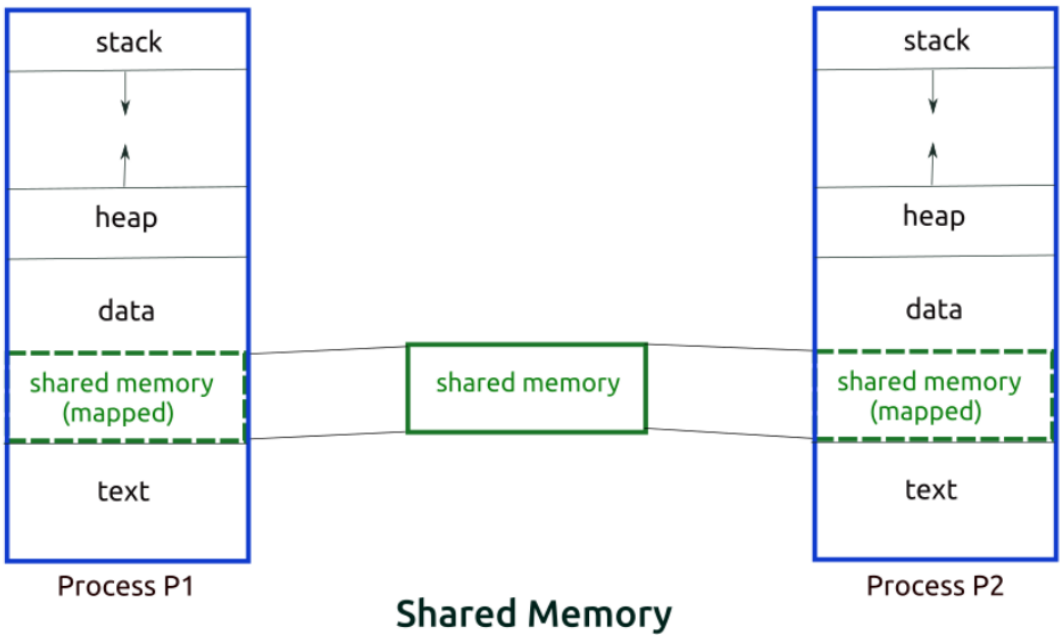
Assignment 8: **Inter-process Communication using Shared Memory using System V. Application to demonstrate: Client and Server Programs in which the server process creates a shared memory segment and writes the message to the shared memory segment. Client process reads the message from the shared memory segment and displays it to the screen.**

Name: **Chaitanya Suresh Uge**
Roll Number: **333064**
GR Number: **21910718**
Division: **C** **Batch:** **C3**

Theory:

Shared Memory:

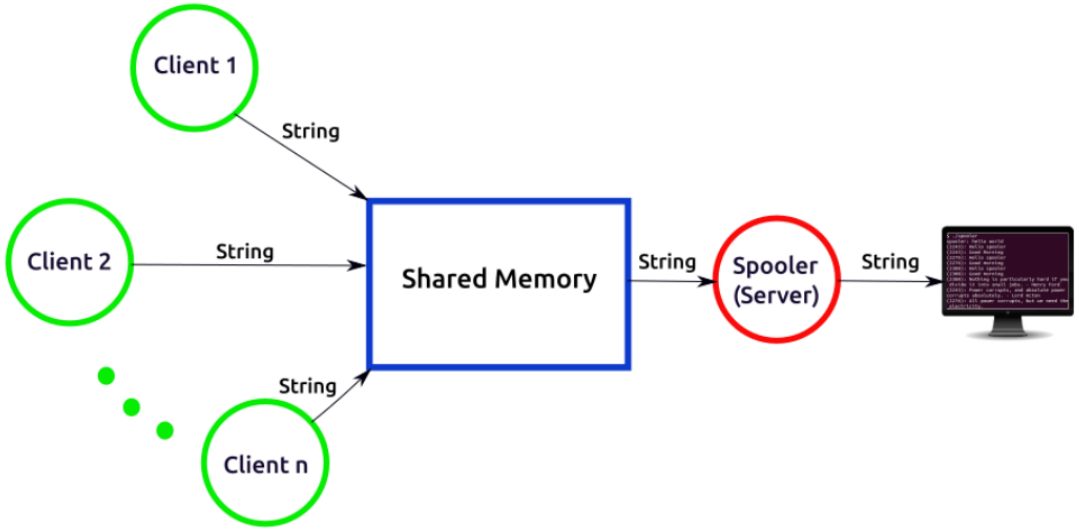
- It's one of the three interprocess communication (IPC) mechanisms available under Linux. (Other two are - Message queues and semaphores)
- A shared memory segment is created by the kernel and mapped to the data segment of the address space of a requesting process.



- In the interprocess communication mechanisms like the pipes, FIFOs and message queues, the work involved in sending data from one process to another is like this.
- Process P1 makes a system call to send data to Process P2.
- The message is copied from the address space of the first process to the kernel space during the system call for sending the message.
- Then, the second process makes a system call to receive the message.
- The message is copied from the kernel space to the address space of the second process.
- The shared memory mechanism does away with this copying overhead. The first process simply writes data into the shared memory segment.
- As soon as it is written, the data becomes available to the second process.
- Shared memory is the fastest mechanism for interprocess communication.

Client-Server communication using System V Shared Memory:

- The example has a server process called server which prints strings received from clients.
- The server is a kind of consumer process which consumes strings.
- The server creates a shared memory segment and attaches it to its address space.
- The client processes attach the shared memory segment and write strings in it. The client processes are producers; they produce strings
- The server takes strings from the shared memory segment and writes the strings on the terminal.
- The server prints strings in the order they are written in the shared memory segment by the clients.
- So, effectively, there are n concurrent processes producing strings at random times and the server prints them nicely in the chronological order. The software architecture is like this.



Software Architecture for Clients and Server using Shared Memory

Source Code:

1. server.c

```
1   /*
2    *
3    *      server.c: Print strings in the shared memory segment
4    *              (Consumer process)
5    */
6
7   #include <stdio.h>
8   #include <stdlib.h>
9   #include <sys/types.h>
10  #include <sys/ipc.h>
11  #include <sys/shm.h>
12  #include <sys/sem.h>
13
14  // Buffer data structures
15  #define MAX_BUFFERS 10
16  #define SHARED_MEMORY_KEY "/tmp/shared_memory_key"
17  #define SEM_MUTEX_KEY "/tmp/sem-mutex-key"
18  #define SEM_BUFFER_COUNT_KEY "/tmp/sem-buffer-count-key"
19  #define SEM_SPOOL_SIGNAL_KEY "/tmp/sem-spool-signal-key"
20  #define PROJECT_ID 'K'
21
22  struct shared_memory {
23     char buf [MAX_BUFFERS] [256];
24     int buffer_index;
```

```
25      int buffer_print_index;
26  };
27
28  void error (char *msg);
29
30  int main (int argc, char **argv)
31  {
32      key_t s_key;
33      union semun
34      {
35          int val;
36          struct semid_ds *buf;
37          ushort array [1];
38      } sem_attr;
39      int shm_id;
40      struct shared_memory *shared_mem_ptr;
41      int mutex_sem, buffer_count_sem, spool_signal_sem;
42
43      printf ("spooler: hello world\n");
44      //  mutual exclusion semaphore
45      /* generate a key for creating semaphore  */
46      if ((s_key = ftok (SEM_MUTEX_KEY, PROJECT_ID)) == -1)
47          error ("ftok");
48      if ((mutex_sem = semget (s_key, 1, 0660 | IPC_CREAT)) == -1)
49          error ("semget");
50      // Giving initial value.
51      sem_attr.val = 0;       // locked, till we finish initialization
52      if (semctl (mutex_sem, 0, SETVAL, sem_attr) == -1)
53          error ("semctl SETVAL");
54
55      // Get shared memory
56      if ((s_key = ftok (SHARED_MEMORY_KEY, PROJECT_ID)) == -1)
57          error ("ftok");
58      if ((shm_id = shmget (s_key, sizeof (struct shared_memory),
59           0660 | IPC_CREAT)) == -1)
60          error ("shmget");
61      if ((shared_mem_ptr = (struct shared_memory *) shmat (shm_id, NULL, 0))
62           == (struct shared_memory *) -1)
63          error ("shmat");
64      // Initialize the shared memory
65      shared_mem_ptr -> buffer_index = shared_mem_ptr -> buffer_print_index =
66  0;
67
68      // counting semaphore, indicating the number of available buffers.
69      /* generate a key for creating semaphore  */
70      if ((s_key = ftok (SEM_BUFFER_COUNT_KEY, PROJECT_ID)) == -1)
71          error ("ftok");
72      if ((buffer_count_sem = semget (s_key, 1, 0660 | IPC_CREAT)) == -1)
73          error ("semget");
74      // giving initial values
75      sem_attr.val = MAX_BUFFERS;    // MAX_BUFFERS are available
76      if (semctl (buffer_count_sem, 0, SETVAL, sem_attr) == -1)
```

```
77          error ("semctl SETVAL");
78
79       // counting semaphore, indicating the number of strings to be printed.
80       /* generate a key for creating semaphore  */
81       if ((s_key = ftok (SEM_SPOOL_SIGNAL_KEY, PROJECT_ID)) == -1)
82          error ("ftok");
83       if ((spool_signal_sem = semget (s_key, 1, 0660 | IPC_CREAT)) == -1)
84          error ("semget");
85       // giving initial values
86       sem_attr.val = 0;    // 0 strings are available initially.
87       if (semctl (spool_signal_sem, 0, SETVAL, sem_attr) == -1)
88          error ("semctl SETVAL");
89
90       // Initialization complete; now we can set mutex semaphore as 1 to
91       // indicate shared memory segment is available
92       sem_attr.val = 1;
93       if (semctl (mutex_sem, 0, SETVAL, sem_attr) == -1)
94          error ("semctl SETVAL");
95
96       struct sembuf asem [1];
97
98       asem [0].sem_num = 0;
99       asem [0].sem_op = 0;
100      asem [0].sem_flg = 0;
101
102      while (1) {  // forever
103         // Is there a string to print? P (spool_signal_sem);
104         asem [0].sem_op = -1;
105         if (semop (spool_signal_sem, asem, 1) == -1)
106               perror ("semop: spool_signal_sem");
107
108         printf ("%s", shared_mem_ptr -> buf [shared_mem_ptr ->
109   buffer_print_index]);
110
111         /* Since there is only one process (the spooler) using the
112           buffer_print_index, mutex semaphore is not necessary */
113         (shared_mem_ptr -> buffer_print_index)++;
114         if (shared_mem_ptr -> buffer_print_index == MAX_BUFFERS)
115           shared_mem_ptr -> buffer_print_index = 0;
116
117         /* Contents of one buffer has been printed.
118           One more buffer is available for use by producers.
119           Release buffer: V (buffer_count_sem);  */
120         asem [0].sem_op = 1;
121         if (semop (buffer_count_sem, asem, 1) == -1)
122               perror ("semop: buffer_count_sem");
123      }
124   }
125
126   // Print system error and exit
127   void error (char *msg)
128   {
```

```
129     perror (msg);
        exit (1);
    }
```

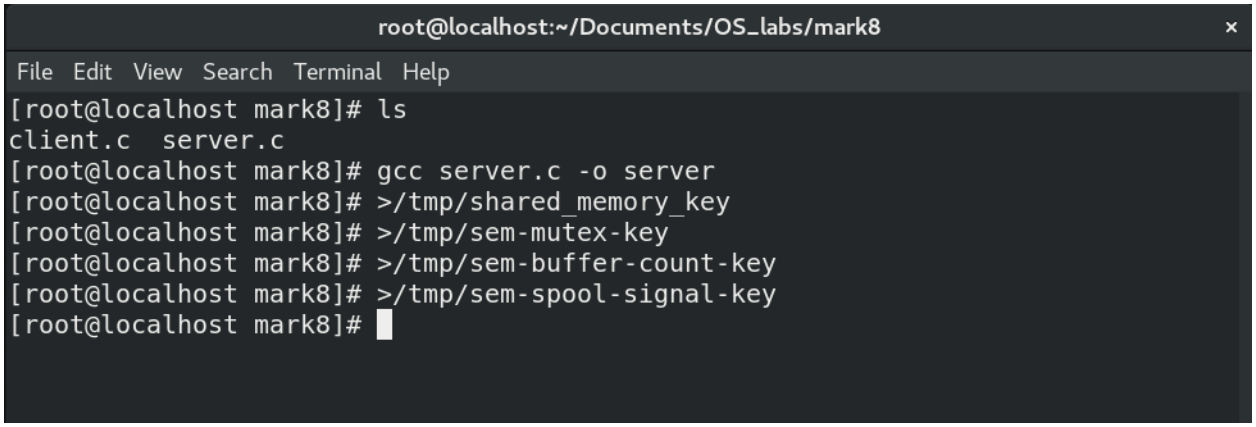
2. client.c

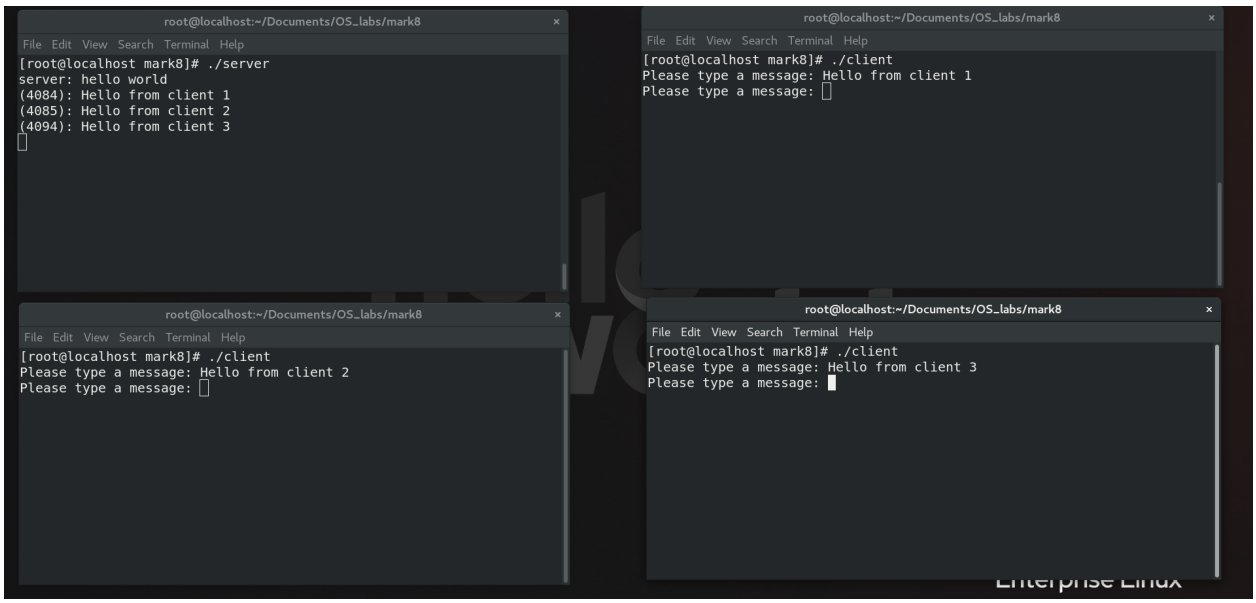
```
1   /*
2    *
3    *      client.c: Write strings for printing in the shared memory segment
4    *              (Producer process)
5    */
6
7   #include <stdio.h>
8   #include <stdlib.h>
9   #include <string.h>
10  #include <sys/types.h>
11  #include <unistd.h>
12  #include <sys/ipc.h>
13  #include <sys/shm.h>
14  #include <sys/sem.h>
15
16  // Buffer data structures
17  #define MAX_BUFFERS 10
18  #define SHARED_MEMORY_KEY "/tmp/shared_memory_key"
19  #define SEM_MUTEX_KEY "/tmp/sem-mutex-key"
20  #define SEM_BUFFER_COUNT_KEY "/tmp/sem-buffer-count-key"
21  #define SEM_SPOOL_SIGNAL_KEY "/tmp/sem-spool-signal-key"
22  #define PROJECT_ID 'K'
23
24  struct shared_memory {
25     char buf [MAX_BUFFERS] [256];
26     int buffer_index;
27     int buffer_print_index;
28  };
29
30  void error (char *msg);
31
32  int main (int argc, char **argv)
33  {
34     key_t s_key;
35     union semun
36     {
37        int val;
38        struct semid_ds *buf;
39        ushort array [1];
40     } sem_attr;
41     int shm_id;
```

```
42      struct shared_memory *shared_mem_ptr;
43      int mutex_sem, buffer_count_sem, spool_signal_sem;
44
45      //  mutual exclusion semaphore
46      /* generate a key for creating semaphore  */
47      if ((s_key = ftok (SEM_MUTEX_KEY, PROJECT_ID)) == -1)
48         error ("ftok");
49      if ((mutex_sem = semget (s_key, 1, 0)) == -1)
50         error ("semget");
51
52      // Get shared memory
53      if ((s_key = ftok (SHARED_MEMORY_KEY, PROJECT_ID)) == -1)
54         error ("ftok");
55      if ((shm_id = shmget (s_key, sizeof (struct shared_memory), 0)) == -1)
56         error ("shmget");
57      if ((shared_mem_ptr = (struct shared_memory *) shmat (shm_id, NULL, 0))
58          == (struct shared_memory *) -1)
59         error ("shmat");
60
61      // counting semaphore, indicating the number of available buffers.
62      /* generate a key for creating semaphore  */
63      if ((s_key = ftok (SEM_BUFFER_COUNT_KEY, PROJECT_ID)) == -1)
64         error ("ftok");
65      if ((buffer_count_sem = semget (s_key, 1, 0)) == -1)
66         error ("semget");
67
68      // counting semaphore, indicating the number of strings to be printed.
69      /* generate a key for creating semaphore  */
70      if ((s_key = ftok (SEM_SPOOL_SIGNAL_KEY, PROJECT_ID)) == -1)
71         error ("ftok");
72      if ((spool_signal_sem = semget (s_key, 1, 0)) == -1)
73         error ("semget");
74
75      struct sembuf asem [1];
76
77      asem [0].sem_num = 0;
78      asem [0].sem_op = 0;
79      asem [0].sem_flg = 0;
80
81      char buf [200];
82
83      printf ("Please type a message: ");
84
85      while (fgets (buf, 198, stdin)) {
86         // remove newline from string
87         int length = strlen (buf);
88         if (buf [length - 1] == '\n')
89            buf [length - 1] = '\0';
90
91         // get a buffer: P (buffer_count_sem);
92         asem [0].sem_op = -1;
93         if (semop (buffer_count_sem, asem, 1) == -1)
```

```
94              error ("semop: buffer_count_sem");
95
96          /* There might be multiple producers. We must ensure that
97             only one producer uses buffer_index at a time.  */
98          // P (mutex_sem);
99          asem [0].sem_op = -1;
100         if (semop (mutex_sem, asem, 1) == -1)
101             error ("semop: mutex_sem");
102
103             // Critical section
104           sprintf (shared_mem_ptr -> buf [shared_mem_ptr -> buffer_index],
105 "(%d): %s\n", getpid (), buf);
106           (shared_mem_ptr -> buffer_index)++;
107           if (shared_mem_ptr -> buffer_index == MAX_BUFFERS)
108              shared_mem_ptr -> buffer_index = 0;
109
110         // Release mutex sem: V (mutex_sem)
111         asem [0].sem_op = 1;
112         if (semop (mutex_sem, asem, 1) == -1)
113             error ("semop: mutex_sem");
114
115           // Tell spooler that there is a string to print: V (spool_signal_sem);
116         asem [0].sem_op = 1;
117         if (semop (spool_signal_sem, asem, 1) == -1)
118             error ("semop: spool_signal_sem");
119
120         printf ("Please type a message: ");
121     }
122
123     if (shmdt ((void *) shared_mem_ptr) == -1)
124         error ("shmdt");
125     exit (0);
126 }
127
128 // Print system error and exit
129 void error (char *msg)
130 {
131     perror (msg);
132     exit (1);
    }
```

OUTPUT:


root@localhost:~/Documents/OS_labs/mark8
File Edit View Search Terminal Help
[root@localhost mark8]# ls
client.c server.c
[root@localhost mark8]# gcc server.c -o server
[root@localhost mark8]# >/tmp/shared_memory_key
[root@localhost mark8]# >/tmp/sem-mutex-key
[root@localhost mark8]# >/tmp/sem-buffer-count-key
[root@localhost mark8]# >/tmp/sem-spool-signal-key
[root@localhost mark8]#


root@localhost:~/Documents/OS_labs/mark8
File Edit View Search Terminal Help
[root@localhost mark8]# ./server
server: hello world
(4084): Hello from client 1
(4085): Hello from client 2
(4094): Hello from client 3
root@localhost:~/Documents/OS_labs/mark8
File Edit View Search Terminal Help
[root@localhost mark8]# ./client
Please type a message: Hello from client 1
Please type a message:
root@localhost:~/Documents/OS_labs/mark8
File Edit View Search Terminal Help
[root@localhost mark8]# ./client
Please type a message: Hello from client 2
Please type a message:
root@localhost:~/Documents/OS_labs/mark8
File Edit View Search Terminal Help
[root@localhost mark8]# ./client
Please type a message: Hello from client 3
Please type a message: