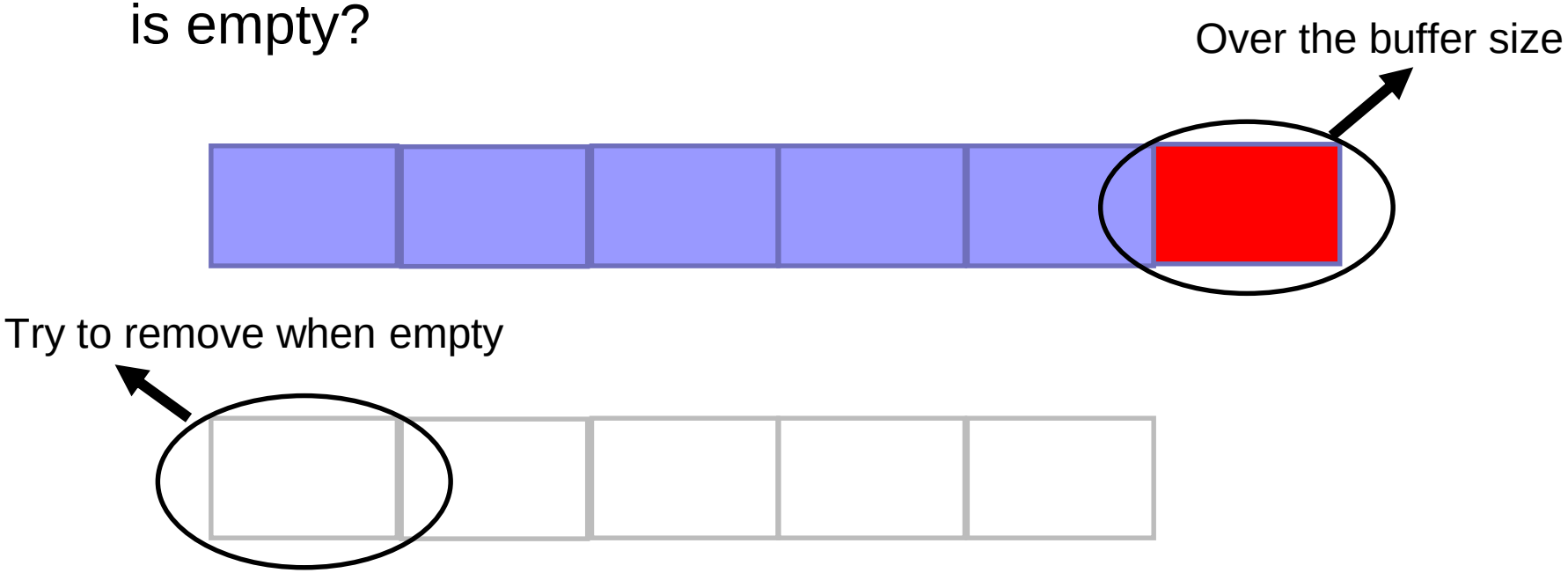
Homework Assignment #3: Solving Producer-Consumer Problem by Semaphore

Outline

- **Producer-consumer problem**
- **Semaphore**
- **Application Programming Interface**
  - Pthread API
  - semaphore
- **Homework Assignment #3**
- **Reference**

Producer-Consumer Problem (1/2)

- There are two process with a fixed-sized buffer.
- Producer generates data and consumer consumes the data.
- Problem: how to prevent producer generate data over the buffer size or consumer remove data when the buffer is empty?



Producer-Consumer Problem (2/2)

```
run consumer count=-1
run consumer count=-2
run consumer count=-3
run consumer count=-4
run consumer count=-5
run producer count=-4
run producer count=-2
run producer count=-1
run producer count=0
run producer count=1
run producer count=-3
run producer count=2
run producer count=3
run producer count=4
run producer count=5
run consumer count=4
run consumer count=3
run consumer count=2
run consumer count=1
run consumer count=0
```

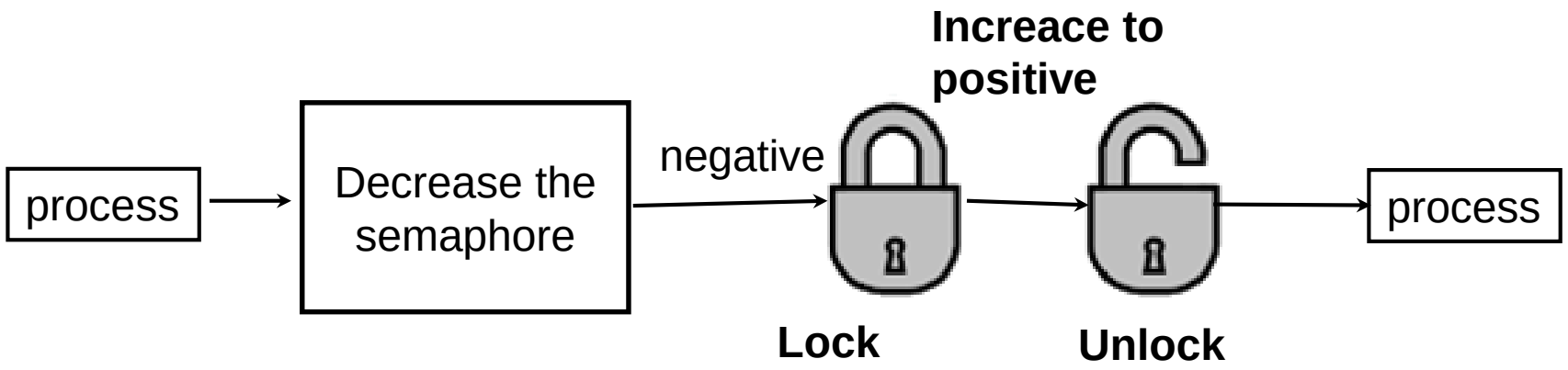
We can see the consumer remove data when the buffer is empty.

Outline

- **Producer-consumer problem**
- **Semaphore**
- **Application Programming Interface**
  - Pthread API
  - semaphore
- **Homework Assignment #3**
- **Reference**

Semaphore

- Sol.: we can use semaphore to record how many buffer is empty and how many buffer is full to prevent race condition.
- Semaphore: can be initialized as zero or a positive integer.
  - When it decreases to less than zero, it while lock the process until it become zero.



Example

A semaphore is similar to the parking guide. When a car comes in the space, it's value is decreased. Besides, it shows the available space we can park.

Outline

- **Producer-consumer problem**
- **Semaphore**
- **Application Programming Interface**
- **Homework Assignment #3**
- **Reference**

Semaphore

- There is a whole set of library calls associated with semaphore, most of whose names start with **`sem_t.`**
- To use these library calls, we must include the file **`semaphore.h`**,

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

sem_t mutex;
sem_t full;
sem_t empty;
```

POSIX Semaphore APIs

- We will use the following six functions
  - ***int sem_init()***
    - Initialize a semaphore.
  - ***int sem_wait()***
    - Decrease a sem's value.
  - ***int sem_post()***
    - Increase a sem's value.
  - ***int sem_destroy()***
    - Release the resource and destroy a semaphore.
  - ***int sem_getvalue()***
    - Get a sem's value.

POSIX Semaphore APIs

- **sem_init**
  - Initializes a semaphore.

```
#include<semaphore.h>
int sem_init(sem_t  *sem, int pshared,unsigned value);
Return -1 if unsuccessful
```

```
EX: sem_init(&sem , 1,1);
```

value:cannot be negative

pshared: A flag indicating whether or not the semaphore should be shared with forked processes.
-Pshared == 0 only threads of process creating semaphore can use semaphore.
Sem_t:the semaphore we initialize

POSIX Semaphore APIs

- **sem_wait**
  - Decrease the value.

```
int sem_wait ( sem_t *sem );
```

```
EX: sem_wait (&sem);
```

When the value becom negative it will lock the process

POSIX Semaphore APIs

- **sem_post**
  - Increase the value.

```
int sem_post( sem_t *sem );
```

```
EX: sem_post(&sem);
```

When there are prossce lock by sem_wait, it will unlock it.
Or it will increace the sem value.

POSIX Semaphore APIs

- **sem_destroy**
  - Destroys a previously declared semaphore.

```
int sem_destroy(sem_t *sem);
```

```
EX: sem_destroy (&sem );
```

Remember to destroy a semaphore when it is no longer needed.

POSIX Semaphore APIs

- **sem_getvalue**
  - Get the current value of sem and places the value in the location pointed to by val.

```
int sem_getvalue(sem_t *sem,int *val);
```

```
EX: sem_getvalue(&sem,&value);
```

```
int value;
sem_t semA;

sem_getvalue(&sem_name, &value);
printf("The value of the semaphors is %d\n", value);
```

Example

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h> //for sleep API
sem_t mutex;
unsigned int x = 0;

void* thread(void* arg)
{
    //wait
    sem_wait(&mutex);

    //critical section
    x = x + 1;

    //signal
    sem_post(&mutex);
}
```

Example (Cont.)

```
int main()
{
    sem_init(&mutex, 0, 1);
    pthread_t t1,t2;
    pthread_create(&t1,NULL,thread,NULL);
    sleep(2);
    pthread_create(&t2,NULL,thread,NULL);
    pthread_join(t1,NULL);
    pthread_join(t2,NULL);
    sem_destroy(&mutex);
    return 0;
}
```

Outline

- **Producer-consumer problem**
- **Semaphore**
- **Application Programming Interface**
- **Homework Assignment #3**
- **Reference**

Homework Assignments #3

- Use Pthreads API to create 4 threads: two are producers and two are consumers.
- Declare a buffer and associated variables to keep track of how many buffer is empty. The total buffer size is set to 5.
- 1st version: Implement the producer-consumer problem but without semaphore. Thus, race condition would be occurred.
- 2nd version: Implement the producer-consumer problem by semaphore to solve the race condition problem.

Execution Results of 1st Version Program

```
run consumer count=-1
run consumer count=-2
run consumer count=-3
run consumer count=-4
run consumer count=-5
run producer count=-4
run producer count=-2
run producer count=-1
run producer count=0
run producer count=1
run producer count=-3
run producer count=2
run producer count=3
run producer count=4
run producer count=5
run consumer count=4
run consumer count=3
run consumer count=2
run consumer count=1
run consumer count=0
```

We can see the consumer remove data when the buffer is empty.

Execution Results of 2nd Version Program

```
oslab@oslab-VirtualBox:~/osHW3$ ./hw3
run producer count=1
run producer count=2
run producer count=3
run producer count=4
run producer count=5
run consumer count=4
run consumer count=3
run consumer count=2
run consumer count=1
run consumer count=0
run producer count=1
run producer count=2
run producer count=3
run producer count=4
run producer count=5
run consumer count=4
run consumer count=3
run consumer count=2
run consumer count=1
run consumer count=0
```

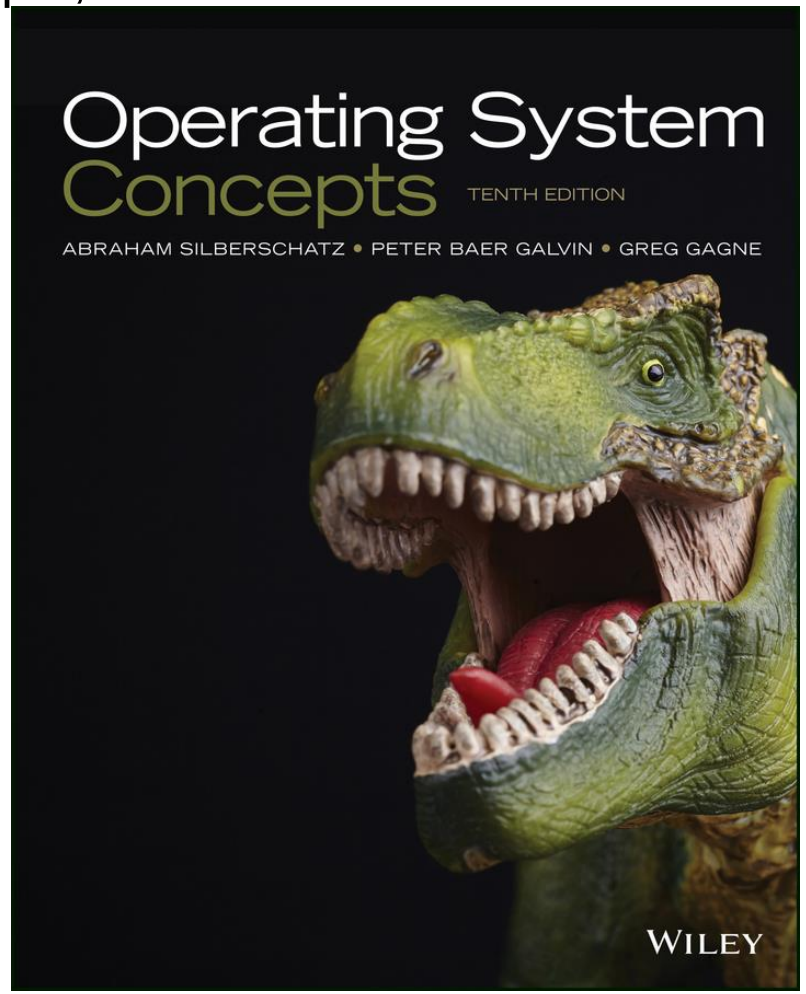
```
run producer count=1
run producer count=2
run producer count=3
run producer count=4
run producer count=5
run consumer count=4
run consumer count=3
run consumer count=2
run consumer count=1
run consumer count=0
run producer count=1
run consumer count=0
```

Outline

- **Producer-consumer problem**
- **Semaphore**
- **Application Programming Interface**
- **Homework Assignment #3**
- **Reference**

Reference

- Operating System Concepts, 10th Edition
- Oslab: Lab12.ppt



Turn in

- Deadline

  2020/12/31 PM.11:59:59
- Upload to iLearning
- File name
  - HW3_ID.zip (e.g. HW3_4106056000.zip)
    - Source code
      - .c file
    - Word

- If you don’t hand in your homework on time, your score will be deducted 10 points every day.

TA

- Name：Bo-Hua, Xu
- Email：g109056017@mail.nchu.edu.tw
  - Title format : OS HW#3 - [your name]
- Lab: OSNET(1001)