Operating System

Unit – 3
(Part-B)
Process Synchronization

Mayank Mishra
School of Electronics Engineering
KIIT-Deemed to be University

Mutex Locks

- The hardware-based solutions to the critical-section problem are complicated as well as generally inaccessible to application programmers.

- Instead, operating-system designers build higher-level software tools to solve the critical-section problem. The simplest of these tools is the mutex lock.(In fact, the term **mutex** is short for **mut**ual **ex**clusion.)

- Mutex Lock is used to protect critical sections and thus prevent race conditions. That is, a process must acquire the lock before entering a critical section; it releases the lock when it exits the critical section.

- The ***acquire()*** function acquires the lock, and the ***release()*** function releases the lock, as illustrated in Figure.

```
while (true) {

      acquire lock

            critical section

      release lock

            remainder section

}
```

Fig. Solution to the critical-section problem using mutex locks

Mutex Locks (Contd.)

- A mutex lock has a boolean variable available whose value indicates if the lock is available or not.

- If the lock is available, a call to **acquire()** succeeds, and the lock is then considered unavailable. A process that attempts to acquire an unavailable lock is blocked until the lock is released.

The definition of acquire() is as follows:

```
acquire() {
   while (!available)
      ; /* busy wait */
   available = false;
}
```

The definition of release() is as follows:

```
release() {
  available = true;
}
```

Mutex Locks (Contd.)

Process P1

The definition of acquire() is as follows:

```
acquire() {
   while (!available)
     ; /* busy wait */
   available = false;
}
```

The definition of release() is as follows:

```
release() {
  available = true;
}
```

Process P2

The definition of acquire() is as follows:

```
acquire() {
   while (!available)
     ; /* busy wait */
   available = false;
}
```

The definition of release() is as follows:

```
release() {
  available = true;
}
```

- Calls to either or **acquire() release()** must be performed atomically**.**

- ***An atomic operation in an operating system (OS) is a sequence of instructions that are executed as a single unit without interruption.***

- The main disadvantage of the implementation given here is that it requires **busy waiting**.

- While a process is in its critical section, any other process that tries to enter its critical section must loop continuously in the call to **acquire().** This continual looping is clearly a problem in a real multiprogramming system, where a single CPU core is shared among many processes. Busy waiting also wastes CPU cycles that some other process might be able to use productively.

- The type of mutex lock we have been describing is also called a **Spinlock** because the process “spins” while waiting for the lock to become available.

- **Spinlocks do have an advantage**, however, in that **no context switch is required** when a process must wait on a lock, and a context switch may take considerable time.

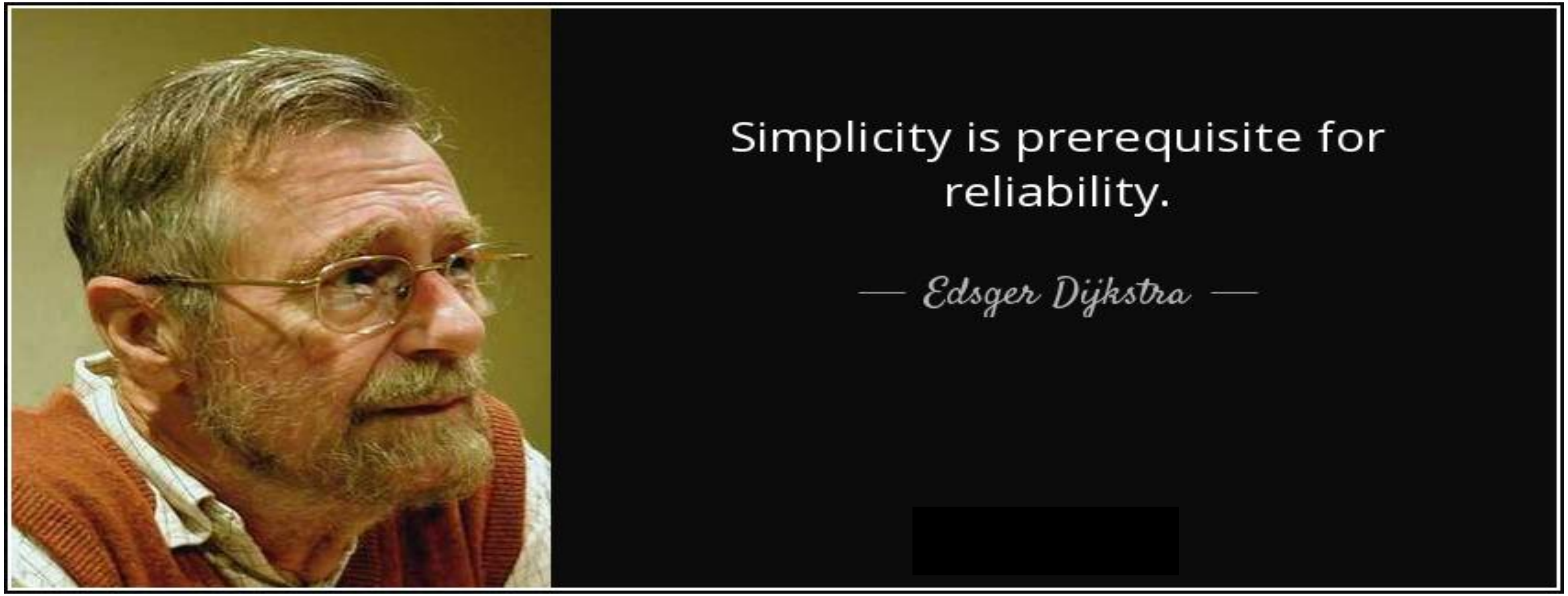
- In certain circumstances on multicore systems, spinlocks are in fact the preferable choice for locking. If a lock is to be held for a short duration, one thread can “spin” on one processing core while another thread performs its critical section on another core.

- On modern multicore computing systems, spinlocks are widely used in many operating systems.

Semaphores

- Semaphore proposed by Dutch computer scientist **Edsger Dijkstra**, is a technique to manage concurrent processes by using a simple integer value, which is known as **Semaphores**.

- A semaphore **S** is an integer variable that, apart from initialization, is accessed only through two standard atomic operations: **wait()** and **signal().**



Semaphores (Contd.)

wait ( ) → P [from the Dutch word proberen, which means "to test"]

signal ( ) → V [from the Dutch word verhogen, which means " to increment"]

- The **wait()** operation was originally termed **P** (from the Dutch ***proberen***, “to test”); **signal()** was originally called **V** (from ***verhogen***, “to increment”).

Semaphores (Contd.)

The definition of wait() is as follows:

```
wait(S) {
    while (S <= 0)
        ; // busy wait
    S--;
}
```

The definition of signal() is as follows:

```
signal(S) {
    S++;
}
```

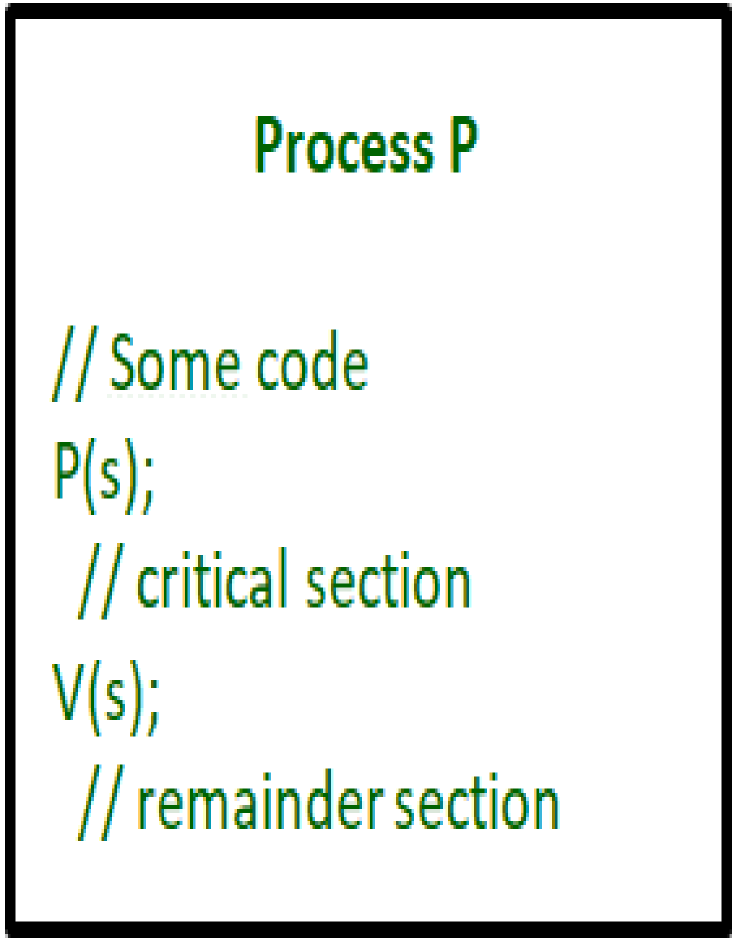
Semaphores (Contd.)

- All modifications to the integer value of the semaphore in the **wait()** and **signal()** operations must be executed **atomically**. That is, when one process modifies the semaphore value, no other process can simultaneously modify that same semaphore value.

- In addition, in the case of **wait(S)**,the testing of the integer value of S ($S \leq 0$), as well as its possible modification (S--), must be executed without interruption.

- Semaphores are of two types:
  - **Binary Semaphore:** This is also known as a **mutex lock**, as they are locks that provide mutual exclusion. It can have only two values: **0 and 1**. **Its value is initialized to 1**. It is used to implement the solution of critical section problems with multiple processes and a single resource.
  - **Counting Semaphore:** Unlike a binary semaphore, which can only take values 0 and 1, a counting semaphore can take any non-negative integer value, which represents the number of available resources. A counting semaphore is initialized with a positive integer value that represents the number of available resources. For example, if a semaphore is initialized to 3, it indicates that there are 3 resources available for use.

- **A critical section is surrounded by both operations to implement process synchronization** (The figure demonstrates the basic mechanism of how semaphores are used to control access to a critical section in a multi-process environment, ensuring that only one process can access the shared resource at a time).



Semaphores (Contd.)

- The main disadvantage of the semaphore definition that was discussed is that it requires **busy waiting.**

- While a process is in its critical section, any other process that tries to enter its critical section must loop continuously in the entry code.

- Busy waiting wastes CPU cycles that some other process might be able to use productively.

- This type of semaphore is also called a **spinlock** because the process “spins” while waiting for the lock.

To overcome the need for busy waiting, we can modify the definition of the wait ( ) and signal ( ) semaphore operations

- When a process executes the wait( ) operation and finds that the semaphore value is not positive, it must wait.

- However, rather than engaging in busy waiting, the process can block itself.

- The block operation places a process into a waiting queue associated with the semaphore, and the state of the process is switched to the waiting state.

- Then control is transferred to the CPU scheduler, which selects another process to execute.

Binary Semaphores

Down( )/ Wait ( )/ P ( )

```
Down (semaphore s)
{
  if (s.value == 1)
  {
    s.value=0;
  }
  else
  {
   sleep(); // Block this process and place in suspend list
  }
}
```

Up( )/ Signal ( )/ V ( )/ Post ( )/ Release ( )

```
Up (semaphore s)
{
  if (Suspend List is Empty)
  {
    s.value=1;
  }
  else
  {
        wake up(); // Block to Ready Queue
  }
}
```

Counting Semaphores

Down( )/ Wait ( )/ P ( )

```
Down (semaphore s)
{
   s.value = s.value - 1;
   if (s.value < 0)
   {
sleep(); // Block this process and place in suspend list
   }
   else
      return;
}
```

Up( )/ Signal ( )/ V ( )/ Post ( )/ Release ( )

```
Up (semaphore s)
{
   s.value = s.value + 1;
   if (s.value <=0 )
   {
wake up(); // Block to Ready Queue
   }
}
```

#Generally FIFO is followed in the Block Queue

Question 1: A counting semaphore S is initialized to 10. Then, 6 P operations and 4 V operations are performed on S. What is the final value of S?

Question 1: A counting semaphore S is initialized to 10. Then, 6 P operations and 4 V operations are performed on S. What is the final value of S?

Solution:

We know-

P operation also called as wait operation decrements the value of semaphore variable by 1.
V operation also called as signal operation increments the value of semaphore variable by 1.

Thus,
Final value of semaphore variable S

= 10 – 6 + 4

= 8

Question 2: A counting semaphore S is initialized to 7. Then, 20 P operations and 15 V operations are performed on S. What is the final value of S?

Question 2: A counting semaphore S is initialized to 7. Then, 20 P operations and 15 V operations are performed on S. What is the final value of S?

Solution:

We know-

P operation also called as wait operation decrements the value of semaphore variable by 1.

V operation also called as signal operation increments the value of semaphore variable by 1.

Thus,

Final value of semaphore variable S

= 7 – 20 + 15

= 2

Question 3: A shared variable x, initialized to zero, is operated on by four concurrent processes W, X, Y, Z as follows. Each of the processes W and X reads x from memory, increments by one, stores it to memory and then terminates. Each of the processes Y and Z reads x from memory, decrements by two, stores it to memory, and then terminates. Each process before reading x invokes the P operation (i.e. wait) on a counting semaphore S and invokes the V operation (i.e. signal) on the semaphore S after storing x to memory. Semaphore S is initialized to two. What is the **maximum** possible value of x after all processes complete execution?

A) -2
B) -1
C) 2
D) None of the above

Solution:

Process W	Process X	Process Y	Process Z
Wait (S)	Wait (S)	Wait (S)	Wait (S)
Read (x)	Read (x)	Read (x)	Read (x)
x = x + 1;	x = x + 1;	x = x - 2;	x = x - 2;
Write (x)	Write (x)	Write (x)	Write (x)
Signal (S)	Signal (S)	Signal (S)	Signal (S)

Solution:

Process W	Process X	Process Y	Process Z
Wait (S)	Wait (S)	Wait (S)	Wait (S)
Read (x)	Read (x)	Read (x)	Read (x)
x = x + 1;	x = x + 1;	x = x - 2;	x = x - 2;
Write (x)	Write (x)	Write (x)	Write (x)
Signal (S)	Signal (S)	Signal (S)	Signal (S)

$\boxed{\textbf{Final Answer: X=2}}$

Question 4: A shared variable x, initialized to zero, is operated on by four concurrent processes W, X, Y, Z as follows. Each of the processes W and X reads x from memory, increments by one, stores it to memory and then terminates. Each of the processes Y and Z reads x from memory, decrements by two, stores it to memory, and then terminates. Each process before reading x invokes the P operation (i.e. wait) on a counting semaphore S and invokes the V operation (i.e. signal) on the semaphore S after storing x to memory. Semaphore S is initialized to two. What is the **minimum** possible value of x after all processes complete execution?

A) -2
B) -1
C) 2
D) None of the above

Solution:

Process W	Process X	Process Y	Process Z
Wait (S)	Wait (S)	Wait (S)	Wait (S)
Read (x)	Read (x)	Read (x)	Read (x)
x = x + 1;	x = x + 1;	x = x - 2;	x = x - 2;
Write (x)	Write (x)	Write (x)	Write (x)
Signal (S)	Signal (S)	Signal (S)	Signal (S)

Final Answer: X= -4

Question 5: If a counting semaphore present value is 20, which of the following operations will result in semaphore value 27 ?

A) 3P, 10V, 3V, 2P

B) 8P, 5V, 12V, 2P, 2V

C) 7P, 6V, 5V, 3P, 6V

D) 6P, 2V, 5V, 3P, 6V

Question 5: If a counting semaphore present value is 20, which of the following operations will result in semaphore value 27 ?

A) 3P, 10V, 3V, 2P

B) 8P, 5V, 12V, 2P, 2V

C) 7P, 6V, 5V, 3P, 6V

D) 6P, 2V, 5V, 3P, 6V

Correct Answer: Option C

Question 6: Consider a non-negative counting semaphore S. The operation P(S) decrements S, and V(S) increments S. During an execution, 18 P(S) operations and 7 V(S) operations are issued in some order. The largest initial value of S for which atleast one P(S) operation will remain blocked________?

A) 10

B) 8

C) 9

D) None

Question 6: Consider a non-negative counting semaphore S. The operation P(S) decrements S, and V(S) increments S. During an execution, 18 P(S) operations and 7 V(S) operations are issued in some order. The largest initial value of S for which atleast one P(S) operation will remain blocked________?

A) 10

B) 8

C) 9

D) None

$S-18+7 = -1$

Correct Answer: Option A

References

1. Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, “Operating System Concepts,” Eleventh Edition (Willey).

2. Andrew S. Tanenbaum, “Modern Operating Systems”, Fourth Edition (Pearson Publications), 2014.

3. https://www.geeksforgeeks.org/

4. https://www.javatpoint.com/

5. https://www.tutorialspoint.com/

6. https://www.nesoacademy.org/

7. https://www.baeldung.com/

8. https://www.educative.io/