



**Computer Engineering Department &**  
**Information Technology Engineering Department**

Academic Year: 2021-2022

Class: S.Y.B.Tech Sem.: 4 Course: OS

<b>Name</b>	<b>Pratik Pujari</b>		
<b>UID no.</b>	<b>2020300054</b>	<b>Class:</b>	<b>Comps C Batch</b>
<b>Experiment No.</b>	8		

<b>AIM:</b>	Process Synchronization using Semaphore.
<b>QUESTION:</b>	The program r.c initializes n number of semaphores. It first assign count equal "-1," which is then used by process p and q. This count is protected by semaphore. It also allocates shared memory of size 40 ints. It waits for process p and q to enter all n1 and n2 elements through different terminals. This program r.c sorts shared data in ascending order. It waits to finish p and q. At end, The program r.c detaches and deletes n semaphores and print the sorted list.
<b>THEORY:</b>	<p><b>WHAT IS PROCESS SYNCHRONIZATION?</b></p> <p>Process Synchronization means coordinating the execution of processes such that no two processes access the same shared resources and data. It is required in a multi-process system where multiple processes run together, and more than one process tries to gain access to the same shared resource or data at the same time.</p> <p>Sections of a Program in OS</p> <p>Following are the four essential sections of a program:</p> <p><b>1. Entry Section:</b> This decides the entry of any process.</p> <p><b>2. Critical Section:</b> This allows a process to enter and modify the shared variable.</p> <p><b>3. Exit Section:</b> This allows the process waiting in the Entry Section, to enter into the Critical Sections and makes sure that the process is removed through this section once it's done executing.</p>



**Computer Engineering Department &**  
**Information Technology Engineering Department**

**Academic Year: 2021-2022**

**Class: S.Y.B.Tech Sem.: 4 Course: OS**

	<p><b>4. Remainder Section:</b> Parts of the Code, not present in the above three sections are collectively called Remainder Section.</p> <p><b>Types of process in Operating System</b> On the basis of synchronization, the following are the two types of processes:</p> <ol style="list-style-type: none"><li>1. Independent Processes: The execution of one process doesn't affect the execution of another.</li><li>2. Cooperative Processes: Execution of one process affects the execution of the other. Thus, it is necessary that these processes are synchronized in order to guarantee the order of execution.</li></ol> <p><b>Critical Section Problem in OS</b> A segment of code that a signal process can access at a particular point of time is known as the critical section. It contains the shared data resources that can be accessed by other processes. Wait() function (represented as P()) handles the entry of a process to the critical section. Whereas signal() function (represented as V()) handles the exit of a process from the critical section. A single process executes in a critical section at a time. Other processes execute after the current process is done executing.</p> <p><b>Rules for Critical Section</b> There are three rules that need to be enforced in the critical section. They are:</p> <ol style="list-style-type: none"><li><b>1. Mutual Exclusion:</b> A special type of binary semaphore used to control access to shared resources. It has a priority inheritance mechanism that helps avoid extended priority inversion problems. Only one process can execute at a time in the critical section.</li><li><b>2. Progress:</b> We use it when the critical section is empty, and a process wants to enter it. The processes that are not present in their reminder section decide who should go in, within a finite time.</li></ol>
--	--



**Computer Engineering Department &**  
**Information Technology Engineering Department**

**Academic Year: 2021-2022**

**Class: S.Y.B.Tech Sem.: 4 Course: OS**

**3. Bound Waiting:** Only a specific number of processes are allowed into their critical section. Thus, a process needs to make a request when it wants to enter the critical section and when the critical section reaches its limit, the system allows the process' request and allows it into its critical section.

Solutions To The Critical Section

Following are some common solutions to the critical section problem:

**1. Peterson Solution:** If a process executes in a critical state, the other process can only execute the rest of the code and vice-versa.

**Example:** Let there be N processes (P1, P2, ... PN) such that at some point of time every process needs to enter the Critical Section. There is a FLAG[] array of size N that is false by default. Therefore, the flag of a process needs to be set true, whenever it wants to enter the critical section.

A variable TURN tells the process number waiting to enter the Critical Section. When a process enters the critical section it changes the TURN to another number from the list of ready processes when it is exiting.

Program:

PROCESS Pi

FLAG[i] = **true**

**while**( (turn != i) AND (CS is !free) ){ wait;  
}

CRITICAL SECTION FLAG[i] = **false**

turn = j; //choose another process

FLAG	
P1	False
P2	True
P3	True
•	
•	
Pn	False



**Computer Engineering Department &**  
**Information Technology Engineering Department**

**Academic Year: 2021-2022**

**Class: S.Y.B.Tech Sem.: 4 Course: OS**

	<p><b>2. Synchronization Hardware:</b> Hardware can also help resolve the problems of critical sections sometimes. Some OS offer lock functionality. This gives a process a lock when it enters the critical section and releases the lock after it leaves a critical section. Due to this other process can't enter a critical section when a process is already inside.</p> <p><b>3. Mutex Locks:</b> Mutex Locks is a strict software method in which a LOCK over critical resources is given to a process in the entry section of code. The process can use this LOCK inside the critical section and can get rid of it in the exit section.</p> <p><b>4. Semaphore Solution:</b> Semaphore is a non-negative variable that is shared between threads. It is a signaling mechanism that uses a thread waiting on a semaphore, to signal another thread. It makes use of wait and signal for process synchronization.</p>
<b>EXPERIMENT 1</b>	
<b>CODE:</b>	<pre>p.c #include &lt;stdio.h&gt; #include &lt;stdlib.h&gt; #include &lt;pthread.h&gt; #include &lt;semaphore.h&gt; #include &lt;time.h&gt; #include &lt;sys/ipc.h&gt; #include &lt;sys/shm.h&gt; #include &lt;stdbool.h&gt; int main() {     key_t key = 1000;     int sh_id = shmget(key, 50 * sizeof(int),         IPC_CREAT   0777);     int *sh = (int *)shmat(sh_id, NULL, 0);     sem_t *p = sem_open("r", 0);     sem_wait(p);     int n, c = 0;     printf("Enter the number of elements:");</pre>



**Computer Engineering Department &**  
**Information Technology Engineering Department**

**Academic Year: 2021-2022**

**Class: S.Y.B.Tech Sem.: 4 Course: OS**

	<pre>scanf("%d", &amp;n); printf("\n"); while (1) {     if (sh[c] == -1)     {         break;     }     else         c++; } for (int i = 0; i &lt; n; i++) {     scanf("%d", &amp;sh[c + i]); } sh[c + n] = -1; sem_post(p); }</pre> <p>q.c</p> <pre>#include &lt;stdio.h&gt; #include &lt;stdlib.h&gt; #include &lt;pthread.h&gt; #include &lt;semaphore.h&gt; #include &lt;time.h&gt; #include &lt;sys/ipc.h&gt; #include &lt;sys/shm.h&gt; #include &lt;stdbool.h&gt; int main() {     int n;     printf("Enter the number of elements:");     scanf("%d", &amp;n);     printf("\n");     key_t key = 1000;     int sh_id = shmget(key, 50 * sizeof(int), IPC_CREAT   0777);</pre>
--	---



**Computer Engineering Department &**  
**Information Technology Engineering Department**

**Academic Year: 2021-2022**

**Class: S.Y.B.Tech Sem.: 4 Course: OS**

	<pre>int *sh = (int *)shmat(sh_id, NULL, 0); sem_t *p = sem_open("r", 0); for (int i = 0; i &lt; n; i++) {     scanf("%d", &amp;sh[i]); } sh[n] = -1; printf("done"); sem_post(p); return 0; }</pre> <p>r.c</p> <pre>#include &lt;stdio.h&gt; #include &lt;semaphore.h&gt; #include &lt;sys/wait.h&gt; #include &lt;sys/shm.h&gt; #include &lt;stdlib.h&gt; #include &lt;string.h&gt; #include &lt;fcntl.h&gt; #include &lt;error.h&gt; #include &lt;unistd.h&gt; #define SHMSZ 27 void swap(int *a, int *b) {     int temp = *a;     *a = *b;     *b = temp; } void sort(int arr[], int n) {     int i, j;     for (i = 0; i &lt; n - 1; i++)     {         for (j = 0; j &lt; n - i - 1; j++)         {             if (arr[j] &gt; arr[j + 1])</pre>
--	---



**Computer Engineering Department &**  
**Information Technology Engineering Department**

**Academic Year: 2021-2022**

**Class: S.Y.B.Tech Sem.: 4 Course: OS**

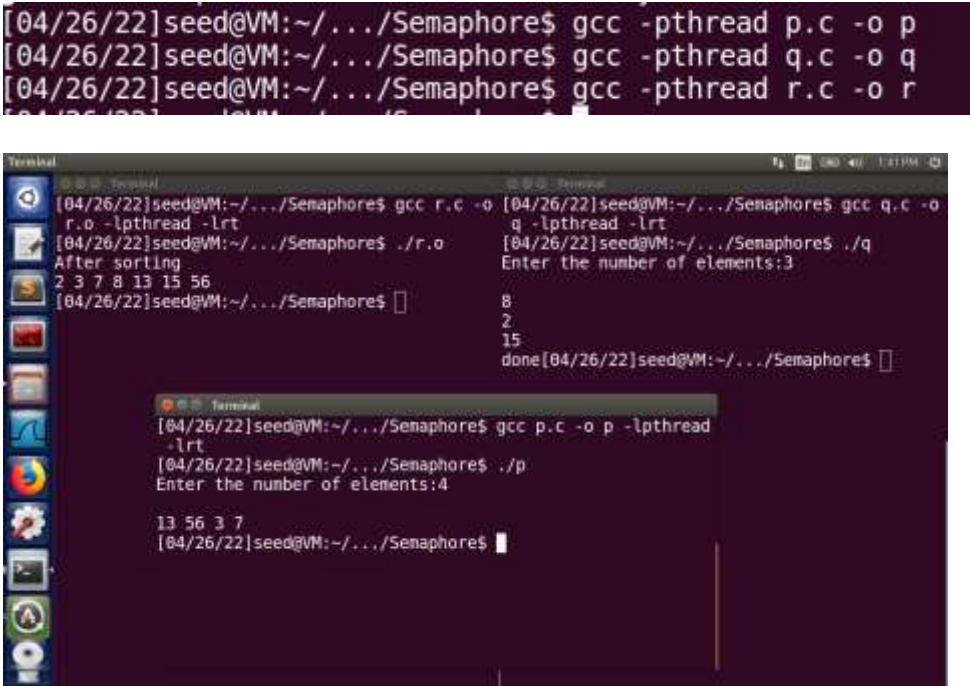
	<pre>swap(&amp;arr[j], &amp;arr[j + 1]);     } } } int main() {     key_t key = 1000;     sem_unlink("r");     sem_t *r = sem_open("r", O_CREAT   O_EXCL, 0660, 0);     if (r == SEM_FAILED)     {         perror("count error");         exit(EXIT_FAILURE);     }     int sh_id = shmget(key, 50 * sizeof(int), IPC_CREAT   0777);     int *sh = (int *)shmat(sh_id, NULL, 0);     sem_wait(r);     sem_post(r);     sleep(2);     sem_wait(r);     int c = 0;     while (1)     {         if (sh[c] == -1)         {             break;         }         else             c++;     }     sort(sh, c);     printf("After sorting\n");     for (int i = 0; i &lt; c; i++)     {         printf("%d ", sh[i]);     } }</pre>
--	---



**Computer Engineering Department &**  
**Information Technology Engineering Department**

**Academic Year: 2021-2022**

**Class: S.Y.B.Tech Sem.: 4 Course: OS**

	<pre>printf("\n"); sem_close(r); }</pre>
<b>OUTPUT TABLE:</b>	 <pre>[04/26/22]seed@VM:~/.../Semaphore\$ gcc -pthread p.c -o p [04/26/22]seed@VM:~/.../Semaphore\$ gcc -pthread q.c -o q [04/26/22]seed@VM:~/.../Semaphore\$ gcc -pthread r.c -o r [04/26/22]seed@VM:~/.../Semaphore\$ gcc r.c -o r [04/26/22]seed@VM:~/.../Semaphore\$ gcc q.c -o q [04/26/22]seed@VM:~/.../Semaphore\$ gcc p.c -o p [04/26/22]seed@VM:~/.../Semaphore\$ ./r.o After sorting 2 3 7 8 13 15 56 [04/26/22]seed@VM:~/.../Semaphore\$ ./q Enter the number of elements:3 8 2 15 done[04/26/22]seed@VM:~/.../Semaphore\$ ./p Enter the number of elements:4 13 56 3 7 [04/26/22]seed@VM:~/.../Semaphore\$</pre>
<b>RESULT:</b> Learnt about process synchronization, also learnt how to implement the different method of process synchronization. Wrote a program in c about process sync which show the implementation of these concepts.	