```
pthread_mutex_t   m = PTHREAD_MUTEX_INITIALIZER; // declare/init a lock
pthread_cond_t    c =  PTHREAD_COND_INITIALIZER; // declare/init a CV

a condition variable (CV) is:
- queue of waiting threads
a single lock is associated with a CV (sometimes N CVs per lock)

wait (cond_t *cv, mutex_t *lock)
- assumes the lock is held when wait() is called)
- puts caller to sleep + releases the lock (atomically)
- when awoken, reacquires lock before returning
signal (cond_t *cv)
- wake a single waiting thread (if >= 1 thread is waiting)
- if there is no waiting thread, just return w/o doing anything
A CV is usually PAIRED with some kind state variable
- e.g., integer (which indicates the state of the program)
int       done = 0; // example of related "state" variable (could be an int)
```

SOLUTION 1: Spin

```
void *child(void *arg) {
    printf("child\n");
    done = 1;
    return NULL;
}

int main(int argc, char *argv[]) {
    pthread_t p;
    printf("parent: begin\n");
    Pthread_create(&p, 0, child, 0);
    while (done == 0)
        ; // spin (inefficient)
    printf("parent: end\n");
    return 0;
}
```

SOLUTION 2: No Lock

```
void *child(void *arg) {
    printf("child\n");
    done = 1;
    Pthread_cond_signal(&c);
    return NULL;
}

int main(int argc, char *argv[]) {
    pthread_t p;
    printf("parent: begin\n");
    Pthread_create(&p, 0, child, 0);
    while (done == 0) {
        Pthread_cond_wait(&c, &m);
    }
    printf("parent: end\n");
    return 0;
```

SOLUTION 3: No State Variable

```
void *child(void *arg) {
    printf("child\n");
    Pthread_mutex_lock(&m);
    Pthread_cond_signal(&c);
    Pthread_mutex_unlock(&m);
    return NULL;
}

int main(int argc, char *argv[]) {
    pthread_t p;
    printf("parent: begin\n");
    Pthread_create(&p, 0, child, 0);
    Pthread_mutex_lock(&m);
    Pthread_cond_wait(&c, &m);
    Pthread_mutex_unlock(&m);
    printf("parent: end\n");
    return 0;
}
```

SOLUTION 4: Actually Works

```
void *child(void *arg) {
    printf("child\n");
    Pthread_mutex_lock(&m);
    done = 1;
    Pthread_cond_signal(&c);
    Pthread_mutex_unlock(&m);
    return NULL;
}

int main(int argc, char *argv[]) {
    pthread_t p;
    printf("parent: begin\n");
    Pthread_create(&p, 0, child, 0);
    Mutex_lock(&m);
    while (done == 0)
        Cond_wait(&c, &m);
    Mutex_unlock(&m);
    printf("parent: end\n");
    return 0;
}
```