Operating Systems – COC 3071L

SE 5th A – Fall 2025

Example 2: Producer-Consumer Problem

Scenario: A factory assembly line

- Producers make items and place them on a conveyor belt (buffer)
- Consumers take items from the belt
- The belt has limited space (say, 10 items max)

Questions:

- "What happens if producers are faster than consumers?"
- "What happens if the belt is empty and a consumer tries to take an item?"
- "What synchronization do we need?"

Key insight: We need to track TWO things:

1. How many empty slots are available
2. How many filled slots are available

Solution approach:

- `empty` semaphore: initialized to buffer size (e.g., 10)
- `full` semaphore: initialized to 0
- `mutex` : to protect the buffer itself

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

#define BUFFER SIZE 5

int buffer[BUFFER_SIZE];
int in = 0;  // Producer index
int out = 0; // Consumer index

sem t empty;  // Counts empty slots
sem t full;   // Counts full slots
pthread mutex t mutex;

void* producer(void* arg) {
```

```
    int id = *(int*)arg;

    for(int i = 0; i < 3; i++) { // Each producer makes 3 items int item = id * 100 + i;

        // TODO: Wait for empty slot sem_wait(&empty);

        // TODO: Lock the buffer pthread_mutex_lock(&mutex);

        // Add item to buffer buffer[in] = item;
        printf("Producer %d produced item %d at position %d\n", id, item, in);
        in = (in + 1) % BUFFER_SIZE;

        // TODO: Unlock the buffer pthread_mutex_unlock(&mutex);

        // TODO: Signal that buffer has a full slot sem_post(&full);

        sleep(1);
    } return NULL;
}

void* consumer(void* arg) { int id =
    *(int*)arg;

    for(int i = 0; i < 3; i++) {
        // TODO: Students complete this similar to producer sem_wait(&full);
        pthread_mutex_lock(&mutex);

        int item = buffer[out];
        printf("Consumer %d consumed item %d from position %d\n", id, item, out);
        out = (out + 1) % BUFFER_SIZE;

        pthread_mutex_unlock(&mutex); sem_post(&empty);

        sleep(2); // Consumers are slower
    } return NULL;
}

int main() {
```

```
    pthread_t prod[2], cons[2];
    int ids[2] = {1, 2};

    // Initialize semaphores
    sem_init(&empty, 0, BUFFER_SIZE);  // All slots empty initially
    sem_init(&full, 0, 0);              // No slots full initially
    pthread_mutex_init(&mutex, NULL);

    // Create producers and consumers
    for(int i = 0; i < 2; i++) {
        pthread_create(&prod[i], NULL, producer, &ids[i]);
        pthread_create(&cons[i], NULL, consumer, &ids[i]);
    }

    // Wait for completion
    for(int i = 0; i < 2; i++) {
        pthread_join(prod[i], NULL);
        pthread_join(cons[i], NULL);
    }

    // Cleanup
    sem_destroy(&empty);
    sem_destroy(&full);
    pthread_mutex_destroy(&mutex);

    return 0;
}
```

Demonstration:

1. It is the code in which there are 2 semaphore and 1 mutex
2. One semaphore is for producer and 1 for consumer and mutex is used to lock and unlock the space
3. Semaphore name are full and empty ,empty count empty space ,and full count filled space
4. Sem_wait() wait for the empty space
5. Sem_post() for mean space is taken
6. Wait for the completing all thread and and destroy at the end
7. Possible value will be
    - Id*100+[i],i=0,1,2 id=1,2
    - 101
    - 102
    - 103
    - 201
    - 202
    - 201