Concurrency #4:

Producer / Consumer

=) new! solution 4:
add different lock
to put/get
why doesn't it work?

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
  }
}
```

(1) recall:
def of
CV -> wait
-> signal

(2) P/C problem (buffer has max entries)

(3) S1: e.g. (together)
-> do single P, C first
-> then P, 2 Cs
2 C's wait
1 P fills, signals, waits
C1 consumes, signals (oops, signaled other consumer!)

(4) S2: 2 CVs, if
C1 runs, waits
P runs, signals, waits (C1 runnable)
C2 races in, consumes
C1 runs, oh oh!

(5) S3: working solution

(6) S4: no mutex on get/put

S1: 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.)

(full) (empty)

S7: 1 producer, 2 consumers (max=2) [assume consumers run first]

C_1: C_1 C_2 C_3 (wait) (now awake)

C_2: C_1 C_2 C_3 (wait)

(empty) C_2 C_4 C_5 (signal ⇒ whom?)

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

(full)

S2: 1 producer, 2 consumers (max=2)

C_1: C_1 C_2 C_3 (wait) (awake)

P: P_1 P_2 P_4 P_5 P_6 P_1 P_2 P_3 (full) (sig) (wait) (awake)

C_2: C_1 C_2 C_4 C_5 C_6 C_1 C_2 C_3 (empty)

(get on empty buffer)

C_4

S3: working solution

S4: does not protect critical section w/in queue