CS 33

Multithreaded Programming V

How About This?

```
pthread_mutex_lock(&m);
if (++count == number) {
  pthread_cond_broadcast(&cond_var);
  count = 0;
} else while (!(count == number)) {
  pthread_cond_wait(&cond_var, &m);
}
pthread_mutex_unlock(&m);
```

And This ...

Quiz 1
Does it work?

a) definitely
b) probably
c) rarely
d) never

```
pthread_mutex_lock(&m);
if (++count == number) {
  pthread_cond_broadcast(&cond_var);
  count = 0;
} else {
  pthread_cond_wait(&cond_var, &m);
}
pthread_mutex_unlock(&m);
```

And This ...

Quiz 1
Does it work?

a) definitely
b) probably
c) rarely
d) never

```
pthread_mutex_lock(&m);
if (++count == number) {
  pthread_cond_broadcast(&cond_var);
  count = 0;
} else {
  pthread_cond_wait(&cond_var, &m);
}
pthread_mutex_unlock(&m);
```

Barrier in POSIX Threads

```
pthread_mutex_lock(&m);
if (++count < number) {
   int my_generation = generation;
   while(my_generation == generation) {
      pthread_cond_wait(&waitQ, &m);
   }
} else {
   count = 0;
   generation++;
   pthread_cond_broadcast(&waitQ);
}
pthread_mutex_unlock(&m);
```

More From POSIX!

```
int pthread_barrier_init(pthread_barrier_t *barrier,
      pthread_barrierattr_t *attr,
      unsigned int count);
int pthread_barrier_destroy(
      pthread_barrier_t *barrier);
int pthread_barrier_wait(
      pthread_barrier_t *barrier);
```

Why *cond_wait* is Weird …

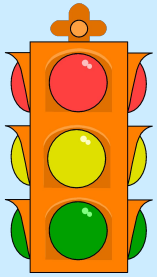
```
pthread_cond_wait(pthread_cond_t *c, pthread_mutex_t *m) {
    pthread_mutex_unlock(m);
    sem_wait(c->sem);
    pthread_mutex_lock(m);
}

pthread_cond_signal(pthread_cond_t *c) {
    sem_post(c->sem);
}
```

Deviations

- Signals

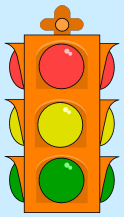
vs

.



- Cancellation
  - tamed lightning

Signals

- 
  - who gets them?
  - who needs them?
- 
  - how do you respond to them?

Dealing with Signals

- Per-thread signal masks
- Per-process signal vectors
- One delivery per signal

Signals and Threads

```
int pthread_kill(pthread_t thread, int signo);
```

- thread equivalent of *kill*

```
int pthread_sigmask(int how,
      const sigset_t *newmask,
      sigset_t oldmask);
```

- thread equivalent of *sigprocmask*

Asynchronous Signals (1)

```
int main( ) {
   void handler(int);
   signal(SIGINT, handler);

      ...

}

void handler(int sig) {
   ...
}
```


Asynchronous Signals (2)

```
int main( ) {
    void handler(int);

    signal(SIGINT, handler);

    ...    // complicated program

    printf("important message: "
           "%s\n", message);

    ...    // more program

}
```

```
void handler(int sig) {

    ...    // deal with signal

    printf("equally important "
           "message: %s\n",
      message);
}
```

Quiz 1

```
int main( ) {
    void handler(int);

    signal(SIGINT, handler);

    ...    // complicated program

  pthread_mutex_lock(&mut);
    printf("important message: "
           "%s\n", message);
  pthread_mutex_unlock(&mut);

    ...    // more program

}
```

```
void handler(int sig) {

    ...    // deal with signal

  pthread_mutex_lock(&mut);
    printf("equally important "
           "message: %s\n",
  message);
  pthread_mutex_unlock(&mut);
}
```

Does this work?

a) yes
b) no

Synchronizing Asynchrony

```
computation_state_t   state;
sigset_t set;
int main( ) {
    pthread_t thread;

    sigemptyset(&set);
    sigaddset(&set, SIGINT);
    pthread_sigmask(SIG_BLOCK,
        &set, 0);
    pthread_create(&thread, 0,
        monitor, 0);
    long_running_procedure( );
}
```

```
void *monitor(void *dummy) {
    int sig;
    while (1) {
        sigwait(&set, &sig);
        display(&state);
    }
    return(0);
}
```

Some Thread Gotchas ...

- Exit vs. pthread_exit
- Handling multiple arguments

Worker Threads

```
int main() {
  pthread_t thread[10];
  for (int i=0; i<10; i++)
    pthread_create(&thread[i], 0,
        worker, (void *)i);
  return 0;
}
```

Better Worker Threads

```
int main() {
  pthread_t thread[10];
  for (int i=0; i<10; i++)
    pthread_create(&thread[i], 0,
        worker, (void *)i);
  pthread_exit(0);
}
```

Multiple Arguments

```
void relay(int left, int right) {
  pthread_t LRthread, RLthread;

  pthread_create(&LRthread,
      0,
      copy,
      left, right);                    // Can't do
  this ...
  pthread_create(&RLthread,
      0,
      copy,
      right, left);                    // Can't do
  this  ...
}
```

Multiple Arguments

```
typedef struct args {
  int src;
  int dest;
} args_t;

void relay(int left, int right) {
  args_t LRargs, RLargs;
  pthread_t LRthread, RLthread;
  ...
  pthread_create(&LRthread, 0, copy, &LRargs);
  pthread_create(&RLthread, 0, copy, &RLargs);
  pthread_join(LRthread, 0);
  pthread_join(RLthread, 0);
}
```

Quiz 1

Does this work?

a) yes

b) no

Multiple Arguments

```
struct 2args {
  int src;
  int dest;
} args;
```

Quiz 2

Does this work?

a) yes

b) no

```
void relay(int left, int right) {
  pthread_t LRthread, RLthread;
  args.src = left; args.dest = right;
  pthread_create(&LRthread, 0, copy, &args);
  args.src = right; args.dest = left;
  pthread_create(&RLthread, 0, copy, &args);
}
```