pthread_mutex_lock();

if (mutex == 1)

mutex--;

else

block the thread;

pthread_mutex_trylock();

0 →

EBUSY (16)

EINVAL (22)

pthread_mutex_unlock();

if (mutex == 0)

release a blocked thread;

else

mutex++;

CPU

the producer-Consumer problem

Semaphore e=n, f=0; ← # of buffers in pool

binary_semaphore b; ← initialized to 1

item type in_item, out_item,

int in=0, out=0; ← position of buffer ptr for p & c

producer:

while(1) {

produce an in_item;

p(e);

b p(b);

buffer(in) = in_item;

in = (in+1)%n;

bV(b);

V(f);

}

Consumer:

while(1) {

p(f);

b p(b);

out_item = buffer[out];

```
    out = (out+1)%n;
    bV(b);
    V(e);
}
```

readers-writers problem

Solution favors reader:

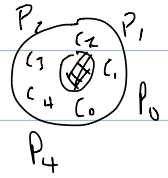
```
binary_semaphore mutex;
binary_semaphore rw_mutex;
int reader_count;
reader:
    while(1){
      bp(mutex);
    reader_count++;
    if(reader_count==1)
      bp(rw_mutex);
    bV(mutex);
    ......
    /*reading occurs here*/
    ------
    bP(mutex);
    reader_count--;
    if (reader_count==0)
      bV(rw_mutex);
    bV(mutex);
```

writer:

```
while(1){
    bP(rw_mutex);
    ....
    /* writing occurs here */
    ....
    bV(rw_mutex);
}
```

K1 → W1 → R2 → K3

Dining philosopher problem




```
semaphore chopsticks[5] = {1,1,1,1,1};
while(1){
    ....
    Feeling hungry;
    p(chopstick[i]);
    p(chopstick[(i+1)%5]);
    ....
    /* eating here */
    ....
    V(chopstick[i]);
    V(chopstick[(i+1)%5]);
```

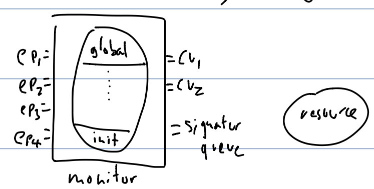
......

(* thinking here */)

......

}

monitor (ADT) don't use classes in C++ (NO SYNCHRO)



bounded_buffer problem:

```
monitor Bounded_Buffer{
  char buffer[n];
  int nextin=0, nextout=0, full_cnt=0;
   condition notempty, notfull;
   deposit(char c){
   ......
  }
    remove(char c){
     ......
   }
}
```