BITS, PILANI – K. K. BIRLA GOA CAMPUS

Operating Systems

by

Dr. Shubhangi

PROCESS SYNCHRONIZATION

Semaphores

We don't want to loop on busy, so will suspend instead:

- Block on semaphore == False,
- Wakeup on signal (semaphore becomes True),
- There may be numerous processes waiting for the semaphore, so keep a list of blocked processes,
- Wakeup one of the blocked processes upon getting a signal (choice of who depends on strategy).

To PREVENT looping, we redefine the semaphore structure as:

```
typedef struct {
   int                  value;
   struct process       *list;      /*  linked list of PTBL waiting on  S  */
} SEMAPHORE;
```

PROCESS SYNCHRONIZATION

Semaphores

```
typedef struct {
    int                 value;
    struct process      *list;      /*  linked list of PTBL waiting on  S  */
} SEMAPHORE;
```

```
SEMAPHORE s;
wait(s) {
      s.value  =  s.value  -  1;
      if (  s.value  <  0 ) {
            add this process to s.L;
block;
      }
}
```

```
SEMAPHORE s;
signal(s)  {
      s.value  =  s.value  +  1;
      if (  s.value  <=  0 )  {
remove a process P from s.L;
            wakeup(P);
      }
}
```

- It's critical that these be atomic - in uniprocessors we can disable interrupts, but in multiprocessors other mechanisms for atomicity are needed.
- Popular incarnations of semaphores are as "event counts" and "lock managers". (We'll talk about these in the next chapter.)

PROCESS SYNCHRONIZATION

Semaphores

DEADLOCKS:

- May occur when two or more processes try to get the same multiple resources at the same time.

P1:

```
wait(S);
wait(Q);
.....
signal(S);
signal(Q);
```

P2:

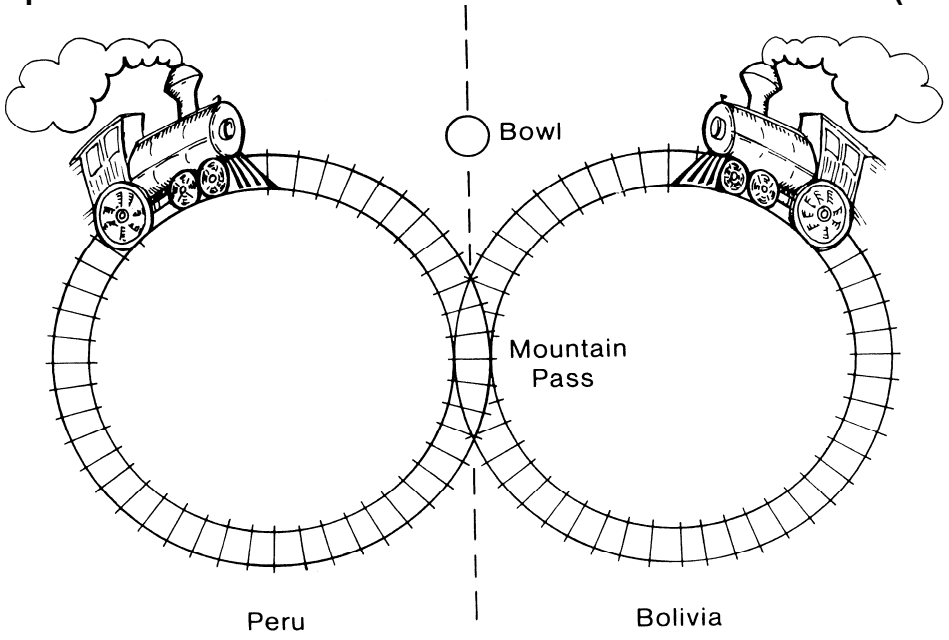
```
wait(Q);
wait(S);
.....
signal(Q);
signal(S);
```

- How can this be fixed?

PROCESS SYNCHRONIZATION

Railways in the Andes; A Practical Problem

High in the Andes mountains, there are two circular railway lines. One line is in Peru, the other in Bolivia. They share a common section of track where the lines cross a mountain pass that lies on the international border (near Lake Titicaca?).



Unfortunately, the Peruvian and Bolivian trains occasionally collide when simultaneously entering the common section of track (the mountain pass). The trouble is, alas, that the drivers of the two trains are both **blind** and **deaf**, so they can neither see nor hear each other.

The two drivers agreed on the following method of preventing collisions. They set up a large bowl at the entrance to the pass. Before entering the pass, a driver must stop his train, walk over to the bowl, and reach into it to see it it contains a rock. If the bowl is empty, the driver finds a rock and drops it in the bowl, indicating that his train is entering the pass; once his train has cleared the pass, he must walk back to the bowl and remove his rock, indicating that the pass in no longer being used. Finally, he walks back to the train and continues down the line.

If a driver arriving at the pass finds a rock in the bowl, he leaves the rock there; he repeatedly takes a siesta and rechecks the bowl until he finds it empty. Then he drops a rock in the bowl and drives his train into the pass. A smart graduate from the University of La Paz (Bolivia) claimed that subversive train schedules made up by Peruvian officials could block the train forever.

Explain

The Bolivian driver just laughed and said that could not be true because it never happened.

Explain

Unfortunately, one day the two trains crashed.

Explain

Following the crash, the graduate was called in as a consultant to ensure that no more crashes would occur. He explained that the bowl was being used in the wrong way. The Bolivian driver must wait at the entry to the pass until the bowl is empty, drive through the pass and walk back to put a rock in the bowl. The Peruvian driver must wait at the entry until the bowl contains a rock, drive through the pass and walk back to remove the rock from the bowl. Sure enough, his method prevented crashes.

Prior to this arrangement, the Peruvian train ran twice a day and the Bolivian train ran once a day. The Peruvians were very unhappy with the new arrangement.

Explain

The graduate was called in again and was told to prevent crashes while avoiding the problem of his previous method. He suggested that two bowls be used, one for each driver. When a driver reaches the entry, he first drops a rock in his bowl, then checks the other bowl to see if it is empty. If so, he drives his train through the pass. Stops and walks back to remove his rock. But if he finds a rock in the other bowl, he goes back to his bowl and removes his rock. Then he takes a siesta, again drops a rock in his bowl and re-checks the other bowl, and so on, until he finds the other bowl empty. This method worked fine until late in May, when the two trains were simultaneously blocked at the entry for many siestas.

Explain

PROCESS SYNCHRONIZATION

Some Interesting Problems

THE BOUNDED BUFFER (PRODUCER / CONSUMER) PROBLEM:

This is the same producer / consumer problem as before. But now we'll do it with signals and waits. Remember: a **wait decreases** its argument and a **signal increases** its argument.

```
BINARY_SEMAPHORE      mutex = 1;                  // Can only be 0 or 1
COUNTING_SEMAPHORE    empty = n;   full = 0;      // Can take on any integer value
```

```
producer:
do {
     /* produce an item in nextp */
     wait (empty);        /* Do action     */
     wait (mutex);        /* Buffer guard*/
     /* add nextp to buffer  */
     signal (mutex);
     signal (full);
  } while(TRUE);
```

```
consumer:
do {
     wait (full);
     wait (mutex);
     /* remove an item from buffer to nextc */
     signal (mutex);
     signal (empty);
     /* consume an item in nextc */
   } while(TRUE);
```