Inter-Process Communication and Synchronization of Processes, Threads and Tasks:

Lesson-9: P and V SEMAPHORES

P and V SEMAPHORES

P and V semaphores

- An efficient synchronisation mechanism
- POSIX 1003.1.b, an IEEE standard.
- POSIX– for portable OS interfaces in Unix.
- P and V semaphores – represents the by integers in place of binary or unsigned integers

P and V semaphore Variables

- The semaphore, apart from initialization, is accessed only through two standard atomic operations─ P and V
- P (for *wait* operation)─ derived from a Dutch word ‘Proberen’, which means 'to test'.
- V (for *signal* passing operation)─ derived from the word 'Verhogen' which means 'to increment'.

P and V Functions for Semaphore

- P semaphore function signals that the task requires a resource and if not available waits for it.
- V semaphore function signals which the task passes to the OS that the resource is now free for the other users.

P Function– P (&Sem1)

1. /* Decrease the semaphore variable*/

sem_1 = sem_1 -1;

2. /* If sem_1 is less than 0, send a message to OS by calling a function waitCallToOS. Control of the process transfers to OS, because less than 0 means that some other process has already executed P function on sem_1. Whenever there is return for the OS, it will be to step 1. */

if (sem_1 < 0){waitCallToOS (sem_1);}

V Function– V (&Sem2)

3. /* Increase the semaphore variable*/

sem_2 = sem_2 + 1;

4. /* If sem_2 is less or equal to 0, send a message to OS by calling a function signalCallToOS. Control of the process transfers to OS, because < or = 0 means that some other process is already executed P function on sem_2. Whenever there is return for the OS, it will be to step 3. */

if (sem_2 < = 0){signalCallToOS (sem_2);}

P and V SEMAPHORE FUNCTIONS WITH SIGNALING OR NOTIFICATION PROPERTY

P and V SEMAPHORE FUNCTIONS WITH SIGNAL OR NOTIFICATION PROPERTY

Process 1 (Task 1):

```
while (true) {
/* Codes */

.
V (&sem_s);
/* Continue Process 1 if sem_s is not equal to
   0 or not less than 0. It means that no process
   is executing at present. */

.
};
```

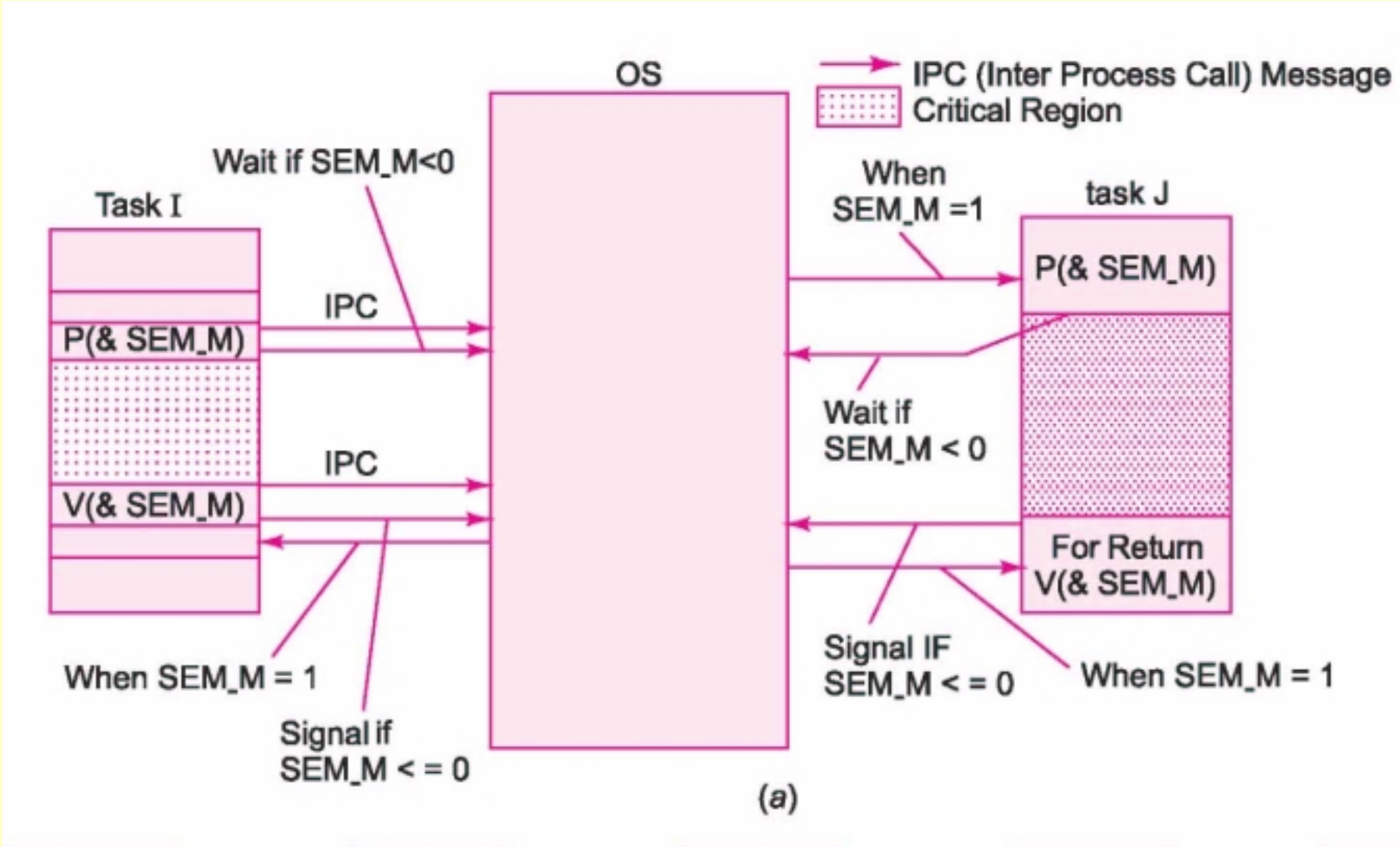
P and V SEMAPHORE FUNCTIONS WITH SIGNAL OR NOTIFICATION PROPERTY

Process 2 (Task 2):

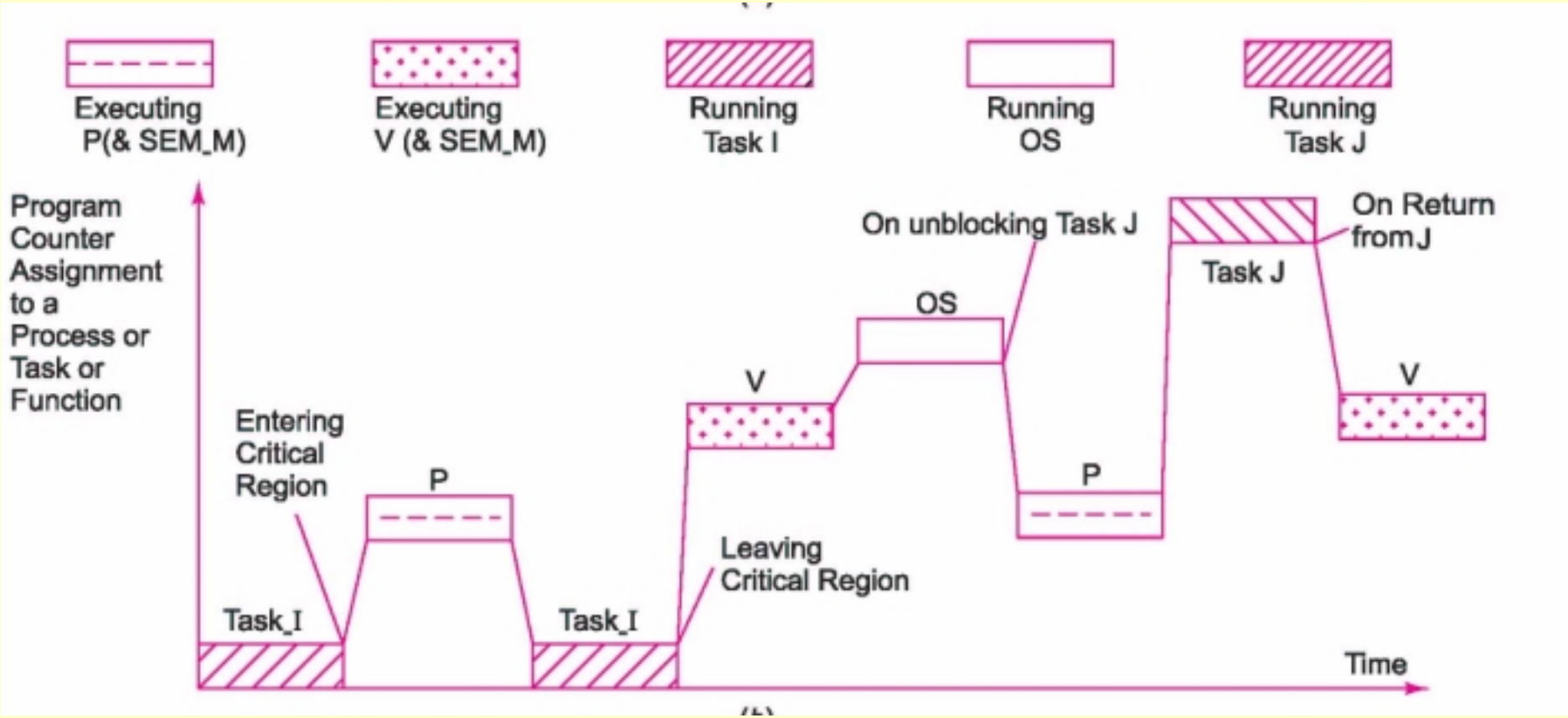
```
while (true) {
/* Codes */
.
P (&sem_s);
/* The following codes will execute only
   when sem_s is not less than 0. */
.
 };
```

P and V SEMAPHORE FUNCTIONS WITH MUTEX PROPERTY

Use of P and V semaphores at the tasks I and J and actions by scheduler



Program counter assignments (context Switch) to the process or function when using P and V semaphores



P and V Semaphore functions with mutex property – Wait for starting Critical Section

Process 1 (Task 1):

```
while (true) {
/* Codes before a critical region*/
.
/* Enter Process 1 Critical region codes*/
P (&sem_m);
/* The following codes will execute only
   when sem_m is not less than 0. */
.
```

P and V Semaphore functions with mutex property – running end exiting Critical Section

Process 1 (Task 1):

.

/* Exit Process 1 critical region codes */

V (&sem_m);

/* Continue Process 1 if sem_m is not equal to 0 or not less than 0. It means that no process is executing at present. */

.

};

P and V Semaphore functions with mutex property ─ Wait for starting other Critical Section

Process 2 (Task 2):

```
while (true) {
/* Codes before a critical region*/
.
/* Enter Process 2 Critical region codes*/
P (&sem_m);
/* The following codes will execute only
   when sem_m is not less than 0. */
.
```

P and V Semaphore functions with mutex property – running end exiting the Critical Section

Process 2 (Task 2):

.

/* Exit Process 2 critical region codes */

V (&sem_m);

/* Continue Process 2 if sem_m is not equal to 0 or not less than 0. It means that no process is executing at present. */

.

};

P and V SEMAPHORE FUNCTIONS WITH COUNTING SEMAPHORE PROPERTY

P and V Semaphore functions with Count property ─ producer Wait for empty place if = 0

Process c (Task c):

```
while (true) {/* sem_c1 represent number of
   empty places*/
/* Codes before a producer region*/
.
./* Exit Process 3 region codes */
P (&sem_c1);
/* Continue Process c if sem_c1 is not equal
   to 0 or not less than 0. */
.
```

P and V SEMAPHORE FUNCTIONS FOR PRODUCER-CONSUMER PROBLEM (BOUNDED BUFFER PROBLEM)

Producer-Consumer and Bounded Buffer Problems

Consider three examples:

- (i) a task transmits bytes to an I/O stream for filling the available places at the stream;
- (ii) a process '*writes*' an I/O stream to a printer-buffer; and
- (iii) a task of producing chocolates is being performed.

Producer-Consumer and Bounded Buffer Problems

- Example (i) another task reads the I/O-stream bytes from the filled places and creates empty places.
- Example (ii), from the print buffer an I/O stream prints after a buffer-read and after printing, more empty places are created.
- Example (iii), a consumer is consuming the chocolates produced and more empty places (to stock the produced chocolates) are created.

Bounded Buffer Problem

- A task blockage operational problem— commonly called Bounded Buffer Problem.
- Example (i)— A task cannot transmit to the I/O stream if there are no empty places at the stream.

Bounded Buffer Problem

- Example (ii)– The task cannot write from the memory to print buffer if there are no empty places at the print-buffer.

Producer-Consumer Problem

- Example (iii)– The producer cannot produce chocolates if there are no empty places at the consumer end.

P and V Semaphore functions with producer-consumer problem solution

Process 3 (Task 3):

```
while (true) {
/* Codes before a producer region. sem_c2
   number of empty places created by process
   4 */

.

/* Enter Process 3 Producing region codes*/
P (&sem_c2);
/* The following codes will execute only
   when sem_c2 is not less than 0. */
```

P and V Semaphore functions with producer-consumer problem solution

Process 3 (Task 3):

.

/* Exit Process 3 region codes */

V (*&sem_c1*);

/* Continue Process 3 if sem_c1 is not equal to 0 or not less than 0. */

.

};

P and V Semaphore functions with producer-consumer problem solution

Process 4 (Task 4):

```
while (true) {/* Codes before a producer
   region. sem_c1 number of filled places
   created by process 3 */

.
/* Enter Process 4 Consuming region codes*/
P (&sem_c1);
/* The following codes will execute only
   when sem_c1 is not less than 0. */
.
```

P and V Semaphore functions with producer-consumer problem solution

Process 4 (Task 4):

.

/* Exit Process 4 region codes */

V (*&sem_c2*);

/* Continue Process 4 if sem_m is not equal to 0 or not less than 0. It means that no process is executing at present. */

.

};

Summary

We learnt

- P and V semaphore functions are POSIX 100.3b IEEE accepted standard for the semaphore IPCs.
- These can be used as event signaling, as mutex, as counting semaphore and as semaphores for the bounded buffer problem solution (producer–consumer problem)

End of Lesson 9 of Chapter 7