Semaphores

January 21, 2019

- Semaphores
- Producer/Consumer
- Implementing Semaphores
- Bounded Semaphores
- Using Java Semaphores to Implement a Bounded Buffer

Introduction

1: Semaphores

1.1: Disadvantages of synchronisation by communication

- complex algorithms
- time wasting — busy waits
- primitive (atomic) actions are too weak

We need an atomic read/write action, and a mechanism to allow processes to "sleep" — semaphores.

Semaphores

1.2: Semaphores

Introduced by *Edsgar Dÿkstra*.

P() *Passeren* (Pass)
[A better name might be *prolagen*, from *proberen* (try) and *verlagen* (reduce)].

V() *Vrÿgeven* (Free)

English mnemonic — **P**oll and **V**ote

A process that calls **P()** when **s = 0** can only complete the call when **s ≠ 0**, after a **V()** operation — they must "sleep".

The order in which processes are woken is unspecified, but it must be *fair*.

Binary semaphores

1.3: Binary and split binary semaphores

Binary semaphore:

Can only take values **0** or **1**

$$\mathbf{0 \leq s.value() \leq 1}$$

Split binary semaphore:

Total value is **0** or **1**

$$\mathbf{0 \leq s_1.value() + s_2.value() \leq 1}$$

Properties can be *programmer's* responsibility — but sometimes only binary (split) semaphores provided.

Producer/consumer problem

2: Producer/Consumer

2.1: Introduction

Models many "buffering" problems

- Producer produces data, snacks, sproglets, ...
- Consumer consumes data, snacks, sproglets, ...
- Buffer
  - synchronises exchange
  - "smooths out" speed differences

Producer/consumer problem

- Initial assumptions
  - infinite buffer
  - simultaneous access for producer/consumer
- Later
  - access to buffer in critical section
  - finite buffer — bounded buffer problem

Producer and Consumer

2.2: The producer and consumer

producer: `buffer.put(item)`	**consumer**: `buffer.get()`

`public void producer() {`
`  while (true) {`
`  // build item to add`
`  buffer.put(item);`
`  }`
`}`
	`public void consumer() {`
	`  while (true) {`
	`  T item = buffer.get();`
	`  // do something with item`
	`  }`
	`}`

Basic algorithm

2.3: Basic algorithm

infinite buffer, simultaneous access

producer: `buffer.put(item)`	**consumer**: `buffer.get()`

`public void put(T item) {`
`  putItem(item);`
`  noOfElements.V();`
`}`
	`public T get() {`
	`  noOfElements.P();`
	`  T item = getItem();`
	`  return item`
	`}`

Here `noOfElements.value()` = no. of elements in buffer.

Producer/consumer problem

2.4: Forbidding simultaneous access

putItem and getItem in critical section

producer: buffer.put(item)	**consumer**: buffer.get()

public void put() {
criticalSection.P();
putItem(item);
criticalSection.V();
noOfElements.V();
}
	public T get() {
	noOfElements.P();
	criticalSection.P();
	T item = getItem();
	criticalSection.V();
	return item
	}

Initialise noOfElements $= 0$, criticalSection $= 1$.

Bounded buffer

2.5: Bounded buffer

producer: buffer.put(item)	**consumer**: buffer.get()

public void put() {
  noOfSpaces.P();
  criticalSection.P();
  putItem(item);
  criticalSection.V();
  noOfElements.V();
}
	public void get() {
	  noOfElements.P();
	  criticalSection.P();
	  T item = getItem();
	  criticalSection.V();
	  noOfSpaces.V();
	  return item;
	}

Implementing Semaphores

3: Implementing Semaphores
3.1: class `Semaphore`: Fields

```
// Value of the semaphore
private int value = 0;
```

Implementing Semaphores

3.2: class `Semaphore`: Constructors

```
// Initialise value to default value of zero
protected Semaphore() {
    value = 0;
}

// Initialise to the specified value
protected Semaphore(int initial) {
    value = initial;
}
```

Implementing Semaphores

3.3: class Semaphore: Methods

```
// Poll method
public synchronized void poll()
  throws InterruptedException {
    value--;
    if (value < 0) {
        wait()
    }
}

// Vote method
public synchronized void vote() {
    value++;
    if (value <= 0) {
        notify();
    }
}
```

Synchronized Methods

An Aside: synchronized Methods

A **synchronized** method is one that may only be executed by at most one process at any one time.

The **wait**() and **notify**() methods can be used to further refine processes' behaviour in **synchronized** methods:

- A process calling **wait**() will go to sleep
- The sleeping process will sleep until another process calls **notify**().

Implementing Semaphores

4: Bounded Semaphores

Three possibilities:

4.1: Absorbing Semaphores

Once the semaphore reaches its limit it stops incrementing its value.

	0	**1**	$\cdots$	**i**	$\cdots$	**N − 1**	**N**
`P()`	wait	**0**	$\cdots$	**i − 1**	$\cdots$	**N − 2**	**N − 1**
`V()`	**1**	**2**	$\cdots$	**i + 1**	$\cdots$	**N**	**N**

Implementing Semaphores

4.2: Crashing Semaphores

Once the semaphore reaches its limit any attempt to increase its value by a V() causes it to crash.

	0	**1**	$\cdots$	**i**	$\cdots$	**N − 1**	**N**
P()	wait	**0**	$\cdots$	**i − 1**	$\cdots$	**N − 2**	**N − 1**
V()	**1**	**2**	$\cdots$	**i + 1**	$\cdots$	**N**	✠

Implementing Semaphores

4.3: Blocking Semaphores

Once the semaphore reaches its limit any attempt to increase its value by a V() causes it to block.

	0	**1**	$\cdots$	**i**	$\cdots$	**N − 1**	**N**
P()	wait	**0**	$\cdots$	**i − 1**	$\cdots$	**N − 2**	**N − 1**
V()	**1**	**2**	$\cdots$	**i + 1**	$\cdots$	**N**	wait

Semaphores in Java

5: Using Java Semaphores to Implement a Bounded Buffer

Need to **import** `java.util.concurrent.Semaphore;`

5.1: class `Buffer⟨T⟩`**: Fields**

```
private Semaphore noOfSpaces, noOfElements, criticalSection;
private T[] buffer;
private int putIndex = 0, getIndex = 0;
```

Semaphores in Java

5.2: class Buffer⟨T⟩: Constructor

```
public Buffer(int size) {
    buffer = new T[size]; // can't actually do this in Java
    noOfSpaces = new Semaphore(size,true); // is fair
    noOfElements = new Semaphore(0);
    criticalSection = new Semaphore(1);
}
```

Semaphores in Java

5.3: class Buffer⟨T⟩: The put method

```
public void put(T item) {
    try {
        noOfSpaces.acquire();
        criticalSection.acquire();
        addItem(item);
        criticalSection.release();
        noOfElements.release();
    } catch (InterruptedException ie) {
        throw new BufferError("Data item not added\n" +
            ie.getMessage());
    }
}
```

Semaphores in Java

5.4: class Buffer⟨T⟩: The get method

```
public T get() {
    try {
        noOfElements.acquire();
        criticalSection.acquire();
        T item = getItem();
        criticalSection.release();
        noOfSpaces.release();
        return item;
    }  catch (InterruptedException ie) {
        throw new BufferError("Data item not fetched\n" +
            ie.getMessage());
    }
}
```

Semaphores in Java

5.5: class Buffer⟨T⟩: Update methods

```
private void addItem(T item) {
    buffer[putIndex] = item;
    putIndex = (putIndex + 1) % buffer.length;
}

private T getItem() {
    T item = buffer[getIndex];
    getIndex = (getIndex + 1) % buffer.length;
    return item;
}
```

End of semaphores lecture