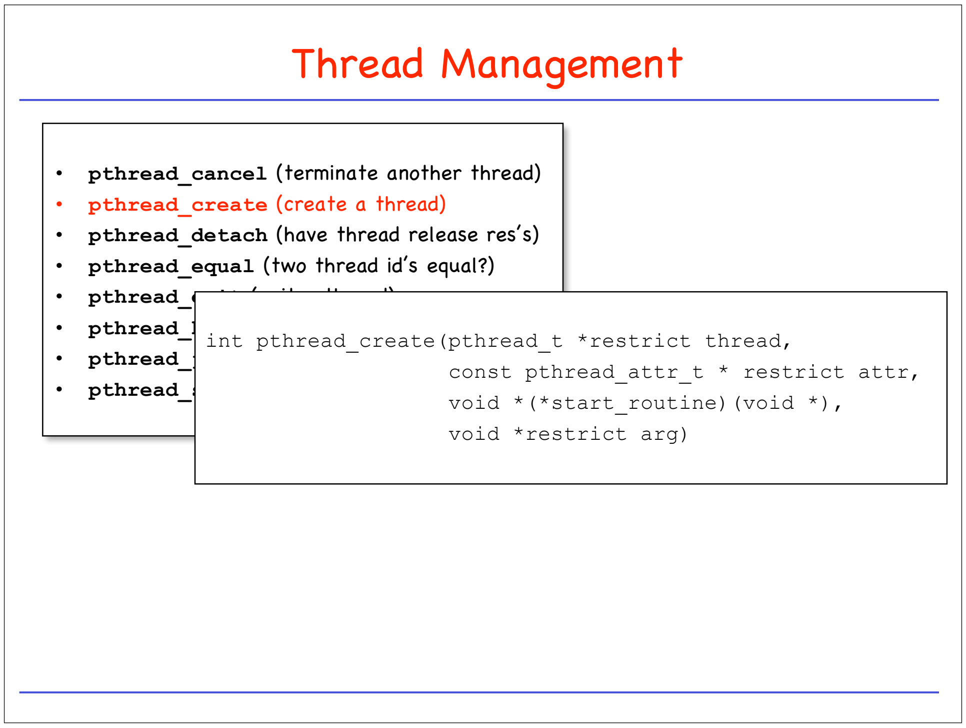
```
#include <pthread.h>

int error;
pthread_t tid;

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



while (1) {
    /* Synchronize to Highest
       Frequency */
    /* Read Keyboard */
    /*    AND Read Mouse */ <- Handled by separate thread!
    /* Recompute Player Position */
    /* Update Display */
   /*      AND all other lights */
    /*    AND emit sounds */
   /*      AND more sounds */
   /*      AND move game physically */
}
```

```
f,BUFSIZE)) <= 0)
```



Thread Management

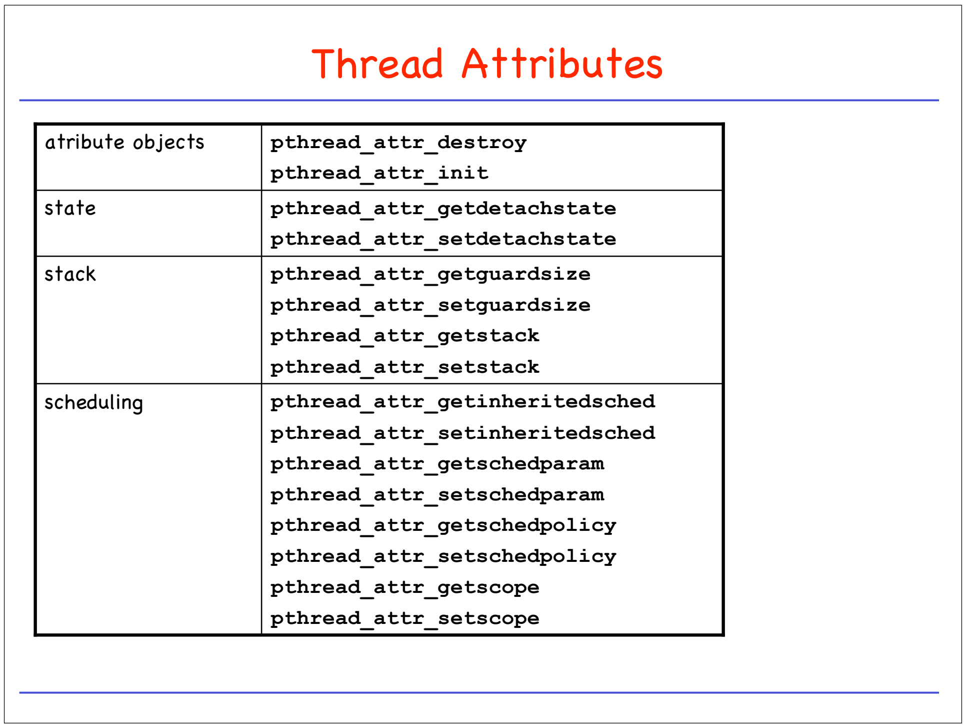
- **pthread_cancel** (terminate another thread)
- **pthread_create** (create a thread)
- **pthread_detach** (have thread release res's)
- **pthread_equal** (two thread id's equal?)
- **pthread_exit** (exit a thread)
- **pthread_kill** (send a signal to a thread)
- **pthread_join** (wait for a thread)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



Thread Management

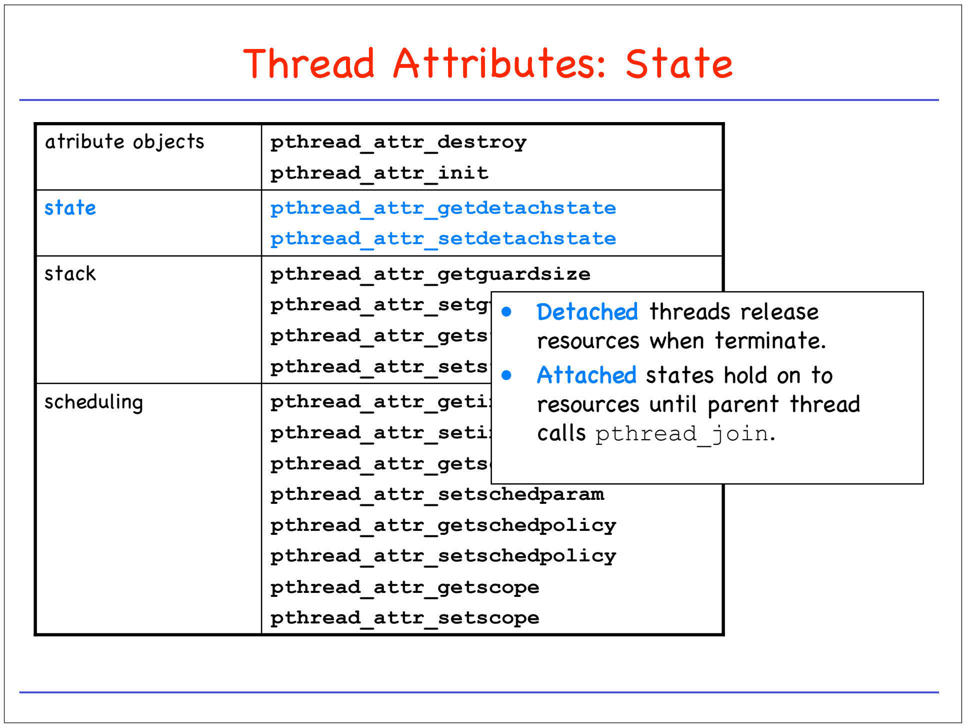
- **pthread_cancel** (terminate another thread)
- **pthread_create** (create a thread)
- **pthread_detach** (have thread release res's)
- **pthread_equal** (two thread id's equal?)
- **pthread_**
- **pthread_**
- **pthread_**
- **pthread_**

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



Thread Attributes

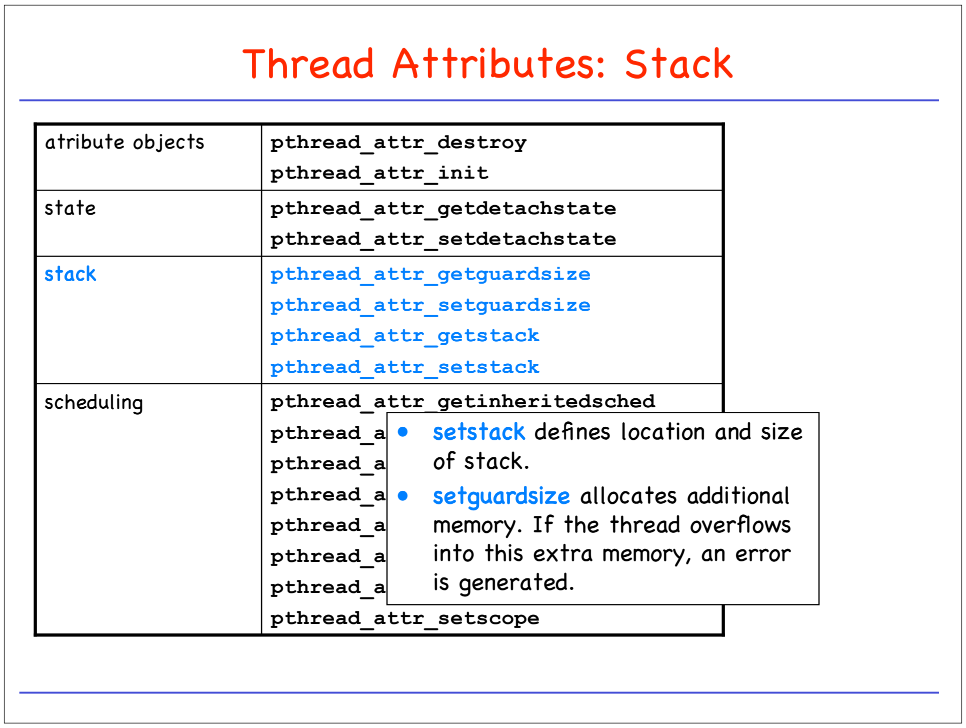
atribute objects	pthread_attr_destroy pthread_attr_init
state	pthread_attr_getdetachstate pthread_attr_setdetachstate
stack	pthread_attr_getguardsize pthread_attr_setguardsize pthread_attr_getstack pthread_attr_setstack
scheduling	pthread_attr_getinheritedsched pthread_attr_setinheritedsched pthread_attr_getschedparam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



Thread Attributes: State

atribute objects	pthread_attr_destroy pthread_attr_init
state	pthread_attr_getdetachstate pthread_attr_setdetachstate
stack	pthread_attr_getguardsize pthread_attr_setg pthread_attr_gets pthread_attr_sets
scheduling	pthread_attr_geti pthread_attr_seti pthread_attr_gets pthread_attr_setschedparam pthread_attr_getschedpolicy pthread_attr_setschedpolicy pthread_attr_getscope pthread_attr_setscope

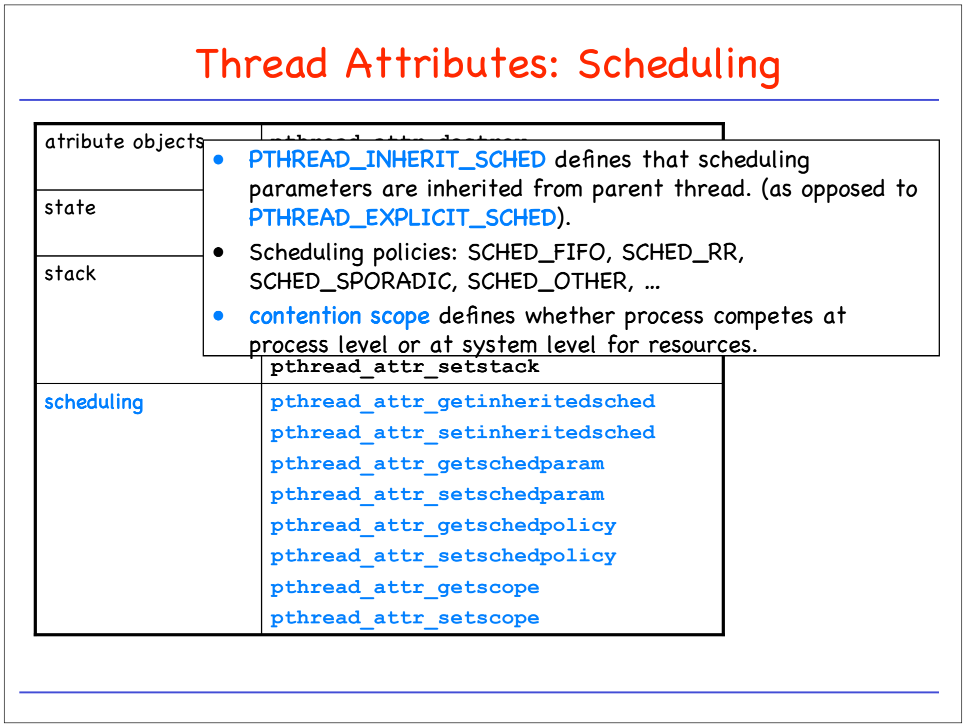
- Detached threads release resources when terminate.
- Attached states hold on to resources until parent thread calls `pthread_join`.



Thread Attributes: Stack

atribute 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_a pthread_a pthread_a pthread_a pthread_a pthread_a pthread_attr_setscope

- 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

atribute objects	pthread_attr_destroy
state	
stack	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

- 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.