IPC- Shared Memory

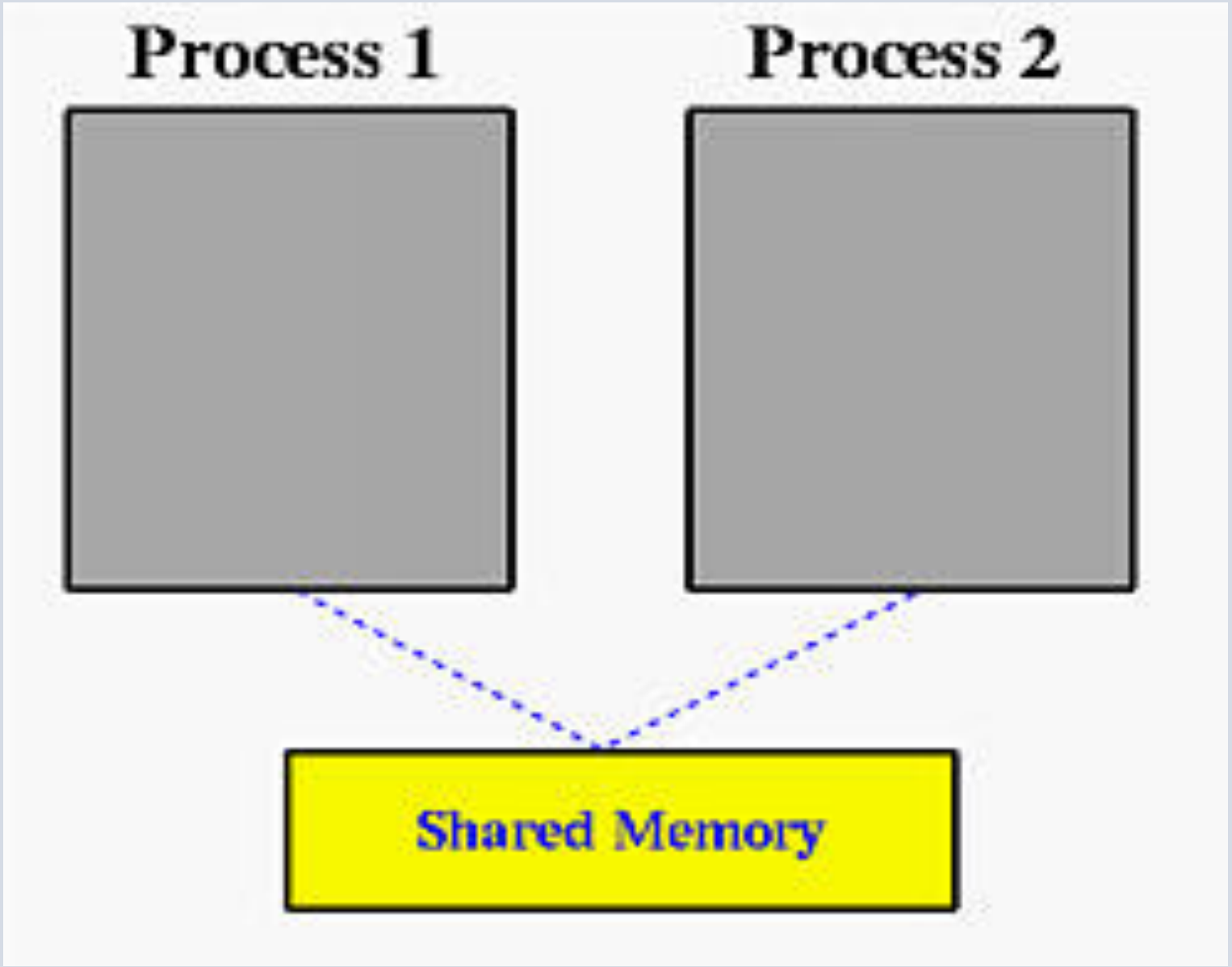
Sandy Joseph

Assistant Professor

Department of CSE, RSET

Shared Memory

- Shared Memory region resides in the address space of the process creating the shared memory segment.
- Processes which wish to communicate should attach this shared memory with their address space.
- Normally OS will prevent one process from accessing another process's memory.
- Shared memory requires that two or more processes remove this restriction.
- They can exchange information by reading and writing in this area.
- They also need to ensure that they are not writing to this location simultaneously.



Steps in IPC using shared memory

- Create a unique key for IPC
- Create shared memory
- Attach the shared memory with the program.
- Write data to the shared memory.
- Detach the shared memory
- Destroy the shared memory

System calls for shared memory

- ***Step 1: Create key***
- ***ftok()****: is use to generate a unique key which represents that IPC.*
- This generates an IPC key on the basis of the supplied filename and ID. The filename can be provided along with its complete path. The filename must refer to an existing file.
- **Syntax:-**
- key_t ftok(const char *filename, int id);
- ftok function will generate the same key value with the same filename(same path) and the same id which are supplied as arguments.
- Upon successful completion it will return the key.
- Upon failure , it return -1.

- ***Step2 Create shared memory***
- ***shmget()****: int shmget(key_t k , size_t size, int shmflg);*
  - Where k is the key for the shared memory segment
  - **k** is of type **key_t** or **IPC_PRIVATE.// IPC_PRIVATE** is for parent and child processes, otherwise use **key_t**
  - size is the size of the shared memory segment
  - shmflag is the create or use flag
  - Two values are important:
    - **IPC_CREAT | 0666** for a server (*i.e.*, creating ,read and write access to the server)
    - **0666** for any client (*i.e.*, read and write access to the client)
  - *upon successful completion, shmget() returns an identifier for the shared memory segment.*

- ***Step 3: Attach shared memory***
- ***shmat()****: Before program can use a shared memory segment, program has to attach itself to it using shmat().*
- *void *shmat(int shmid ,void *shmaddr ,int shmflg);*
  - *shmid is shared memory id.*
  - *shmaddr specifies specific address to use but we can set it to zero and OS will automatically choose the address.*
  - If **SHM_RDONLY** is specified in *shmflg*, the segment is attached for reading and the process must have read permission for the segment.
  - If it is 0, the segment is attached for read and write and the process must have read and write permission for the segment.

- ***Step 4: Write data to shared memory***
- ***Step 5: Detach shared memory***
- ***Step 6 shmdt()****: When you're done with the shared memory segment, your program should detach itself from it using shmdt().*
  - *Syntax*
    - *int shmdt(void *shmaddr);*
    - *Eg:- shmdt(str)*
    - *Where str is the shared memory.*

- ***Step 6:- Destroy shared memory***
- ***shmctl()****: when you detach from shared memory, it is not destroyed. So, to destroy shmctl() should be used.*
- ***Syntax***
  - int shmctl(int *shmid*, int *cmd*, struct shmid_ds **buf*);
  - *shmid- ID of the shared memory*
  - *cmd – is the shared memory control operation*
    - *It takes value IPC_RMID for destroying the shared memory created.*
    - The *buf* argument is a pointer to a *shmid_ds* structure, defined in <sys/shm.h> as follows:

```
struct shmid_ds
 {
struct ipc_perm  shm_perm;  /* Ownership and permissions */
size_t  shm_segsz;      /* Size of segment (bytes) */
time_t  shm_atime;      /* Last attach time */
time_t  shm_dtime;      /* Last detach time */
time_t  shm_ctime;      /* Last change time */
pid_t  shm_cpid;        /* PID of creator */
pid_t shm_lpid;         /* PID of last shmat()/shmdt() */
shmatt_t  shm_nattch;   /* No. of current attaches */

...
};
```

Writer program writes data to the shared memory

```
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
int main()
{       // ftok to generate unique key
        key_t key = ftok("shmfile",65);
        // shmget returns an identifier in shmid
        int shmid = shmget(key, 1024, 0666|IPC_CREAT);
        // shmat to attach to shared memory
        char *str = (char*) shmat(shmid,(void*)0,0);
```

```
        printf(“Enter data“);
        gets(str);
        printf("Data written in shared memory: %s\n",str);
        //detach from shared memory
        shmdt(str);
        return 0;
}
```

Reader reads data written by the writer program from shared memory


```
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
int main()
{       // ftok to generate unique key
        key_t key = ftok("shmfile",65);
        // shmget returns an identifier in shmid
        int shmid = shmget(key,1024,0666|IPC_CREAT);
        // shmat to attach to shared memory
        char *str = (char*) shmat(shmid,(void*)0,0);
```

```
    printf("Data read from memory: %s\n",str);
    //detach from shared memory
    shmdt(str);
    // destroy the shared memory
    shmctl(shmid, IPC_RMID, NULL);
    return 0;
}
```

Points:-

- Both programs can function as reader and writer.
- We run both programs simultaneously in two different terminals.
- Give delay in reader so that writer gets enough time to write to shared memory.
- Since writer creates the shared memory, create statement in reader will not execute.

Shared memory Program to be done in OS lab

Exp 1) Write a shared memory program, where there are two programs

a) Writer – which finds the n!, r! and (n-r)! value of a given n and r and writes to the shared memory.

b) Reader – which reads the values from the shared memory and calculate the nCr value and prints it.

Exp 2) Write a shared memory program where there are two programs a) *writer* b) *reader* . Implement IPC where the *writer* writes a string into shared memory and *reader* reads and counts the number of vowels, symbols and numbers in it.

Exp 3) Write a shared memory program where writer writes a number to the shared memory, reader reads it from shared memory and check whether its palindrome. Reader also prints the reverse.

Topics beyond syllabus

RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY (AUTONOMOUS)

Integer array in shared memory

```
array = (int *)shmat(shmid, 0, 0);
array = malloc(sizeof(int)*count);
```

It is possible to declare **structure** in shared memory

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <string.h>
 typedef struct People
 {
   int age;
 } Person;
```

```
int main()
 {
 Person arun;
 Person *p_arun;
 int id;
 int key = ftok("Rajagiri", 67);
 p_arun = &arun;
id = shmget( key, sizeof(arun), IPC_CREAT | 0666);
p_arun = shmat(id, NULL, 0);
 (*p_arun).age = 21;
printf("%d", (*p_arun).age);
return 0;  }
```

RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY (AUTONOMOUS)

RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY
(AUTONOMOUS)