Creating Shared Memory segment

Syntax:

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

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

Arguments

Key -value is to be supplied by user any integer or **ftok()** can be used to get key.

size - Indicates size of shred memory you want to create. When a new shared memory segment is created, a **nonzero** value for size must be specified. If an existing shared memory segment is being **referenced, size should be 0**.

shmflg – use IPC_CREAT **or** IPC_EXCL and permissions specified as **0666** etc.

shmflg =IPC_CREAT|0666

Example:

```
/* Creating shred memory using shmget() */

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

main()
{
        int shmid,flag;
        key_t key=0x1000;

        shmid=shmget(key,10,IPC_CREAT|0666);
        if(shmid<0)
        {
                perror("shmid failed to create ");
                exit(1);
        }
        printf("Success shmid is %d \n",shmid);
}
```

oflag argument	*key* does not exist	*key* already exists
no special flags	error, errno = ENOENT	OK, references existing object
IPC_CREAT	OK, creates new entry	OK, references existing object
IPC_CREAT | IPC_EXCL	OK, creates new entry	error, errno = EEXIST

/* After executing type- **ipcs -m** at terminal and see the created shared memory segments */

Mapping Shared Memory Segment so that other process can access.

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

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

shmat() attaches the shared memory segment identified by **shmid** to the address space of the calling process. The attaching address is specified by **shmaddr** with one of the following criteria:

If **shmaddr is NULL i.e 0**, **the system chooses a suitable (unused) address** at which to attach the segment.

Finally, the shmflg can be set to **SHM_RDONLY** if you **only want to read** from it, **0 otherwise.**

RETURN VALUE

On success shmat() **returns the address of the attached shared memory segment**; on error (void *) **-1** is returned, and errno is set to indicate the cause of the error.

Type casting can be used to cast ptr to void to appropriate type.

Note: data remains in shared memory, unless explicitly cleared using shmat() & shmctl()

Shmdt()

When a process is finished with a shared memory segment, it **detaches the segment** by calling shmdt .

```
#include <sys/shm.h>
int shmdt (const void *shmaddr) ;
```

RETURN VALUE

Returns: **0** if success otherwise **-1** on error

When a process terminates, all shared memory segments currently attached by the process are detached.

Note that this call does not delete the shared memory segment. Deletion is accomplished by calling **shmctl** with a command of IPC_RMID, which is described below.

Shmctl()

shmctl() provides a variety of operations on a shared memory segment.

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

#include <sys/shm.h>

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

shmctl() performs the control operation specified by **cmd** on the shared memory segment whose identifier is given in shmid.

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(2)/shmdt(2) */
        shmatt_t        shm_nattch;  /* No. of current attaches */
        ...
};
```

Three commands are provided:

IPC-RMID Remove the shared memory segment identified by shmid from the system and destroy the shared memory segment.

IPC-SET Set the following three members of the shmid-ds structure for the shared memory segment from the corresponding members in the structure pointed to by the buff argument: **shm_perm.uid, shm_perm.gid**, and **shm_perm.mode.** The **shm_ctime** value is also replaced with the **current time**.

IPC-STAT Return to the caller (through the buff argument) the current shmid_ds structure for the specified shared memory segment.

RETURN VALUE

Returns: 0 if success otherwise -1 on error

Reading and Writing Shared Memory Segment

Example:

/ Shared memory is created by shmget() ; shared memory segment is mapped by getting ptr to the beginning of that memory using shmat() ; in child using this pointer Hello World is copied; same segment is accessed using pointer in parent and parent displays Hello world. Therefore ptr1 displays Hello World , ptr2 displaying garbage */*

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

```
main()
{

        int shmid,pid;
        char * ptr1 , *ptr2;

                shmid=shmget(0x100,20,IPC_CREAT|0666);

                ptr1=(char *)shmat(shmid,NULL,0);
                ptr2=(char *) malloc(20);
                pid=fork();

                if(pid==0)
                {
                        strcpy(ptr1,"Hello World ");
                        strcpy(ptr2,"Good Morning World ");
                }
                else
                {

                        printf("parent reads from shared memeory (ptr1)- %s \n",ptr1);
                        // This data available here because it uses shared memory to store data, so both
                        printf("parent reads from ptr2 -  %s \n",ptr2);
                        // This data not avaiable here ptr2 because data eneted is in child
                        wait(0);

                }
}
```

This is the example where data (Hello World string) shared between two processes -child and parent.

Example: Using standard input(keyboard) to accept data putting into shared memory and shared memory content is read and displayed on (written to) standard output(VDU)

/* Using read command data is accepted from standard input(keyboard) and put into shared memory and Using write command data is read from shared memory and displayed on Standard output (VDU) */

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

```
#define SIZE 5*1024

main()
{
        struct databuf{
                int nread;
                char buf[SIZE];
                };

        struct databuf* ptr;
        int shmid,pid;

        shmid=shmget((key_t)1,SIZE,IPC_CREAT|0666);

        ptr=(struct databuf *)shmat(shmid,NULL,0);

        pid=fork();
        if(pid==0)
        {
                printf(" In child accepting data from shared memory \n");
                        // First argument 0 in read command indicates it is standard input
                ptr->nread=read(0,ptr->buf,SIZE);

        }
        else
        {
                wait(0);
                printf(" In parent reading data from shared memory \n");
i               write(1,ptr->buf,ptr->nread);
        }
}
```

***/*Example:** Imputing two numbers and operator into shared memory by child and parent takes data from shared memory and displays result*/*

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

#define SIZE 5*1024

main()
{
        struct databuf{
                int no1;
```

```
                int no2;
                char op;
                }buf;

        struct databuf* ptr;
        int shmid,pid,result;

        shmid=shmget((key_t)1,SIZE,IPC_CREAT|0666);
        ptr=(struct databuf *)shmat(shmid,(struct databuf *)0,0);
        pid=fork();
        if(pid==0)
        {
                printf(" In Child - putting data into shared memory\n");
                ptr->no1=10;
                ptr->no2=20;
                ptr->op='+';


        }
        else
        {
                wait(0);
                printf(" In Parent-taking data from shared memory  \n");

                result=ptr->no1+ptr->no2;
                printf(" Result of addition %d \n",result);
        }
}
```

Example: Simple chat program (**p315.c and p316.c**)

p315.c

/ a simple chat program this program p315.c accepts some string puts into shared memory, another program p316.c reads and displays content put by p315.c, similarly p316.c put string to shared memory and p315.c reads from memory and displays. This continues till one of them indicate end of conversation by special char @ (in p316.c and p315.c). To identify which sentence to take from shared memory, synchronization is needed, which is done by putting another special character at the beginning of string $ (p315.c puts) and *(in p316.c) */*

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


main()
```

```
{
	char c, *ptr;
	int shmid,i;

	shmid=shmget((key_t)1,500, IPC_CREAT|0666);

	ptr=(char *)shmat(shmid,0,0);

	while(1)
	{
		gets(&ptr[1]);
		ptr[0]='$';
		if(ptr[1]=='@')// ending conversation
			break;
		while(1)
		{
			if(ptr[0]=='*')
				break; //0th segment found * means , from 1th segment is the message
from p315.c
		}

		puts(&ptr[1]);
	}
	shmctl(shmid,IPC_RMID,0);
}
```

p316.c

/* a simple chat program this program p315.c accepts some string puts into shared memory , another program p316.c reads and displays content put by p315.c ,similarly p316.c put string to shared memory and p315.c reads from memory and displays. This continues till one of them indicate end of conversation by special @ (in p316.c and p315.c). To identify which sentence to take from shared memory ,synchronization is needed ,which is done by putting another special character at the beggining of string $ (p315.c puts) and *(in p316.c)*/

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


main()
{
	char c, *ptr;
	int shmid,i;
```

```
	shmid=shmget((key_t)1,500,IPC_CREAT|0666);

	ptr=(char *)shmat(shmid,0,0);
	while(1)
	{
		while(1)
		{
			if(ptr[0]=='$') //0th segment found $ means , from 1th segment is the message
from p315.c
				break;
		}
		if(ptr[1]=='@')
			break;  // ending conversation

		puts(&ptr[1]);
		gets(&ptr[1]);
		ptr[0]='*';
	}
}
```

Example: *try1b.c*

/ using shared memory to put array(using pointers) of structure and accessing from shared memory */*

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


main()
{
	/* define a structure to be used in the given shared memory segment. */
	struct country
	{
	   char name[30];
	   char capital_city[30];
	   char currency[30];
	   int population;
	};
	struct country* countries;
	int shm_id,i;

	shm_id = shmget((key_t)100, 2048, IPC_CREAT |0600);

	countries = (struct country*)(shmat(shm_id, (struct country*)0, 0));
```

```
        strcpy(countries[0].name, "U.S.A");
        strcpy(countries[0].capital_city, "Washington");
        strcpy(countries[0].currency, "U.S. Dollar");
        countries[0].population = 250000000;

        strcpy(countries[1].name, "Japan");
        strcpy(countries[1].capital_city, "Tokiyo");
        strcpy(countries[1].currency, "Yen");
        countries[1].population = 600000;

        strcpy(countries[2].name, "India");
        strcpy(countries[2].capital_city, "New Delhi");
        strcpy(countries[2].currency, "Rupees");
        countries[2].population = 900000000;

for (i=0; i < 3; i++)
        {
                printf("Country %d:\n", i+1);
                printf(" name: %s:\n", countries[i].name);
                printf(" capital city: %s:\n", countries[i].capital_city);
                printf(" currency: %s:\n", countries[i].currency);
                printf(" population: %d:\n", countries[i].population);
        }
shmctl(shm_id,IPC_RMID,0);
}
```