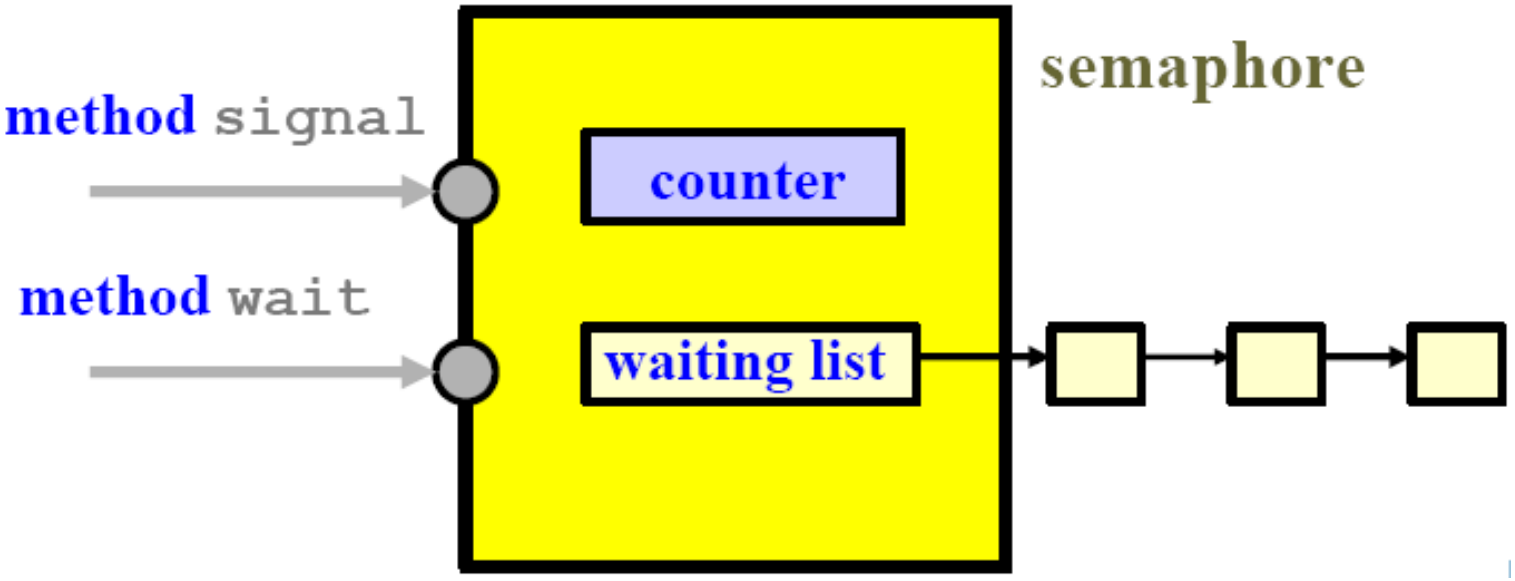
# بعض الأمثلة على إستخدام السيمافور و الميوتكس

د. فادي تركاوي

# السيمافور

- A *semaphore* is an object that consists of a counter, a waiting list of processes and two methods (*e.g.*, functions): `signal` and `wait`.



# Skeleton Code for Binary Semaphore

- wait(sem): if sem = 1 then sem:= 0; else place process in sem.queue;
- signal(sem): if sem.queue not empty
- then remove a process from the queue and place in ready list
- else sem :=1;
- A binary semaphore allows just one process in the critical region at a time (mutual exclusion)
    - السيمافور الثنائي يسمح فيه لعملية واحدة بالدخول للمنطقة الحرجة.

## Skeleton Code for Counting Semaphore

- السيمافور المتعدد يكون فيه متحول السمافور له القيمة n و في هذه الحالة يسمح لعدد n من العمليات بالدخول للمنطقة الحرجة.

- Counting semaphore is initialized to "n"
- It then allows n processes into critical region or allocation of n resources
- wait(sem): s:=s-1; if s<0 place process on sem.queue
- signal(sem): s:=s+1; if s <= n then move process from sem.queue to ready list

# مثال برمجي

```
semaphore S1 = 1, S2 = 0;
```

process 1

```
while (1) {
    // do something
    S1.wait();
        cout << "1";
    S2.signal();
    // do something
}
```

process 2

```
while (1) {
    // do something
    S2.wait();
        cout << "2";
    S1.signal();
    // do something
}
```

notify

notify

- Process 1 uses `S2.signal()` to notify process 2, indicating "I am done. Please go ahead."
- The output is `1 2 1 2 1 2 .....`
- What if both `S1` and `S2` are both 0's or both 1's?
- What if `S1` = 0 and `S2` = 1?

# Protecting a critical region
# حماية المنطقة الحرجة بسيمافور

- sem mutex =1;
  process CS[i = 1 to n] {
  while (true) {
  wait(mutex);
  critical region;
  signal(mutex);
  noncritical region;
  }
  }

# Implementing a 2-process barrier
# حاجز ضمان وصول عمليتان

- Neither process can pass the barrier until both have arrived.
- Must be able continuously reuse the barrier; therefore it must reinitialize after letting processes pass the barrier.
  - This will require two semaphores: a signaling semaphore for each of arrival and departure.
  - Each process x signals its arrival using a V(arrivex) and then waits on the other process' (y) semaphore with a P(arrivey);
- sem arrive1 = 0, arrive2 = 0;

```
process Worker1 {
...
S1.signal(); /*signal arrival */
s2.wait(); /*await arrival of other process */
...
}
```

```
•process Worker2
{
...
S2.signal();
s1.wait ();
..
}
```

# Protecting a critical region
# حل مشكلة القارئ الكاتب بإستخدام سيمافور

- Use a single shared buffer
- Only one process (producer or consumer) can be reading or writing the buffer at one time
- Producers put things in the buffer with a **deposit**
- Consumers take things out of the buffer with a **fetch**
- Need two semaphores:
  - **empty** will keep track of whether the buffer is empty
  - **full** will keep track of whether the buffer is full

# Protecting a critical region
# حل مشكلة القارئ الكاتب بإستخدام سيمافور

- typeT buf; /* a buffer of some type */
  sem empty =1; /*initially buffer is empty */
  sem full = 0; /*initially buffer is not full */
- process Producer [i = 1 to m] {
  while (true) {
  ...
  /*produce data, then depositi it in th
  empty.wait();
  buf = data;
  full.signal();
  }
  }

- process Consumer [j=1 to n] {
  while (true) {
  full.wait();
  result = buf;
  empty.signal();
  ...
  }
  }

# simple Semaphore implementation (للإطلاع) in JAVA

```
package com.javapapers.thread;
 public class BinarySemaphore {
      private boolean locked = false;
               BinarySemaphore(int initial) {
                        locked = (initial == 0);
               }
public synchronized void waitForNotify() throws InterruptedException {
      while (locked) { wait(); }
               locked = true;
 }
 public synchronized void notifyToWakeup() {
      if (locked) { notify(); }
      locked = false;
}
}
```

# السيمافور بقيم متعددة counting Semaphor ( للإطلاع )

```
package com.javapapers.thread;
public class CountingSemaphore {
private int value = 0;
private int waitCount = 0;
private int notifyCount = 0;
public CountingSemaphore(int initial) {
if (initial > 0) {
value = initial;
}
}
public synchronized void waitForNotify() {
if (value <= waitCount) {
waitCount++;
try {
do {
wait();
} while (notifyCount == 0);
} catch (InterruptedException e) {
notify();
} finally {
waitCount--;
}
notifyCount--;
}
value--;
}
public synchronized void notifyToWakeup() {
value++;
if (waitCount > notifyCount) {
notifyCount++;
notify();
}    }    }
```

# المصطلحات المستخدمة في كل البرمجيات لعمليات السيمافورات

- In software engineering practice, they are often called *signal* and *wait*, *release* and *acquire* (which the standard Java library uses), or *post* and *pend*. Some texts call them *vacate* and *procur*.

# ملف الفلاش ١ مطلوب في المسائل و يجب تحميله

process sync using semaphor.swf

# ملف الفلاش ٢ مطلوب في المسائل و يجب تحميله

sema.swf

# Coke machine


```
/* number of full slots (Cokes) in machine */
semaphore fullSlot = 0;
/* number of empty slots in machine */
semaphore emptySlot = 100;


DeliveryPerson()
{
    emptySlot.P( );        /* empty slot avail? */
    mutex.P( );            /* exclusive access */
    put 1 Coke in machine
    mutex.V( );
    fullSlot.V( );         /* another full slot! */
}

ThirstyPerson()
{
    fullSlot.P( );         /* full slot (Coke)? */
    mutex.P( );            /* exclusive access */
    get 1 Coke from machine
    mutex.V( );
    emptySlot.V( );        /* another empty slot! */
}
```

## الخلاصة:

بالطبع و على نفس المبدأ بالإضافة لأن السيمافور يؤمن الإستبعاد المتبادل و التزامن للمصادرفي الحاسب فهو أيضا له نفس المبدأ للتحكم بالآلات المبرمجة لـــ :

- لآلات الصودا (المشروب الغازي)
- ألات النقود ATM
- ألات غسل الصحون Dish washer
- .....