Shared Memory

Shared memory is a mechanism that allows multiple processes to access the same block of memory in a system. It is one of the Inter-Process Communication (IPC) methods used in operating systems to exchange data between processes efficiently.

☞ Unlike pipes, shared memory allows processes to directly access and modify data in a shared region without additional copying.

Shared memory allows processes to read and write from the same memory segment.
Header file: #include<sys/shm.h>.

☞ The kernel maintains a structure, shmid_ds with the following members for each shared memory segment:

```
struct shmid_ds {
struct ipc_perm shm_perm; /*operation permission structure */
size_t shm_segsz; /* size of segment in bytes */
pid_t shm_lpid; /*process ID of last operation */
pid_t shm_cpid; /*process ID of creator */
shmatt_t shm_nattch; /* number of current attaches */
time_t shm_atime; /* time of last shmat */
time_t shm_dtime; /* time of last shmdt */
time_t shm_ctime; /* time of last shctl */
```

Creating/Accessing a Shared Memory Segment:

```
#include <sys/shm.h>
int shmget(key_t key, size_t size, int shmflg);
```

Returns:
(1) If successful, shmget returns a nonnegative integer corresponding to the shared memory segment identifier.
(2) If unsuccessful, shmget returns -1 and sets errno.

Shared Memory Writer

Program 1 : shmwriter.c

```
#include<stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include<sys/shm.h>
int main()
{
int id,*var;
key_t key;
key=ftok("key.txt",65);
id=shmget(key,50,0664|IPC_CREAT);
printf("Shared memory Identifier=%d\n",id);
```

```
var=(int *)shmat(id, NULL,0);
*var=50;
shmdt(var);
return 0;
}
```

Explain :

id: store the shared memory segment identifier.
*var: access the shared memory segment.
key: uniquely identifies the shared memory segment.

key Generation: key can be selected one of the three ways:

IPC_PRIVATE: System select
Pick a key directly: Select a random kay
Using ftok(): System generate using the function ftok(). ftok convert a pathname and a project identifier to a System VIPC key
key_t ftok(const char *pathname, int proj_id);

ftok: Generates a unique key based on a file path (key.txt) and a project identifier (65)
key.txt should exist in the same directory as the program.
65 is a project-specific identifier that can be any number.

Create/Get Shared Memory Segment:

```
id = shmget(key, 50, 0664 | IPC_CREAT);
```

50: The size of the memory segment in bytes
0664 | IPC_CREAT: 0664: File permissions for the segment . IPC_CREAT: If the shared memory segment does not already exist, create it.

Attach to Shared Memory: shared memory segment is attached to the process’s memory space
---
```
var = (int *)shmat(id, NULL, 0);
```

shmat: Attaches the shared memory segment to the process's address space.
id: The identifier of the shared memory segment.
NULL: Let the OS decide the attachment address.
0: Flags (default behavior, such as read/write access).

Write to Shared Memory: *var = 50;

Detach from Shared Memory : shmdt(var);

Verify Shared Memory Segment: ipcrm -m <ID>

Shared Memory Reader

☞ A unique key is generated using ftok (same as in the writer program).
☞ The shared memory segment is accessed using shmget with the same key.
☞ The memory segment is attached in read-only mode using shmat.

☞ The program reads the value from the shared memory and prints it.
☞ The segment is detached using shmdt.
☞ The segment is deleted from the system using shmctl with the IPC_RMID flag.

Program 2 : shmreader.c

```
#include<stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include<sys/shm.h>
int main()
{
int id,*rvar;
key_t key;
key=ftok("key.txt",65);
id=shmget(key,50,0664);
printf("Shared memory Identifier=%d\n",id);
rvar=(int *)shmat(id, NULL,SHM_R);
printf("Value in shared memory=%d\n",*rvar);
shmdt(rvar);
shmctl(id,IPC_RMID,NULL);
return 0;
}
```

id: store the shared memory segment identifier.
*rvar: reading data from the shared memory.

Access Shared Memory Segment : id = shmget(key, 50, 0664);

shmget: Accesses an existing shared memory segment.

Attach to Shared Memory: rvar = (int *)shmat(id, NULL, SHM_R);

shmat: Attaches the shared memory segment to the process’s address space for reading.
SHM_R: Attaches the segment in read-only mode to prevent accidental modification.

Remove Shared Memory Segment: shmctl(id, IPC_RMID, NULL);

shmctl: Performs control operations on the shared memory segment.
IPC_RMID: Removes the shared memory segment from the system. This ensures it is cleaned up after use.
NULL: No additional arguments are required for this operation.

*** Q1. Create a C code named shmwriter.c to create a shared memory segment of integer size and store
500 to the segment. Create another program named shmreader.c to access the stored value from the
shared memory segment and display it. Let the shmreader.c update the value to 600. Now update the
shmwriter.c code to get the updated value and display it. You are not allowed to use semaphore.

Program 1: shmwriter.c

```
#include <stdio.h>
```

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

int main() {
    int id, *var;
    key_t key;

    key = ftok("key.txt", 65);  1.Generate a unique key

    id = shmget(key, sizeof(int), 0664 | IPC_CREAT); 2.Create a shared memory
segment of integer size
    printf("Shared memory Identifier=%d\n", id);

    var = (int *)shmat(id, NULL, 0); 3.Attach to the shared memory

    *var = 500;  4. Write the initial value (500) to the shared memory
    printf("Value written to shared memory: %d\n", *var);

    shmdt(var); 5. Detach from the shared memory

    var = (int *)shmat(id, NULL, 0); 6. Reattach to get the updated value
    printf("Updated value in shared memory: %d\n", *var);

    shmdt(var); 7.Detach again before exiting

    return 0;
}
```

Program 2: shmreader.c

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

int main() {
    int id, *rvar;
    key_t key;

    key = ftok("key.txt", 65);   1.Generate a unique key

    id = shmget(key, sizeof(int), 0664);  2.Create a shared memory segment of
integer size
    printf("Shared memory Identifier=%d\n", id);

    rvar = (int *)shmat(id, NULL, 0); 3.Attach to the shared memory

    printf("Value in shared memory: %d\n", *rvar); 4. Read and display the value
from shared memory

    *rvar = 600; 5. Update the value in shared memory to 600
    printf("Value updated in shared memory: %d\n", *rvar);

    shmdt(rvar); 6.Detach from the shared memory

    return 0;
}
```

```
gcc -o shmwriter shmwriter.c
gcc -o shmreader shmreader.c

./shmwriter

Shared memory Identifier=345
Value written to shared memory: 500

./shmreader

Shared memory Identifier=12345
Value in shared memory: 500
Value updated in shared memory: 600

Re-run the writer to see the updated value: ./shmwriter

Shared memory Identifier=12345
Value written to shared memory: 500
Updated value in shared memory: 600
```

Concept : Shared Memory in Related Processes:

```
int main()
{
pid_t pid;int shmid,*shvar;
key_t key=ftok(".",45);
shmid=shmget(key,20,0664|IPC_CREAT);
printf("Key=%x ........Shmid=%d\n",key,shmid);
shvar=shmat(shmid,NULL,0);
printf("Default initial value of shvar=%d\n",*shvar);
*shvar=10;

pid=fork();

if(pid==0){
*shvar=*shvar+90;
printf("child update=%d\n",*shvar);
exit(0);
}
else
{
wait(NULL);
*shvar=*shvar+110;
printf("parent updates=%d\n",*shvar);
}
return 0;
}
```

Q2: Write a program to create a shared memory segment of size 10 bytes. Make 4 attachments to the shared memory segment to the address space of the calling process and print the number of attachments using the structure filed number of current attachments present in the structure shmid_ds defined in the header

<sys/shm.h>. Check the number of attachment using the shell provided command ipcs -m.

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

int main() {
    int shmid;
    char *shm1, *shm2, *shm3, *shm4;
    key_t key;
    struct shmid_ds shm_info;

    key = ftok("key.txt", 65);

    shmid = shmget(key, 10, 0666 | IPC_CREAT);
    printf("Shared memory ID: %d\n", shmid);

    //Attach the shared memory segment 4 times

    shm1 = (char *)shmat(shmid, NULL, 0);
    shm2 = (char *)shmat(shmid, NULL, 0);
    shm3 = (char *)shmat(shmid, NULL, 0);
    shm4 = (char *)shmat(shmid, NULL, 0);

    printf("Number of attachments: %ld\n", shm_info.shm_nattch);

    shmdt(shm1);
    shmdt(shm2);
    shmdt(shm3);
    shmdt(shm4);

    shmctl(shmid, IPC_RMID, NULL);

    return 0;
}
```

ipcs -m : Verifies the number of attachments

```
------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x678      345        root        666        10         4
```