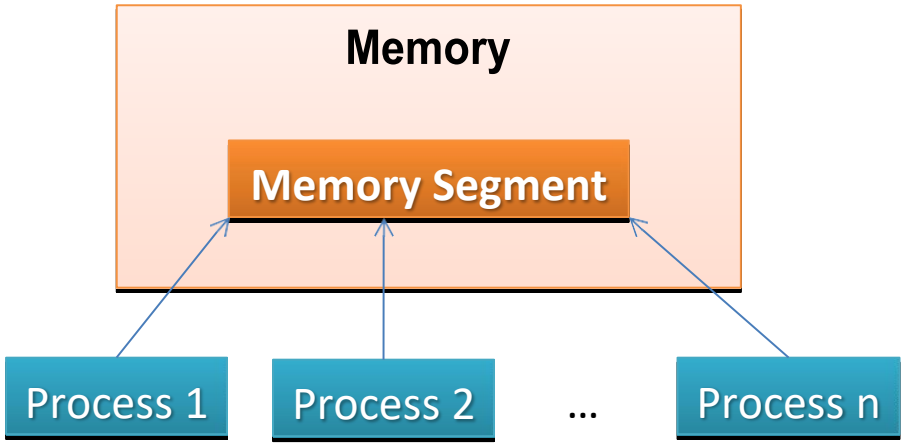
Chapitre V SHARED MEMORY

Shared Memory Segment

- What is shared memory?
  - Shared memory (SHM) is one method of inter-process communication (IPC) whereby 2 or more processes share a single chunk of memory to communicate.


Memory
Memory Segment
Process 1
Process 2
...
Process n

Steps of Shared Memory IPC

1. Creating the segment and connecting
2. Getting a pointer to the segment
3. Reading and Writing
4. Detaching from and deleting segments

1. Creating the segment and connecting

- System V IPC is used in these examples
- A shared memory segment is ‘created’ and ‘connected to’ via the **shmget()** call

```
int shmget(key_t key, size_t size, int  shmflg);
```

- The *key* argument should be created using **ftok()**.
- The *size*, is the size in bytes of the shared memory segment.
- The *shmflg* should be set to the permissions of the segment bitwise-ORd with IPC_CREAT if you want to create the segment, but can be 0 otherwise.
- Upon successful completion, **shmget()** returns an identifier for the shared memory segment.

1. Creating the segment and connecting Cont.

- Here's an example call that creates a 1K segment with 644 permissions (rw-r--r--)

```
key_t key;
int shmid;

key = ftok("/home/beej/somefile3", 'R');
shmid = shmget(key, 1024, 0644 | IPC_CREAT);
```

2. Getting a pointer to the segment

- The shared memory segment must be attached using **shmat()** before its used.

```
void *shmat(int shmid, void *shmaddr, int  shmflg);
```

- *shmid* is the shared memory ID from the **shmget()** call.
- *shmaddr*, which you can use to tell **shmat()** which specific address to use. When set it to 0, the OS will decide the address.
- *shmflg* can be set to SHM_RDONLY if you only want to read from it, 0 otherwise.

2. Getting a pointer to the segment Cont.

- Here's a more complete example of how to get a pointer to a shared memory segment:

```
key_t key;
int shmid;
char *data;

key = ftok("/home/beej/somefile3", 'R');
shmid = shmget(key, 1024, 0644 | IPC_CREAT);
data = shmat(shmid, (void *)0, 0);
```

3. Reading and Writing

- The *data* pointer from the above example is a char pointer. Thus it reads chars from it.
- lets say the 1K shared memory segment contains a null-terminated string.
- It can be printed like this:

```
printf("shared contents: %s\n", data);
```

- And we could store something in it as easily as this:

```
printf("Enter a string: ");
gets(data);
```

4. Detaching from and deleting segments

- When you're done with the shared memory segment, your program should detach itself from it using the **shmdt()** call:

```
int shmdt(void *shmaddr);
```

  - The only argument, *shmaddr*, is the address you got from **shmat()**.
  - The function returns -1 on error, 0 on success.

4. Detaching from and deleting segments Cont.

- **Remember!** When you detach from the segment, it isn't destroyed. Nor is it removed when *everyone* detaches from it.

- You have to specifically destroy it using a call to **shmctl()**

```
shmctl(shmid, IPC_RMID, NULL);
```

- The above call deletes the shared memory segment, assuming no one else is attached to it.

Code Example

Or else,

- As always, you can destroy the shared memory segment from the command line using the **ipcrm** Unix command.

```
ipcrm [–m shmid]
```

- Also, be sure that you don't leave any unused shared memory segments sitting around wasting system resources.

- All the System V IPC objects you own can be viewed using the **ipcs** command.

Shared Memory, Pros and Cons

- Pros
  - Fast bidirectional communication among any number of processes
  - Saves Resources

- Cons
  - Needs concurrency control (leads to data inconsistencies like ‘Lost update’)
  - Lack of data protection from Operating System (OS)