Practical 7 Guidelines

Prepared By: Khushboo Patel

Definition

Write the following programs using inter process communication – shared memory.

The program ‘writer.c’ will print 1 to 100 in shared memory region.

Another program ‘reader.c’ that will read all the numbers from shared memory to make addition of it and display it.

Interprocess Communication

A process can be of two types:

1. **Independent process:**

   An independent process is not affected by the execution of other processes

1. **Co-operating process**

   A co-operating process can be affected by other executing processes.

What is Interprocess Communication

- Inter process communication (IPC) is a mechanism which allows processes to communicate each other and synchronize their actions.
- The communication between these processes can be seen as a method of co-operation between them.
- Processes can communicate with each other using these two ways:

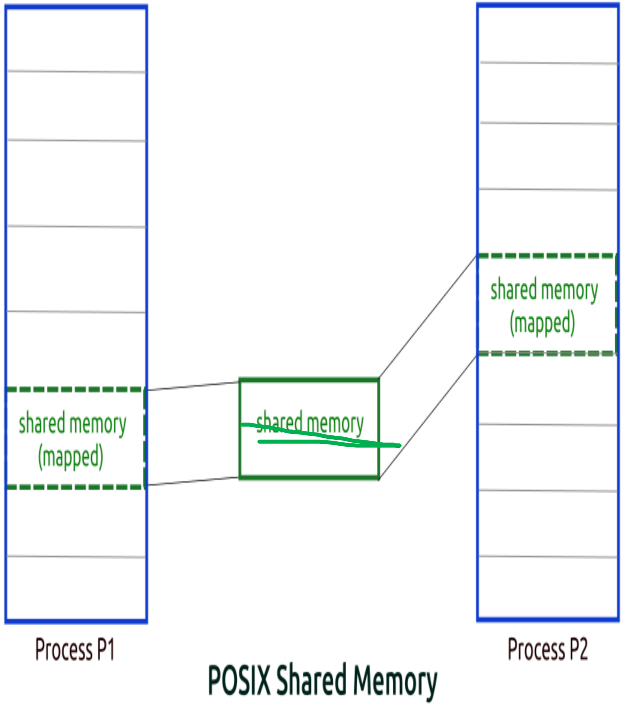
1. Shared memory →
2. Message Passing → msg queues

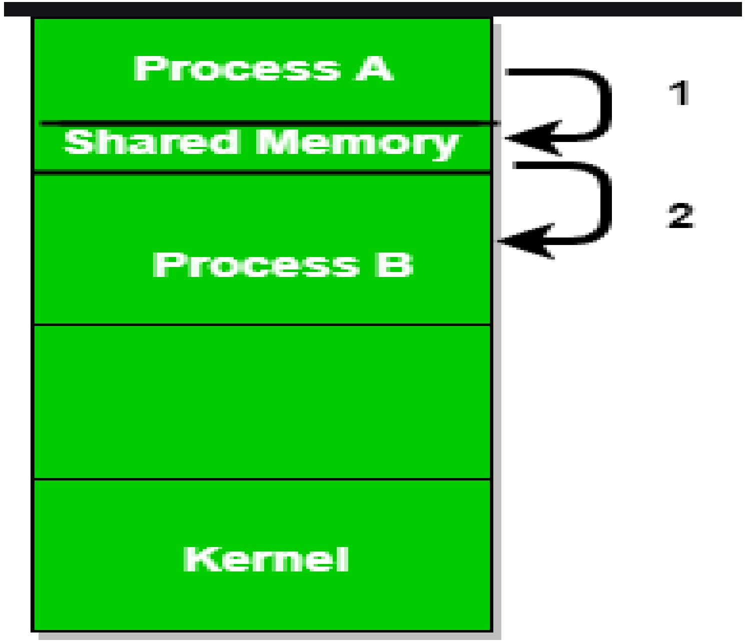
What is Shared Memory?

P1 P2

- The parent and child processes are run in *separate* address spaces.
- A *shared memory segment* is a piece of memory that can be ① allocated and ② attached to an address space. Thus, processes that have this memory segment attached will have access to it.
- But, *race conditions can occur*

Shared Memory problem





Process for using Shared Memory

=123 P1 P2

- ✓ Find a *key*. Unix uses this key for identifying shared memory segments.
- ✓ Use `shmget()` to allocate a shared memory.
- ✓ Use `shmat()` to attach a shared memory to an address space.
- Use `shmdt()` to detach a shared memory from an address space.
- Use `shmctl()` to deallocate a shared memory

Step 1: Generate a key

- Unix requires a ***key*** of type **key_t** defined in file **sys/types.h** for requesting resources such as shared memory segments, message queues and semaphores.
- A key is simply an integer of type **key_t**; however, you should not use **int** or **long**, since the length of a key is system dependent.
- Keys are global entities. If other processes know your key, they can access your shared memory.

Step 1: Generate a ‘Key’ (Continue..)

- **To use shared memory, include the following**

```
#include<sys/types.h> ✓

#include <sys/ipc.h> ✓

#include <sys/shm.h> ✓
```

- **There are 3 methods to generate a key.**

Method 1: generate a key (Do it yourself)

```
key_t  SomeKey;
SomeKey = 1234;
```

Method 2: generate a key (Use ftok())

```
key_t = ftok(char *path, int ID);
```

- `path` **is a path name (*e.g.*,** `“./”`**)**
- `ID` **is an integer (*e.g.*,** `‘a’`**)**
- **Function** `ftok()` **returns a key of type** `key_t`**:**

  ```
  SomeKey = ftok(“./”, ‘x’);
  ```

- Upon successful completion, ftok() shall return a key. Otherwise, ftok() shall return -1 and set errno to indicate the error.

Method 3: generate a key (Ask system to provide key)

- **Ask the system to provide a private key using IPC_PRIVATE**
- used with shmget()

Step 2: Ask for Shared Memory (Use shmget() to request a shared memory)

```
shm_id = shmget(
    key_t   key,       /* identity key */
    int     size,      /* memory size */
    int     flag);     /* creation or use */
```

- shmget()returns a shared memory ID.
- The flag, for our purpose, is either 0666 (rw) or IPC_CREAT | 0666.

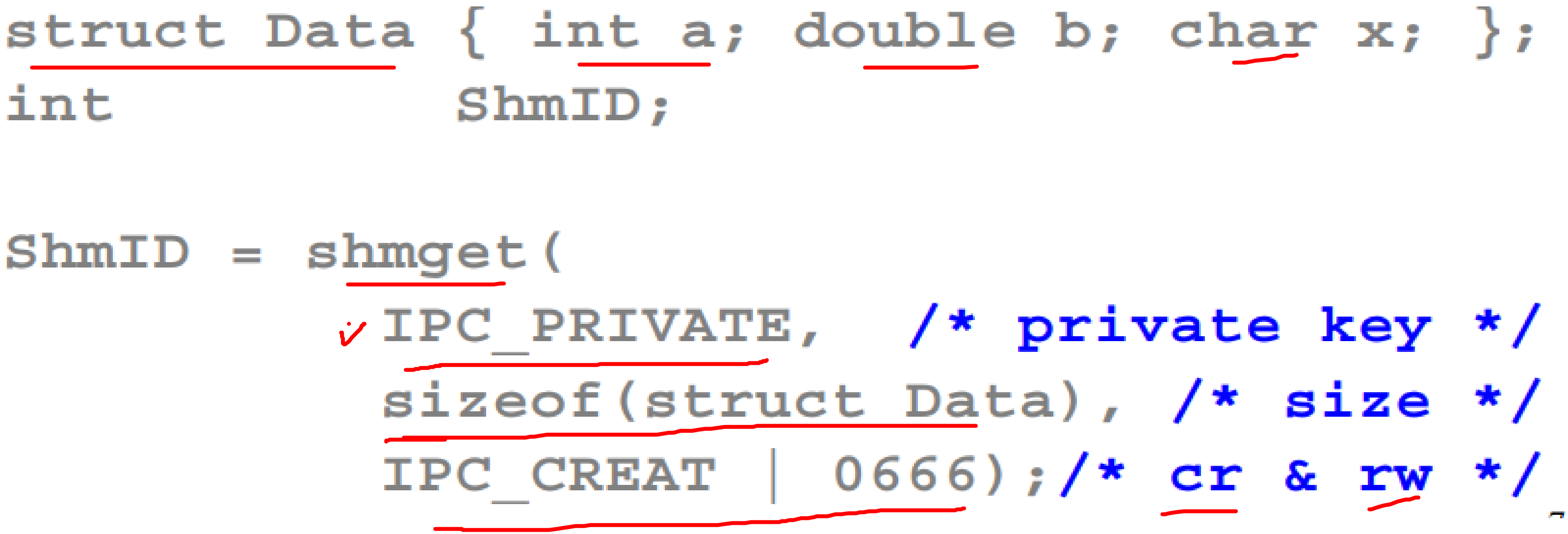
r - 4
w - 2
x - 1

u g o

- **IPC_CREAT | 0666** for a server (*i.e.*, creating and granting read and write access to the server)
- **0666** for any client (*i.e.*, granting read and write access to the client)

Step 2 (Continue..)

- The following creates a shared memory of size struct Data with a private key IPC_PRIVATE. This is a creation (IPC_CREAT) and permits read and write (0666).



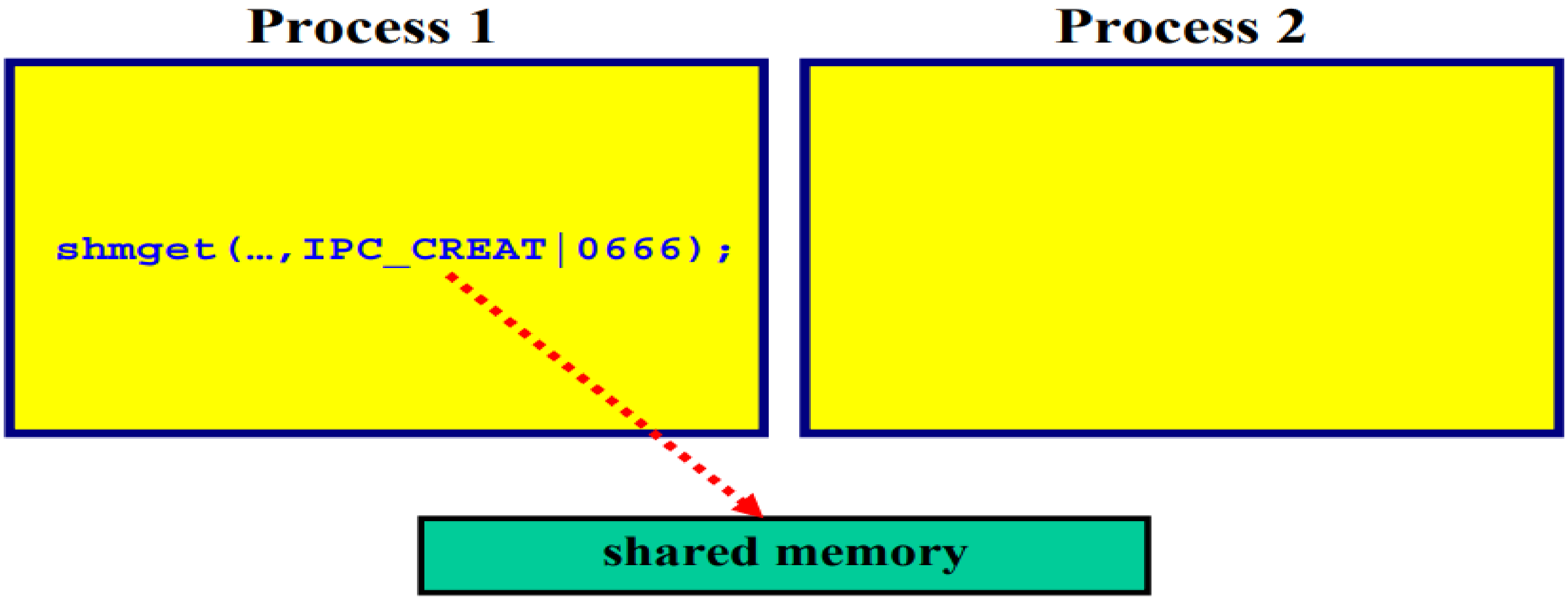
```
struct Data { int a; double b; char x; };
int         ShmID;

ShmID = shmget(
          IPC_PRIVATE,  /* private key */
          sizeof(struct Data), /* size */
          IPC_CREAT | 0666);/* cr & rw */
```

Homework:

Check out other shmflg values

For ex, IPC_EXCL

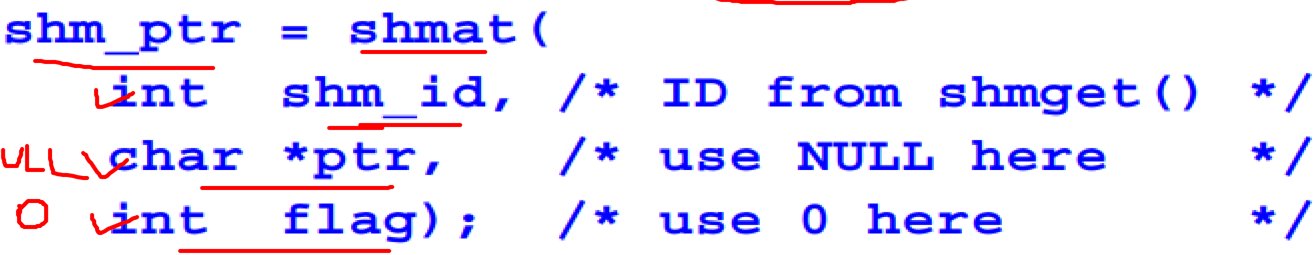
After Execution of shmget()



Shared memory is allocated; but, is not part of the address space

Step 3: Attach Shared Memory

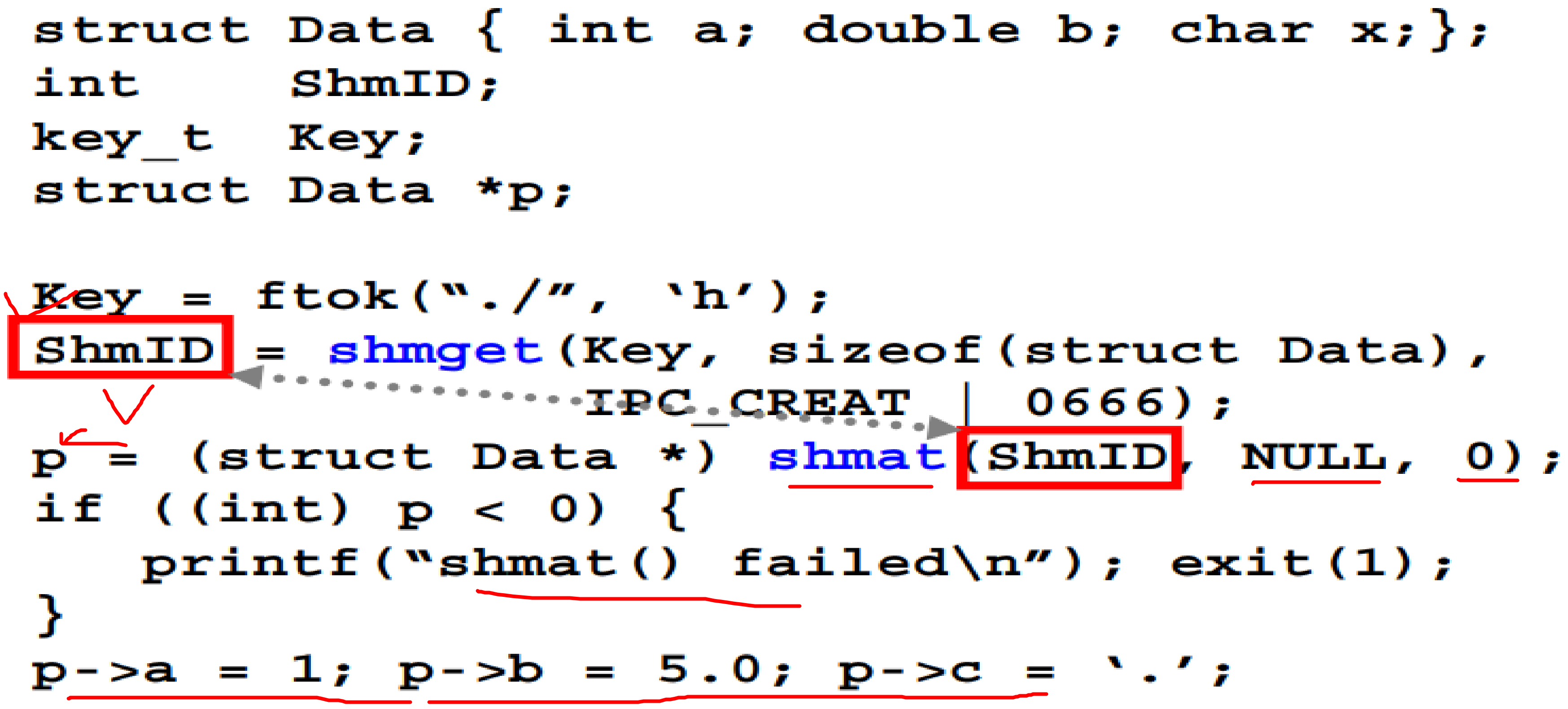
- Use shmat() to attach an existing shared memory to an address space.



```
shm_ptr = shmat(
         int   shm_id, /* ID from shmget() */
    NULL char  *ptr,   /* use NULL here    */
    0    int   flag);  /* use 0 here       */
```

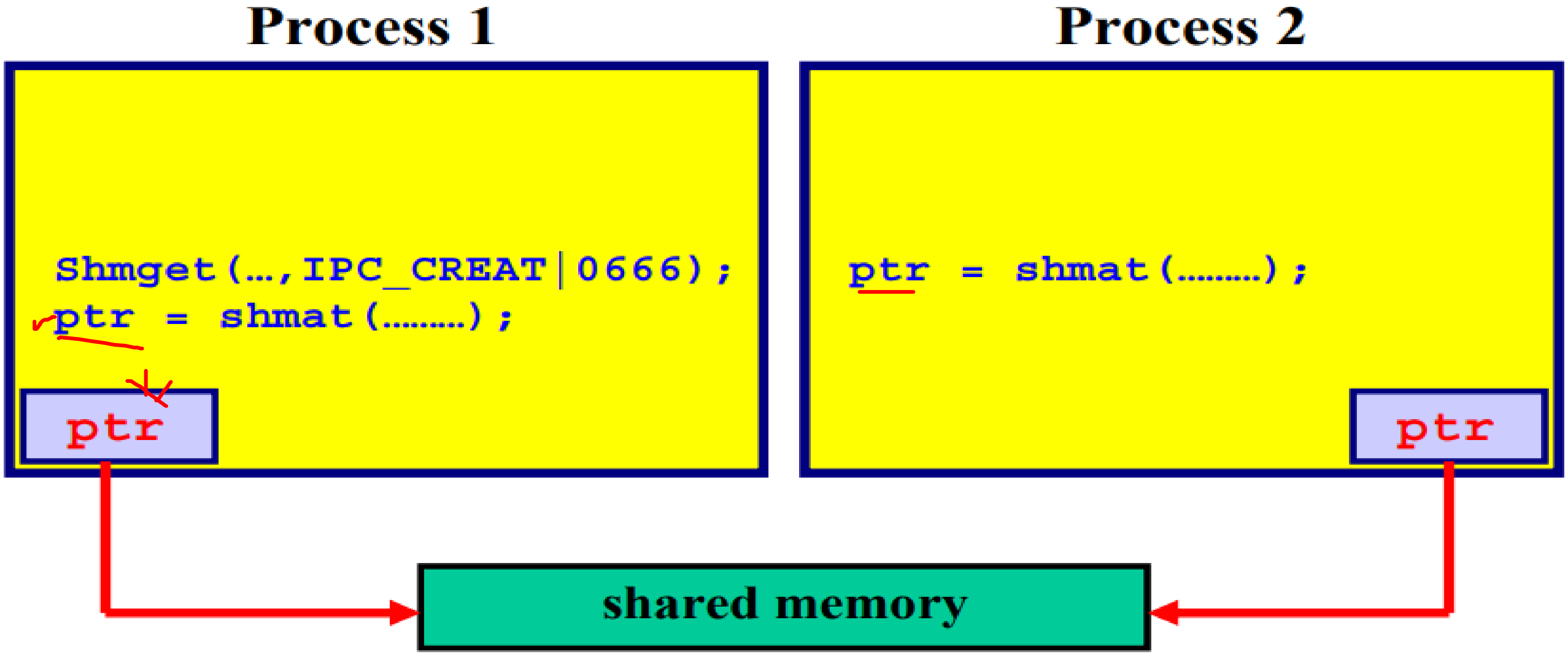
- shm_id is the shared memory ID returned by shmget() .
- Use NULL and 0 for the second and third arguments, respectively.
- shmat() returns a void pointer to the memory. If unsuccessful, it returns a negative integer.
- If you provide second argument NULL then it will take address space automatically.
- If the flag is **SHM_RDONLY**, this shared memory is attached as a read-only memory; otherwise, it is readable and writable.

Step 3: (Continue..)



```
struct Data { int a; double b; char x;};
int      ShmID;
key_t    Key;
struct Data *p;

Key = ftok(“./”, ‘h’);
ShmID = shmget(Key, sizeof(struct Data),
               IPC_CREAT | 0666);
p = (struct Data *) shmat(ShmID, NULL, 0);
if ((int) p < 0) {
    printf(“shmat() failed\n”); exit(1);
}
p->a = 1; p->b = 5.0; p->c = ‘.’;
```

Step 3 (Continue..)



Process 1

```
Shmget(…,IPC_CREAT|0666);
ptr = shmat(………);
```

Process 2

```
ptr = shmat(………);
```

Now processes can access the shared memory

Step 4 : Detaching/Removing Shared Memory

- To detach a shared memory, use

  shmdt(shm_ptr);

- shm_ptr is the pointer returned by shmat() .
- After a shared memory is detached, it is still there. You can re-attach and use it again.
- To remove a shared memory, use

  shmctl(shm_ID, IPC_RMID, NULL);

- shm_ID is the shared memory ID returned by shmget().
- After a shared memory is removed, it no longer exists.

Reason for 0666

If a client wants to use a shared memory created with **IPC_PRIVATE**, it must be a child process of the server, created ***after*** the parent has obtained the shared memory, so that the private key value can be passed to the child when it is created. For a client, changing **IPC_CREAT | 0666** to **0666** works fine. **A warning to novice C programmers:** don't change **0666** to **666**. The leading **0** of an integer indicates that the integer is an octal number. Thus, **0666** is 110110110 in binary. If the leading zero is removed, the integer becomes six hundred sixty six with a binary representation 1111011010.