CSCE 311 - Operating Systems

Synchronization: Producer-Consumer Problem

Qiang Zeng, Ph.D.
Fall 2018
Previous Class...

- Restroom problem
- Bar problem
- Enforcing execution order
Binary Semaphore for the restroom problem: mutual exclusion

• Given a single-spot restroom, write the function for using the restroom
• Hint: Binary Semaphore, a special semaphore, whose counter value can only be 0 or 1
• Here, Binary Semaphore is used as Mutex, a blocking lock for enforcing MUTual EXclusion

S = 1; // shared among customers (processes)

use_restroom() {
    down(S); // try to enter the restroom; = lock()
    Use the restroom //critical section
    up(S); // leave the restroom; = unlock()
}

S = 1; // shared among customers (processes)

use_restroom() {
    down(S); // try to enter the restroom; = lock()
    Use the restroom //critical section
    up(S); // leave the restroom; = unlock()
}
Semaphore for the bar problem

• Capacity = 100
• Many customers try to enter the bar concurrently
• Please write code to make sure customers <= 100
• Caution: a Mutex will not work well; why?

```c
S = 100; // shared among processes

// each process does the following
down(S); // try to enter the bar
Have fun;
up(S); // leave the bar
```
Semaphore for enforcing order

• Two processes: p0 and p1
• Hint: use two semaphores

```c
S1 = 0; // shared
S2 = 0; // shared

// Process 0
A1;
up(S1);
down(S2);
A2;

// Process 1
down(S1);
B1;
B2;
up(S2);
```
Outline

• Single-slot producer-consumer problem
• Multi-slot producer-consumer problem
Using Semaphores:
Single-slot Producer-Consumer problem
Single-slot Producer-Consumer problem

- One slot; multi Producers keep producing items, while multi Consumers keep consumes items
- Hint:
  - `removeItem()` cannot proceed if there is no item in the slot
  - `fillSlot()` cannot proceed if the slot still has an item

```c
S_slot = 1; // shared
S_item = 0; // shared

// Producer
while(true) {
    down(S_slot);
    fillSlot();
    up(S_item);
}

// Consumer
while(true) {
    down(S_item);
    removeItem();
    up(S_slot);
}
```
Using Semaphores: The Producer-Consumer Problem

Acknowledgement: some slides courtesy of Dr. Brighten Godfrey
Producer-consumer problem

- Chefs cook items and put them on a conveyer belt

Waiters pick items off the belt
Producer-consumer problem

- Now imagine many chefs!

...and many waiters!
Producer-consumer problem

• A potential mess!
Producer-consumer problem

Chef (Producer) inserts items

Waiter (Consumer) removes items

Shared resource: bounded buffer

Efficient implementation: circular fixed-size buffer
Shared buffer

Chef (Producer)

Waiter (Consumer)
Shared buffer

Chef (Producer)

Waiter (Consumer)

What does the chef do with a new pizza?

Where does the waiter take a pizza from?
Shared buffer

Chef (Producer)

Waiter (Consumer)

insertPtr

removePtr

Insert pizza
Shared buffer

Chef (Producer)

Waiter (Consumer)

Insert pizza

insertPtr

removePtr
Shared buffer

Chef (Producer)

Waiter (Consumer)

Insert pizza

insertPtr

removePtr
Shared buffer

Chef (Producer)

Waiter (Consumer)

Remove pizza

insertPtr

removePtr

removePtr
Shared buffer

Chef (Producer)

Waiter (Consumer)

Insert pizza

insertPtr

removePtr
Shared buffer

Chef (Producer)

Waiter (Consumer)

Insert pizza

insertPtr

removePtr
Shared buffer

Chef (Producer)

BUFFER FULL:
Producer must wait!

Insert pizza

Waiter (Consumer)

insertPtr
removePtr
Shared buffer

Chef (Producer)

Waiter (Consumer)

removePtr

insertPtr

Remove pizza
Shared buffer

Chef (Producer)

Waiter (Consumer)

Remove pizza

removePtr

insertPtr
Shared buffer

Chef (Producer)

Waiter (Consumer)

removePtr

insertPtr

Remove pizza
Shared buffer

Chef (Producer)

removePtr

insertPtr

Waiter (Consumer)

Remove pizza
Shared buffer

Chef (Producer)  
Waiter (Consumer)

Remove pizza
Shared buffer

Chef (Producer)

Waiter (Consumer)

Remove pizza

insertPtr

removePtr
Shared buffer

Chef (Producer)

Waiter (Consumer)

Remove pizza

insertPtr

removePtr
Shared buffer

Chef (Producer)

Buffer empty:
Consumer must be blocked!

Waiter (Consumer)

Remove pizza

insertPtr

removePtr

STOP

CSCE 311 – Operating Systems
30
Designing a solution

Chef (Producer)
- Wait for empty slot
- Insert item
- Signal item arrival

Waiter (Consumer)
- Wait for item arrival
- Remove item
- Signal empty slot available

What synchronization do we need?
Designing a solution

Chef (Producer)

Wait for empty slot
Insert item
Signal item arrival

Mutex (shared buffer)

Wait for item arrival
Remove item
Signal empty slot available

Waiter (Consumer)

What synchronization do we need?
Designing a solution

Chef (Producer)
- Wait for empty slot
- Insert item
- Signal item arrival

Semaphore (# empty slots)

Waiter (Consumer)
- Wait for item arrival
- Remove item
- Signal empty slot available

What synchronization do we need?
What synchronization do we need?

Semaphore

(# filled slots)

Wait for empty slot
Insert item
Signal item arrival

Wait for item arrival
Remove item
Signal empty slot available

Chef (Producer)

Waiter (Consumer)
buffer[pi] = data;
pi = (pi + 1) % N;

result = buffer[ci];
 ci = (ci +1) % N;
Single-slot Producer-Consumer problem

- One slot; multi Producers keep producing items, while multi Consumers keep consumes items
- Hint:
  - Consumer.removeItem() has to occur after Producer.fillSlot()
  - Once the slot is filled, Producer.fillSlot() has to occur after Consumer.removeItem()

```java
S_slot = 1; // shared
S_item = 0; // shared

// Producer
while(true) {
    down(S_slot);
    fillSlot();
    up(S_item);
}

// Consumer
while(true) {
    down(S_item);
    removeItem();
    up(S_slot);
}
```
Producer-Consumer Code

Counting semaphore – check and decrement the number of free slots

`sem_wait(&slots);`

Block if there are no free slots

`mutex_lock(&mutex);
buffer[pi] = data;
pi = (pi + 1) % N;
mutex_unlock(&mutex);`

`sem_post(&items);`

Done – increment the number of available items

Counting semaphore – check and decrement the number of available items

`sem_wait(&items);`

`mutex_lock(&mutex);
result = buffer[ci];
ci = (ci + 1) % N;
mutex_unlock(&mutex);`

`sem_post(&slots);`

Done – increment the number of free slots
Implementation based on Monitor and CV

```plaintext
monitor ProducerConsumer
c conditioned cv_full, cv_cempty;
int count;

procedure produce();
{
    while (count == N) wait(cv_full); // if buffer is full, block
    put_item(widget);               // put item in buffer
    count = count + 1;              // increment count of full slots
    if (count == 1) signal(cv_empty); // if buffer was empty, wake consumer
}

procedure consume();
{
    while (count == 0) wait(cv_empty); // if buffer is empty, block
    remove_item(widget);              // remove item from buffer
    count = count - 1;                // decrement count of full slots
    if (count == N-1) signal(cv_full); // if buffer was full, wake producer
}
```
Summary

• Single-slot producer-consumer problem
• Multi-slot producer-consumer problem
References

• Monitor type: Hoare vs Mesa (Mesa is the choice of most OSes)