# Recitation 9

Semaphores

# Semaphore Usage

- Counting Semaphore:
  - Integer values range over an unrestricted domain
- Binary Semaphore:
  - Integer value ranges between 0 and 1
  - Same as a mutex lock

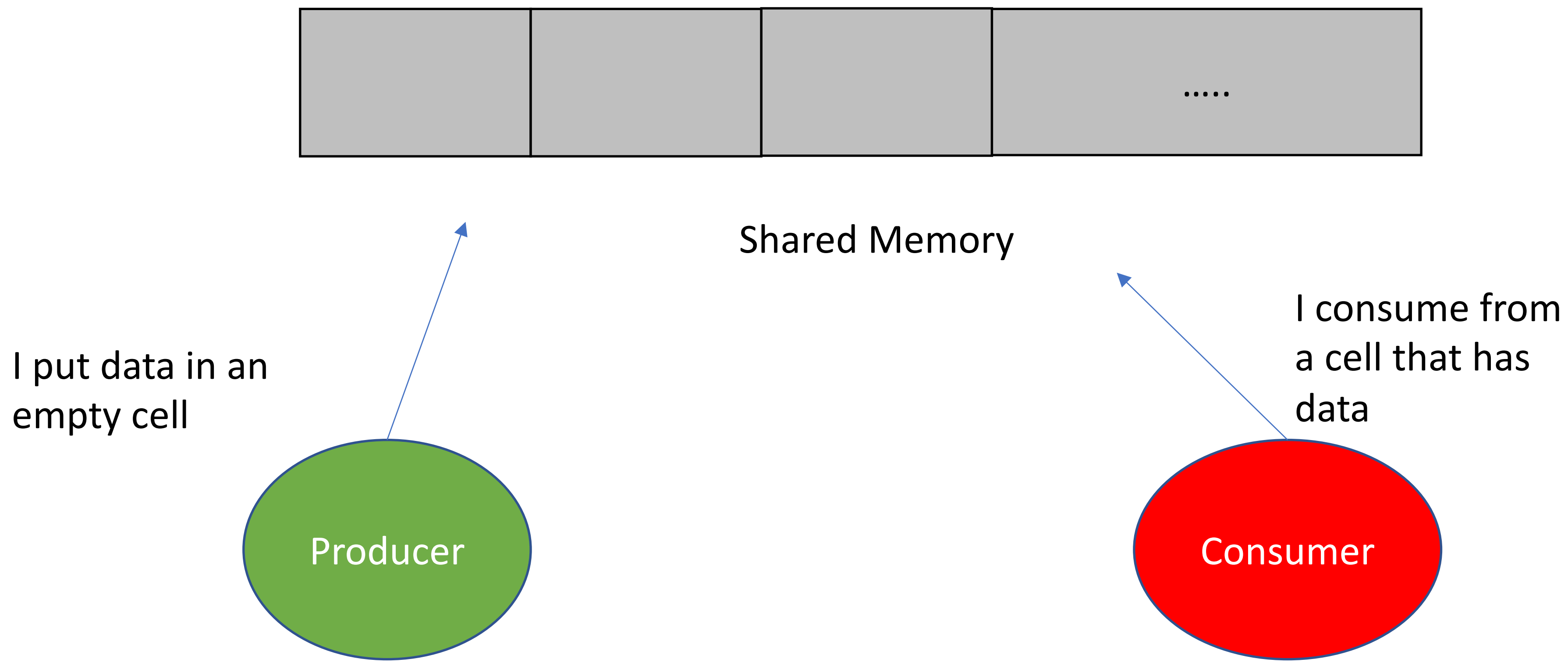
# Counting Semaphore

```
class Semaphore {
  int value;
  ProcessList pl;

  void down () {
    value -= 1;
    if (value < 0) {
      // add this process to pl
      pl.enqueue(currentProcess);
      Sleep();
    }
  }

  void up () {
    Process P;
    value += 1;
    if (value <= 0) {
      // remove a process P from
      pl
      P = pl.dequeue();
      Wakeup(P);
    }
  }
}
```

Taken from Dr. Misurda's slides

# The classic Producer-Consumer Problem

Producer and consumer are two processes running in parallel



# Producer-Consumer Problem - Example

- Please NOTE that the order of execution of the producer and consumer processes may differ from what is shown in the example over the next few slides
  - This is because when two processes are running in parallel, you do not really know the order in which they get executed.

# Producer-Consumer Problem - Example

Semaphores

full = 0
empty = 3
mutex = 1

Producer

```
while(true)
    down(empty)
    down(mutex)

    //Produce

    up(mutex)
    up(full)
```

Shared Memory

Consumer

```
while(true)
    down(full)
    down(mutex)

    //Consume

    up(mutex)
    up(empty)
```

Here, even if the consumer wants to consume first, it will have to wait because full = 0. So, down(full) will add consumer to list of waiting processes.

# Producer-Consumer Problem - Example

Semaphores

full = -1
empty = 3
mutex = 1

Producer

```
while(true)
    down(empty)
    down(mutex)

    //Produce

    up(mutex)
    up(full)
```

Shared Memory

Consumer

```
while(true)
    down(full)
    down(mutex)

    //Consume

    up(mutex)
    up(empty)
```

Consumer ENQUEUED in the list of waiting processes and then put to sleep.

# Producer-Consumer Problem - Example

Semaphores

full = -1
empty = 2
mutex = 1

Producer

```
while(true)
    down(empty)
    down(mutex)

    //Produce

    up(mutex)
    up(full)
```

Shared Memory

Consumer

```
while(true)
    down(full)
    down(mutex)

    //Consume

    up(mutex)
    up(empty)
```

# Producer-Consumer Problem - Example

Semaphores

full = -1
empty = 2
mutex = 0

Producer

```
while(true)
    down(empty)
    down(mutex)

    //Produce

    up(mutex)
    up(full)
```

Shared Memory

Consumer

```
while(true)
    down(full)
    down(mutex)

    //Consume

    up(mutex)
    up(empty)
```

# Producer-Consumer Problem - Example

Producer

```
while(true)
    down(empty)
    down(mutex)

    //Produce

    up(mutex)
    up(full)
```

Semaphores

full = -1
empty = 2
mutex = 0

Shared Memory

Consumer

```
while(true)
    down(full)
    down(mutex)

    //Consume

    up(mutex)
    up(empty)
```

# Producer-Consumer Problem - Example

Semaphores

full = -1
empty = 2
mutex = 1

Producer

```
while(true)
    down(empty)
    down(mutex)

    //Produce

    up(mutex)
    up(full)
```

Shared Memory

Consumer

```
while(true)
    down(full)
    down(mutex)

    //Consume

    up(mutex)
    up(empty)
```

# Producer-Consumer Problem - Example

Semaphores

full = 0
empty = 2
mutex = 1

Producer

```
while (true)
    down(empty)
    down(mutex)

    //Produce

    up(mutex)
    up(full)
```

Shared Memory

Consumer

```
while(true)
    down(full)
    down(mutex)

    //Consume

    up(mutex)
    up(empty)
```

Consumer DEQUEUED from list of waiting processes and then wakes up

# Producer-Consumer Problem - Example

Producer

```
while (true)
    down(empty)
    down(mutex)

    //Produce

    up(mutex)
    up(full)
```

Semaphores

full = 0
empty = 1
mutex = 0

Shared Memory

Consumer

```
while(true)
    down(full)
    down(mutex)

    //Consume

    up(mutex)
    up(empty)
```

Now suppose Consumer executes down (mutex) before Producer

# Producer-Consumer Problem - Example

Semaphores

full = 0
empty = 1
mutex = -1

Producer

```
while (true)
    down(empty)
    down(mutex)

    //Produce

    up(mutex)
    up(full)
```

Shared Memory

Consumer

```
while(true)
    down(full)
    down(mutex)

    //Consume

    up(mutex)
    up(empty)
```

Now Producer waits for the mutex. Gets added to the list of waiting processes and put to sleep

# Producer-Consumer Problem - Example

Producer

```
while (true)
    down(empty)
    down(mutex)

    //Produce

    up(mutex)
    up(full)
```

Semaphores

full = 0
empty = 1
mutex = 0

Shared Memory

Consumer

```
while(true)
    down(full)
    down(mutex)

    //Consume

    up(mutex)
    up(empty)
```

Producer DEQUEUED from the list of waiting processes and then of wakes up

# Producer-Consumer Problem - Example

Semaphores

full = 0
empty = 2
mutex = 0

Producer

```
while (true)
    down(empty)
    down(mutex)

    //Produce

    up(mutex)
    up(full)
```

Shared Memory

Consumer

```
while(true)
    down(full)
    down(mutex)

    //Consume

    up(mutex)
    up(empty)
```

# Producer-Consumer Problem - Example

Semaphores

full = -1
empty = 2
mutex = 1

Producer

```
while (true)
    down(empty)
    down(mutex)

    //Produce

    up(mutex)
    up(full)
```

Shared Memory

Consumer

```
while(true)
    down(full)
    down(mutex)

    //Consume

    up(mutex)
    up(empty)
```

Consumer DEQUEUED from the list of waiting processes and then wakes up

And this goes on …

# Producer-Consumer Problem - Example

- Please NOTE that the order of execution of the producer and consumer processes may differ from what is shown in the example over the previous few slides
  - This is because when two processes are running in parallel, you do not really know the order in which they get executed.

# For Project 3

- Use sem_wait() and sem_post()
- You are using a binary semaphore (between 0 and 1)
- Store the semaphore in shared space so that the same copy of it can be accessed by all the child processes
  - Use mmap
  - If you do not allocate semaphore in the shared space, then every new child process will have its own copy of semaphore
    - This will not ensure mutually exclusive access to critical region