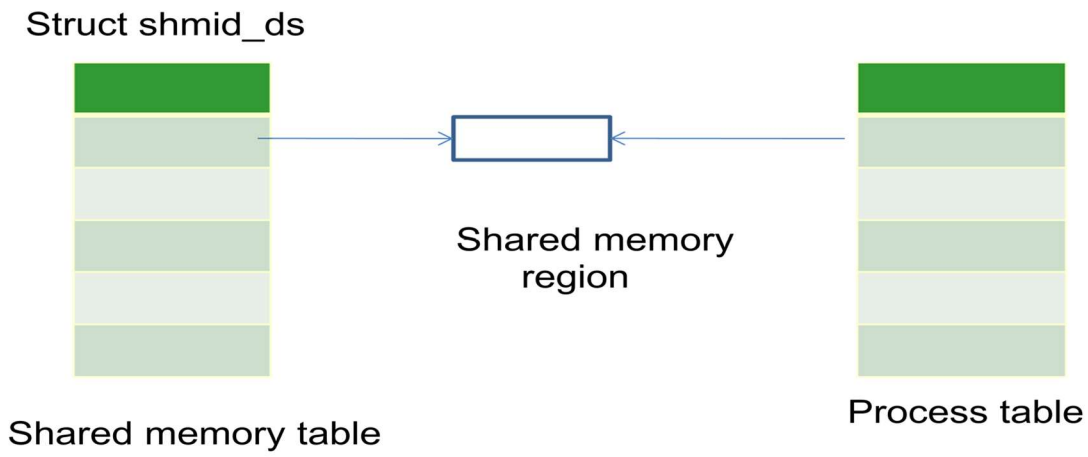
Practical 9 Shared memory

- Shared memory allows multiple processes to map portion of their virtual addresses to common memory region.
- Any process can read or write data from and to shared memory.
- Generally used with semaphore.
- Kernel address space has shared memory table to keep track of all shared memory region.
- Each entry has
    1) Integer ID key assigned by creator process to shared memory.
    2) Creator user and group ID
    3) Assigned owner user and group ID
    4) RW permission for owner, group and other
    5) Size of number of bytes
    6) Time when last process attached to region
    7) Time when last process detached to region
    8) Time when last process changed control data of region



Header files needed

```
#include<sys/types.h>
#include<sys/ipc.h>
#incluce<sys/shm.h>
```

- Shmget

  int shmget(key_t key, int size, int flag);
  - Returns non negative descriptor of shared memory. -1 if fails.
  - If key is +ve integer than opens shared memory having that key value.

- If IPC_PRIVATE then allocate new shared memory
- Size indicates size of shared memory that may be attached by calling process.
- If shared memory is created then it is size of shared memory.
- If flag is 0, system call fails if no shared memory of key ID
- If new shared memory, then key is bitwise OR of IPC_CREAT and read write permission
  ex: shmget(IPC_PRIVATE, 1024, IPC_CREAT | 0644);

- shmat

void *shmat(int shmid, void * addr, int flag);

- Attaches shared memory referenced by shmid to calling process virtual address space.
- Then process can read/write data in shared memory.
- Addr is starting virtual address to which location shared memory must be mapped. If value is 0 kernel find appropriate address.
- Flag is SHM_RND indicate that address may be rounded off to align with page boundary.
- Flag can be SHM_RDONLY indicating read only permission.
  If not set then read-write permission.
- Return value is mapped virtual address of shared memory or -1 if fails.

- shmdt

int shmdt(void *addr);

- Detaches or unmap shared memory from specified virtual address of calling process.
- Return value is 0 if succeeds and -1 if fails.

- shmctl

int shmctl(int shmid, int cmd, struct shmid_ds *buf);

- Query or change control data of shared memory
- **Buf** is address of struct shmid_ds type. It is used to specify and retrieve control data of shared memory.

Value of **cmd** are

IPC_STAT	Copy control data of shared memory to object pointed by buf /obt ain status information for the shared memory
IPC_SET	Change control data of shared memory by data specified in buf
IPC_RMID	Remove shared memory. Removal operation is delayed until all proce ss detac
SHM_LOCK	Lock shared memory must have superuser previledges.
SHM_UNLOCK	Unlock shared memory must have superuser previledges.

Program List

```
1) //This program creates shared memory
//use ipcs -m to see shared memory and ipcrm -m shmid to remove  it

#include<stdio.h>
#include<sys/shm.h>
#include<sys/ipc.h>

int main()
{
	int shmid;

	shmid=shmget(IPC_PRIVATE,2048,IPC_CREAT|0644);

	if(shmid==-1)
	{
		printf("Shared memory error...\n");
		perror("shmget");
		exit(1);
	}
	else
		printf("shmid=%d\n",shmid);
	return(0);
}
```

2) `//This program attaches some value and then detaches using shared  memory`

```
#include<stdio.h>
#include<sys/shm.h>
#include<sys/ipc.h>

int main()
{
	int shmid,stat;
	char *buf;
	int k,*val;
```

```
	struct shmid_ds sds;

	shmid=shmget(IPC_PRIVATE,100,IPC_CREAT|0644);
	if(shmid==-1)
	{
		perror("shmget");
		exit(1);
	}

	buf=(char *) shmat(shmid,0,SHM_RND);
	val=(int *)  shmat(shmid,0,SHM_RND);
	*val=10;
	buf[0]='a';

	printf("Integer=%d Character=%c\n",*val,buf[0]);

	//delete the shared memory
	shmdt(buf);
	shmdt(val);

	k=shmctl(shmid,IPC_RMID,&sds);
	if(k==-1)
	{
		perror("Error:");
		exit(2);
	}
}
```

3) `//allocate shared memory, parent process will store AAA and child will`
`//convert it to lower case. Then parent process will print`

```
#include<stdio.h>
#include<sys/shm.h>
#include<sys/ipc.h>

int main()
{
	int i,shmid,pid,id;
	char *buf;

	struct shmid_ds sds;

	shmid=shmget(IPC_PRIVATE,100,IPC_CREAT|0644);
	if(shmid==-1)
	{
		perror("shmget:");
		exit(1);
	}

	buf=(char *)shmat(shmid,0,SHM_RND);

	for(i=0;i<3;i++)
		buf[i]='A';
```

```
	pid=fork();

	if(pid==0)
	{
		//child process
		for(i=0;i<3;i++)
			buf[i]='a';
	}
	else
	{
		//parent process
		for(i=0;i<3;i++)
			printf("%c",buf[i]);
		printf("\n");
	}



	id=shmctl(shmid,IPC_RMID,NULL);
	if(id==-1)
	{
		perror("shmctl:");
		exit(2);
	}
	shmdt(buf);


}
```