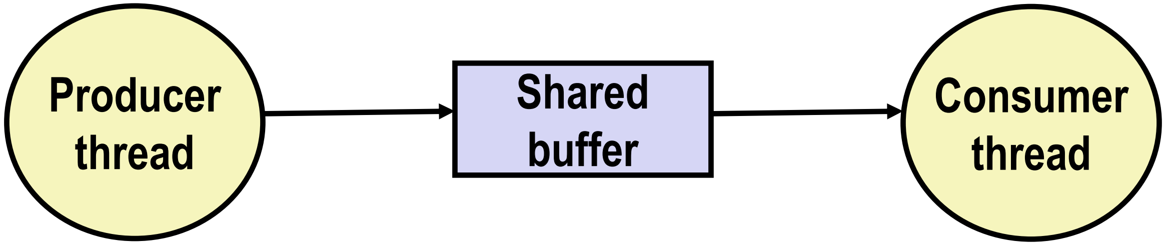
Concurrent Programming:
Synchronizing threads:
3. *The Producer-Consumer Problem*

Using Semaphores to Coordinate Access to Shared Resources

- **Basic idea: Thread uses a semaphore operation to notify another thread that some condition has become true**
  - Use counting semaphores to keep track of resource state and to notify other threads
  - Use mutex to protect access to resource

- **Two classic examples:**
  - The Producer-Consumer Problem (this lecture)
  - The Readers-Writers Problem (next lecture)

Producer-Consumer Problem



- **Common synchronization pattern:**
  - Producer waits for empty ***slot***, inserts item in buffer, and notifies consumer
  - Consumer waits for ***item***, removes it from buffer, and notifies producer
  - May have more than one producer, more than one consumer

- **Examples**
  - Multimedia processing:
    - Producer creates MPEG video frames, consumer renders them
  - Event-driven graphical user interfaces
    - Producer detects I/O events (mouse, keyboard), puts them in buffer
    - Consumer retrieves events from buffer and paints the display

Producer-Consumer on an *n*-element Buffer

- **Requires a mutex and two counting semaphores:**
  - `mutex`: enforces mutually exclusive access to the the buffer
  - `slots`: counts the available slots in the buffer
  - `items`: counts the available items in the buffer

- **Implemented using a shared buffer package called `sbuf`**

- **We will use the package in our prethreaded concurrent server:**
  - Main thread puts new connfd's in buffer, and back to `accept`
  - Pool of threads ready to serve new connfd's by reading buffer

sbuf Package - Declarations

```
#include "csapp.h"

typedef struct {
    int *buf;          /* Buffer array */
    int n;             /* Maximum number of slots */
    int front;         /* buf[(front+1)%n] is first item */
    int rear;          /* buf[rear%n] is last item */
    sem_t mutex;       /* Protects accesses to buf */
    sem_t slots;       /* Counts available slots */
    sem_t items;       /* Counts available items */
} sbuf_t;

void sbuf_init(sbuf_t *sp, int n);
void sbuf_deinit(sbuf_t *sp);
void sbuf_insert(sbuf_t *sp, int item);
int sbuf_remove(sbuf_t *sp);
```

sbuf.h

sbuf Package - Implementation

Initializing and deinitializing a shared buffer:

```
/* Create an empty, bounded, shared FIFO buffer with n slots */
void sbuf_init(sbuf_t *sp, int n)
{
    sp->buf = Calloc(n, sizeof(int));
    sp->n = n;                       /* Buffer holds max of n items */
    sp->front = sp->rear = 0;        /* Empty buffer iff front == rear */
    Sem_init(&sp->mutex, 0, 1);      /* Binary semaphore for locking */
    Sem_init(&sp->slots, 0, n);      /* Initially, buf has n empty slots */
    Sem_init(&sp->items, 0, 0);      /* Initially, buf has 0 items */
}

/* Clean up buffer sp */
void sbuf_deinit(sbuf_t *sp)
{
    Free(sp->buf);
}
```

sbuf.c

sbuf Package - Implementation

Inserting an item into a shared buffer:

```
/* Insert item onto the rear of shared buffer sp */
void sbuf_insert(sbuf_t *sp, int item)
{
    P(&sp->slots);                          /* Wait for available slot */
    P(&sp->mutex);                          /* Lock the buffer */
    ++sp->rear;                             /* Move rear */
    sp->buf[sp->rear % sp->n] = item;       /* Insert the item */
    V(&sp->mutex);                          /* Unlock the buffer */
    V(&sp->items);                          /* Announce available item */
}
```

sbuf.c

sbuf Package - Implementation

Removing an item from a shared buffer:

```
/* Remove and return the first item from buffer sp */
int sbuf_remove(sbuf_t *sp)
{
    int item;
    P(&sp->items);                          /* Wait for available item */
    P(&sp->mutex);                          /* Lock the buffer */
    ++sp->front;                            /*  Move front */
    item = sp->buf[sp->front % sp->n];      /* Remove the item */
    V(&sp->mutex);                          /* Unlock the buffer */
    V(&sp->slots);                          /* Announce available slot */
    return item;
}
```

sbuf.c