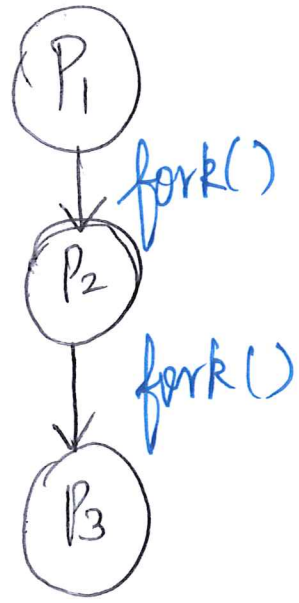
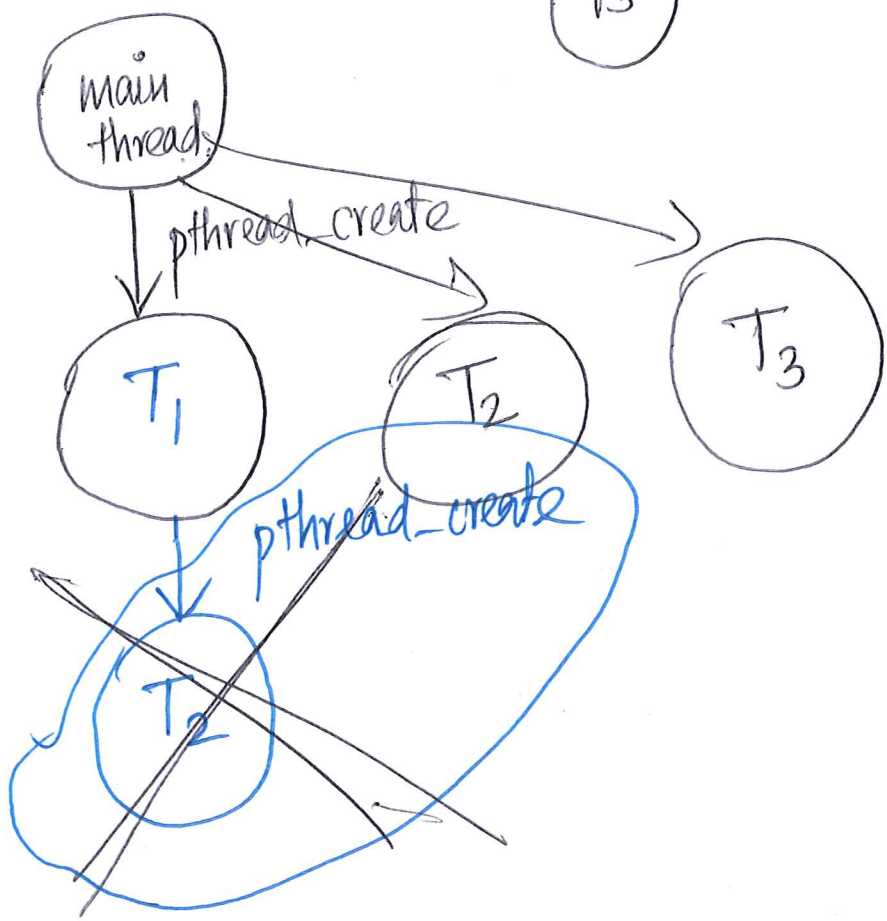
CVs

1. Threads
2. Locks
   - HW Locks
   - Adv. Locks
3. CVs

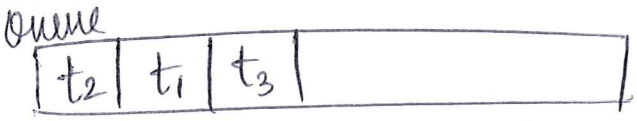




Condition Variables

```
mutex_t    lock;   // declare a lock
cond_t     cv;     // declare a condition variable
```

a **condition variable** (CV) is:
- queue of waiting threads



a single **lock** is associated with each CV
- see below for usage

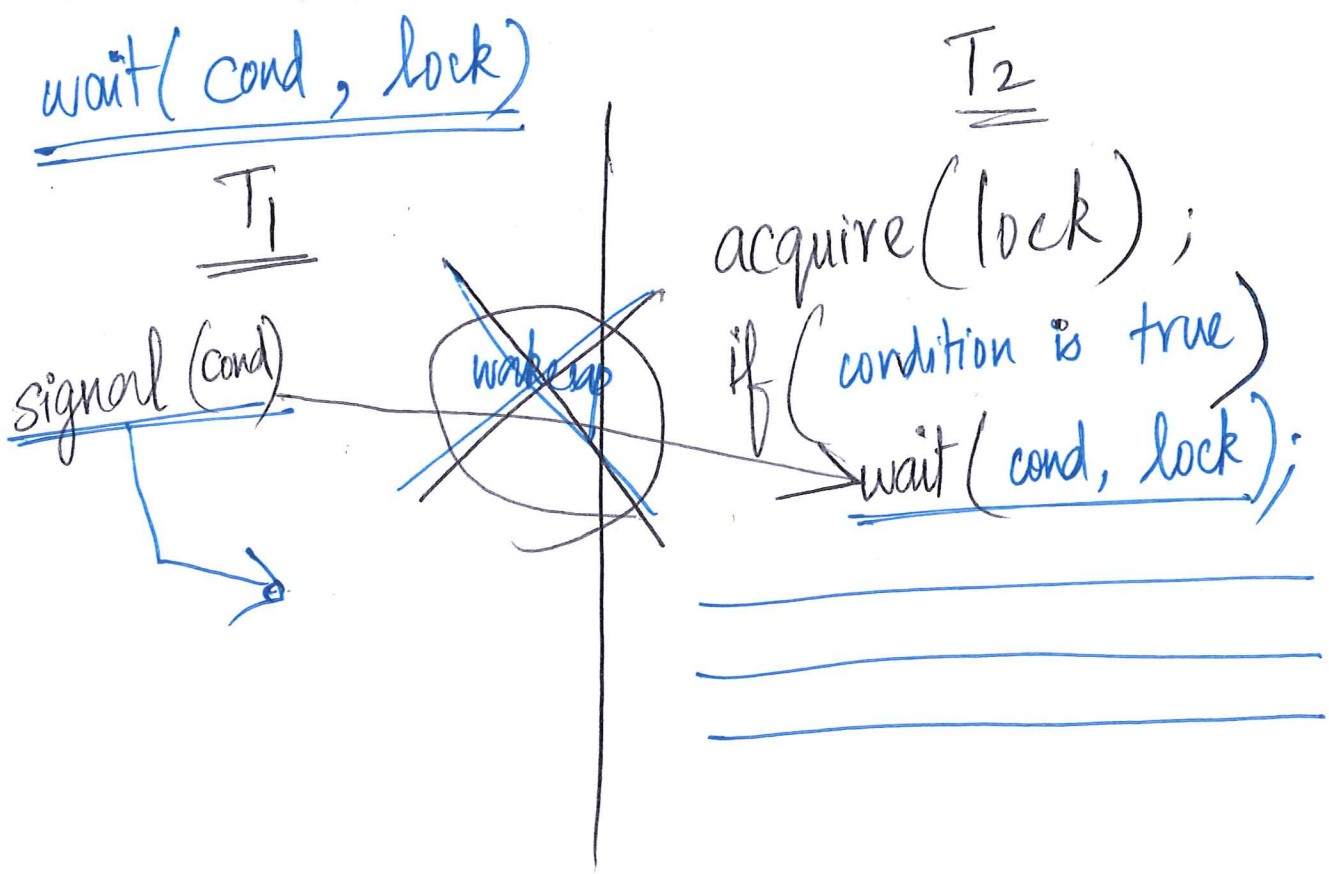
There are two main operations that are important for CVs:

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

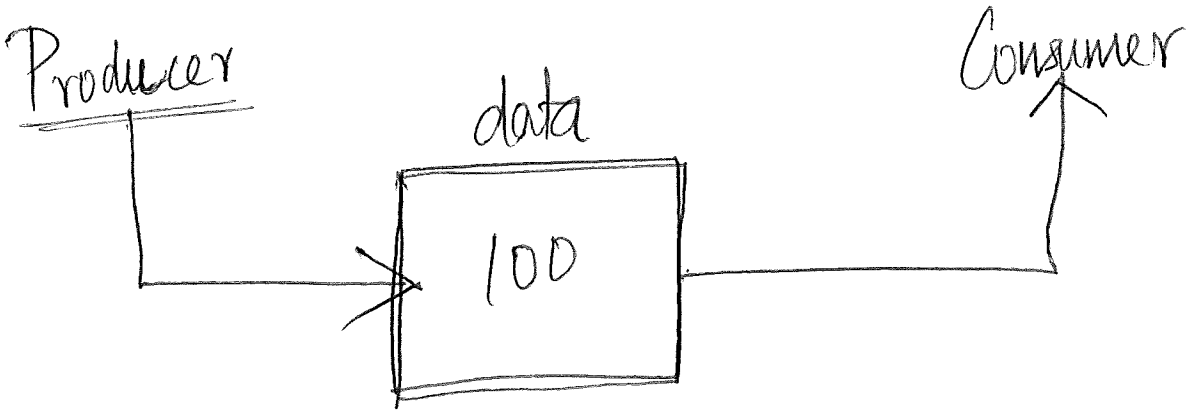
signal (cond_t *cv)
- wake a single waiting thread (if >= 1 thread is waiting)
- if there is no waiting thread, just return w/o doing anything

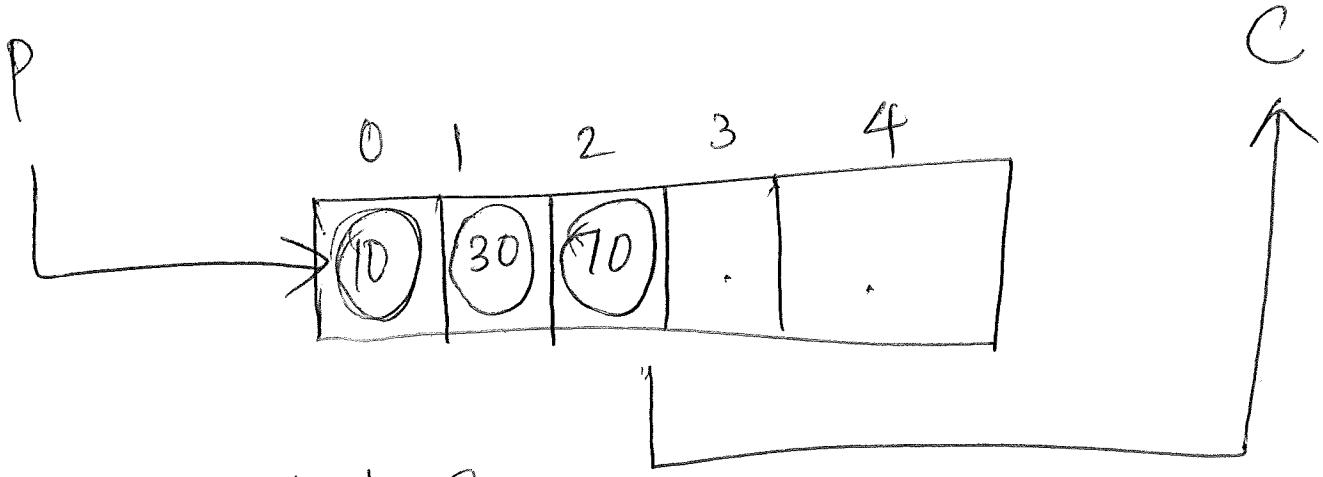
A CV is usually **PAIRED** with some kind **state variable**
- e.g., integer (which indicates the state of the system that we're interested in)

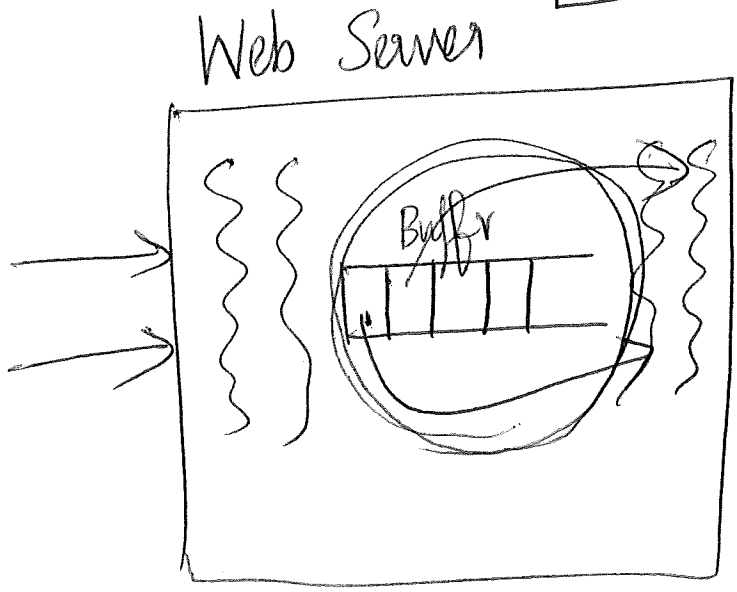
```
int        state; // related "state" variable (could be an int)
```



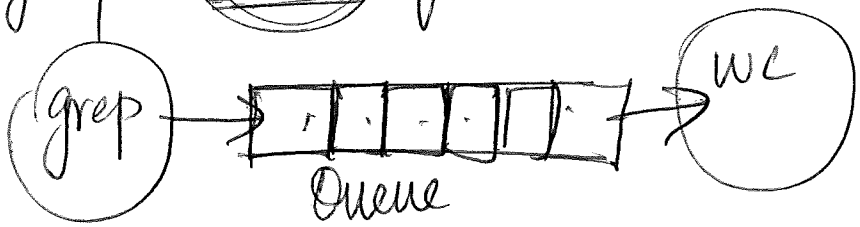
Producer - Consumer (Bounded Buffer)

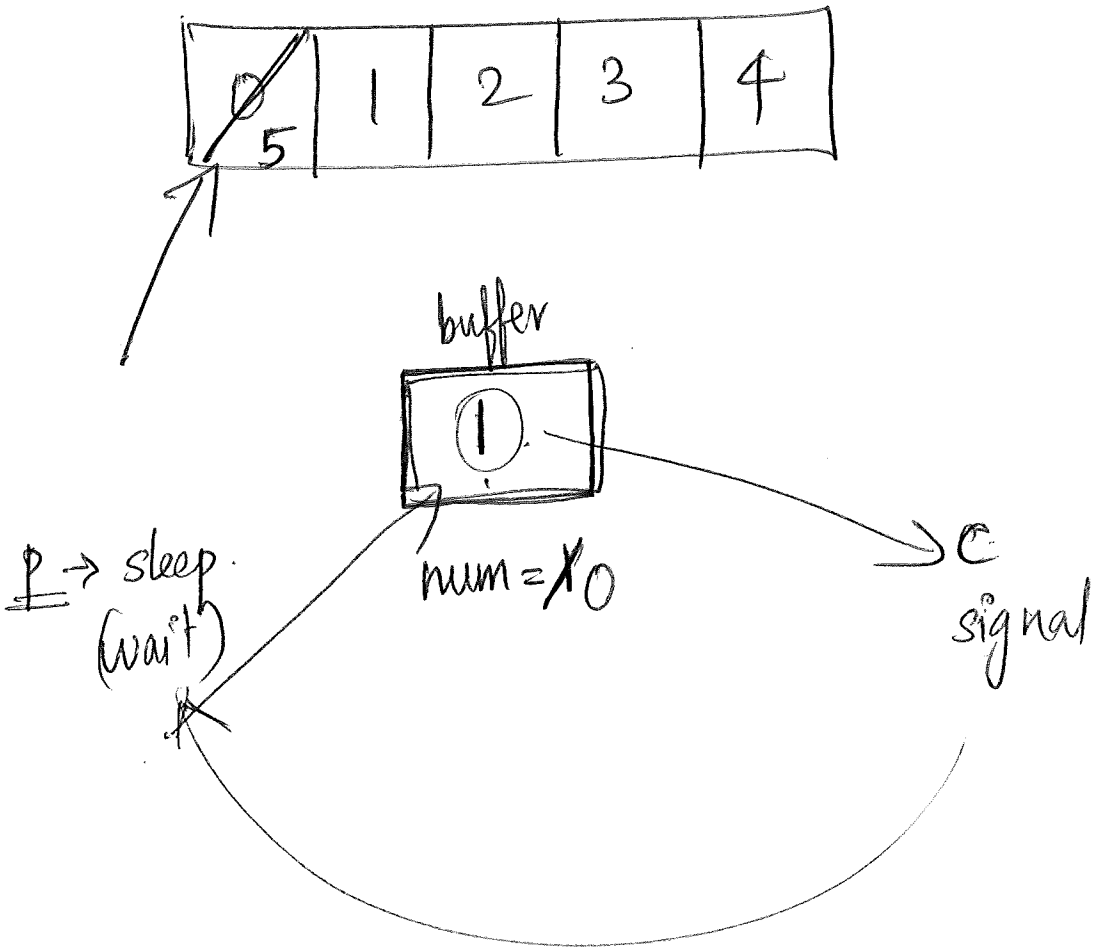






$ grep CS537 file.txt | wc -l




0
5
1
2
3
4
buffer
1
num = ~~1~~ 0
P → sleep
(wait)
C
signal

MAIN PROGRAM

```
int main(int argc, char *argv[]) {
    max   = atoi(argv[1]);
    loops = atoi(argv[2]);
    consumers = atoi(argv[3]);
    buffer = (int *) Malloc(max * sizeof(int));
    pthread_t pid, cid[CMAX];
    Pthread_create(&pid, NULL, producer, NULL);
    for (int i = 0; i < consumers; i++)
        Pthread_create(&cid[i], NULL, consumer, NULL);
    Pthread_join(pid, NULL);
    for (i = 0; i < consumers; i++)
        Pthread_join(cid[i], NULL);
}
```

QUEUE GET/PUT

```
void do_fill(int value) {
    buffer[fillptr] = value;
    fillptr = (fillptr + 1) % max;
    numfull++;
}

int do_get() {
    int tmp = buffer[useptr];
    useptr = (useptr + 1) % max;
    numfull--;
    return tmp;
}
```

Solution v1 (Single CV)

```
void *producer(void *arg) {
  for (int i = 0; i < loops; i++) {
    Mutex_lock(&m);              // p1
    while (numfull == max)       // p2
      Cond_wait(&cond, &m);      // p3
    do_fill(i);                  // p4
    Cond_signal(&cond);          // p5
    Mutex_unlock(&m);            // p6
  }
}
void *consumer(void *arg) {
  while (1) {
    Mutex_lock(&m);              // c1
    while (numfull == 0)         // c2
      Cond_wait(&cond, &m);      // c3
    int tmp = do_get();          // c4
    Cond_signal(&cond);          // c5
    Mutex_unlock(&m);            // c6
    printf("%d\n", tmp);
  }
}
```

Solution v2 (2 CVs, "if")

```
void *producer(void *arg) {
  for (int i = 0; i < loops; i++) {
    Mutex_lock(&m);              // p1
    if (numfull == max)          // p2
      Cond_wait(&empty, &m);     // p3
    do_fill(i);                  // p4
    Cond_signal(&fill);          // p5
    Mutex_unlock(&m);            // p6
  }
}
void *consumer(void *arg) {
  while (1) {
    Mutex_lock(&m);              // c1
    if (numfull == 0)            // c2
      Cond_wait(&fill, &m);      // c3
    int tmp = do_get();          // c4
    Cond_signal(&empty);         // c5
    Mutex_unlock(&m);            // c6
    printf("%d\n", tmp);
  }
}
```

Solution v3 (2 CVs, "while")

```
void *producer(void *arg) {
  for (int i = 0; i < loops; i++) {
    Mutex_lock(&m);              // p1
    while (numfull == max)       // p2
      Cond_wait(&empty, &m);     // p3
    do_fill(i);                  // p4
    Cond_signal(&fill);          // p5
    Mutex_unlock(&m);            // p6
  }
}
void *consumer(void *arg) {
  while (1) {
    Mutex_lock(&m);              // c1
    while (numfull == 0)         // c2
      Cond_wait(&fill, &m);      // c3
    int tmp = do_get();          // c4
    Cond_signal(&empty);         // c5
    Mutex_unlock(&m);            // c6
    printf("%d\n", tmp);
  }
}
```

Solution v4 (2 CVs, "while", unlock)

```
void *producer(void *arg) {
  for (int i = 0; i < loops; i++) {
    Mutex_lock(&m);              // p1
    while (numfull == max)       // p2
      Cond_wait(&empty, &m);     // p3
    Mutex_unlock(&m);            // p3a
    do_fill(i);                  // p4
    Mutex_lock(&m);              // p4a
    Cond_signal(&fill);          // p5
    Mutex_unlock(&m);            // p6
  }
}
void *consumer(void *arg) {
  while (1) {
    Mutex_lock(&m);              // c1
    while (numfull == 0)         // c2
      Cond_wait(&fill, &m);      // c3
    Mutex_unlock(&m);            // c3a
    int tmp = do_get();          // c4
    Mutex_lock(&m);              // c4a
    Cond_signal(&empty);         // c5
    Mutex_unlock(&m);            // c6
  }
}
```

Producer-Consumer (Bounded Buffer) Problem.

S1: Single CV

1 producer, 1 consumer (max = 1)

P: P_1 P_2 P_4 P_5 P_6 P_1 P_2 P_3 (wait)

(now awake)

C: C_1 C_2 C_4 C_5 C_6 C_1 ... (etc)

(empty)

S1: 1 producer, 2 consumers (max = 1) [assume consumers run first].

(empty) (signal) whom?

Ca: C_1 C_2 C_3 (wait)

(now awake)

Cb: C_1 C_2 C_3 (wait)

C_2 C_4 C_5

P: P_1 P_2 P_4 (fill) P_5 (signal) P_6 P_1 P_2 P_3 (wait)

S2: 2 CVs, "if"

1 producer, 2 consumers (max = 1)

(get on empty buffer) C_4

Ca: C_1 C_2 C_3 (wait)

(awake)

(full) (signal) (wait)

P: P_1 P_2 P_4 P_5 P_6 P_1 P_2 P_3

Cb: C_1 C_2 C_4 C_5 C_6 C_1 C_2 C_3

(empty) (signal) (wait)

S3: 2 CVs, "while"

Working Solution

S4: 2 CVs, "while", unlock

does NOT protect critical section w/ a queue.