Chapter 30: Condition Variables

Overview

- **Main Idea**: Condition variables (CVs) enable threads to wait for specific conditions while releasing locks, avoiding busy-waiting.
- **Key Terms**: `wait()`, `signal()`, `broadcast()`, predicate, Mesa vs. Hoare semantics.

---

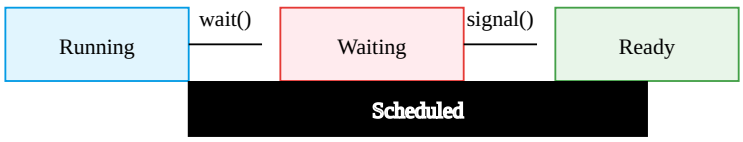
1. Condition Variable Basics

Core Operations

```
pthread_cond_t c = PTHREAD_COND_INITIALIZER;
pthread_cond_wait(pthread_cond_t *c, pthread_mutex_t *m);
pthread_cond_signal(pthread_cond_t *c);
pthread_cond_broadcast(pthread_cond_t *c);
```

Usage Pattern

```
pthread_mutex_lock(&m);
while (condition == false) {  // Always use while!
    pthread_cond_wait(&c, &m);
}
// Critical section work
pthread_mutex_unlock(&m);
```



Lock State During wait():

1. Releases lock when sleeping
2. Re-acquires lock before returning

---

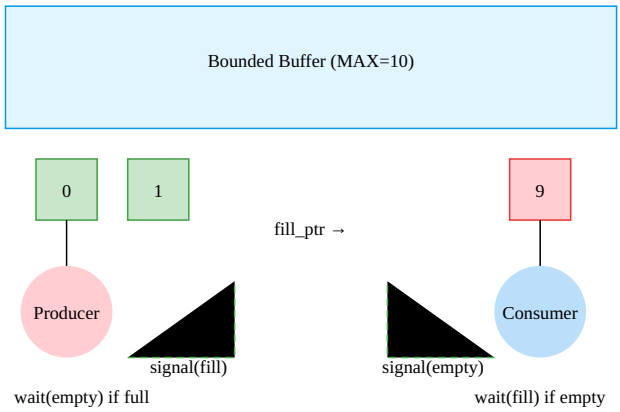
2. Producer-Consumer Problem

Bounded Buffer Implementation

```
int buffer[MAX];
int fill = 0, use = 0, count = 0;

void put(int value) {
    buffer[fill] = value;
    fill = (fill + 1) % MAX;
    count++;
}

int get() {
    int tmp = buffer[use];
    use = (use + 1) % MAX;
    count--;
    return tmp;
}
```

Synchronization with CVs

```
pthread_cond_t empty, fill;
pthread_mutex_t m;

void *producer(void *arg) {
    pthread_mutex_lock(&m);
    while (count == MAX) {
        pthread_cond_wait(&empty, &m);
    }
    put(value);
    pthread_cond_signal(&fill);
    pthread_mutex_unlock(&m);
}

void *consumer(void *arg) {
    pthread_mutex_lock(&m);
    while (count == 0) {
        pthread_cond_wait(&fill, &m);
    }
    int tmp = get();
    pthread_cond_signal(&empty);
    pthread_mutex_unlock(&m);
    return tmp;
}
```



---

3. CV Semantics

Mesa vs. Hoare Semantics

Aspect	Mesa (Pthreads)	Hoare
Wakeup	Signal moves waiter to ready queue	Signal immediately switches
Guarantee	Condition may not hold when woken	Condition holds when running
Usage	Requires `while` loop	Can use `if`

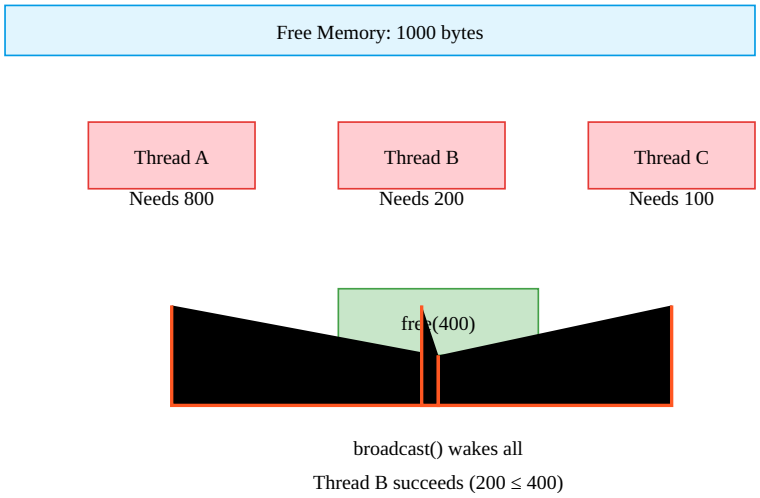
---

4. Covering Conditions

Example: Memory Allocation

```
// Allocate memory if enough available
void *allocate(int size) {
    pthread_mutex_lock(&m);
    while (bytes_left < size) {
        pthread_cond_wait(&c, &m);
    }
    void *ptr = malloc(size);
```

```
    bytes_left -= size;
    pthread_mutex_unlock(&m);
    return ptr;
}

// Free memory and signal all waiters
void free(void *ptr, int size) {
    pthread_mutex_lock(&m);
    bytes_left += size;
    pthread_cond_broadcast(&c); // Wake all waiters
    pthread_mutex_unlock(&m);
}
```



---

5. Common Pitfalls

1. **Forgetting to Use While**:

   ```
   if (count == 0) {  // BUG!
       pthread_cond_wait(&fill, &m);
   }
   ```

2. **Unlocked Signal/Broadcast**:

   ```
   pthread_cond_signal(&c);  // May miss wakeup
   ```

---

6. Summary Table

Concept	Key Insight
Waiting	Always wait in a loop checking the predicate
Signaling	Hold lock when signaling (except optimizations)
Broadcast	Use when multiple waiters may proceed
Performance	CVs avoid busy-waiting and context switch costs

Homework Insights

1. **Thread Ordering**: Use `-P` flag in `x86.py` to test wakeup ordering.
2. **Lost Wakeups**: Simulate missed signals by commenting out `signal()` calls.
3. **Covering Conditions**: Implement multi-size memory allocator with `broadcast()`.