SHARED MEMORY

1 QUICK REFERENCE

- A shared memory segment can be addressed by
  a) its **key**: a 32 bit integer (use your SSN),
  b) its **shared memory id**: an integer assigned by system.
- Your program should have the following includes:

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

- There are four primitive operations on shared memory segments:
  a) **`int shmget (long key, int nbytes, int flags)`**

  gets **`nbytes`** bytes of shared memory and returns a shared memory id:

```
int shmid;
shmid = shmget(key, NB, 0666 | IPC_CREAT);
```

  The flag **`IPC_CREAT`** requests the creation of the segment if it did not exist already.

  b) **`char *shmat (int shmid, int address, int flags)`**

  attaches the shared memory segment to an address space:

```
char *pmem;
pmem = shmat(shmid, 0, 0);
```

  To test for error,use

```
if (pmem == (char *)(-1)) ...
```

  The shared memory segment looks now like an array of characters created using **`malloc();`**

  c) **`int shmdt(char *pmem)`**

  detaches the shared memory segment **before** destroying it

  d) **`int shmctl (int shmid, int cmd, int arg)`**

  To destroy a shared memory segment use:

```
shmctl(shmid, 0, IPC_RMID);
```

2 SYNOPSIS

Shared memory is implemented in a very similar way to messages.
Every shared memory segment in the system has an associated shared memory identifier (shmid)

The kernel maintains a structure for every shared memory segment in the system. It contains the following information :

- operation permission
- segment size
- pid of last operation
- number of processes attached
- others

System calls

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

shmget(2) creates a shared memory segment, but does not provide access to the segment for the calling process.

Using shmat(2), a process can attach the shared memory segment. This system call returns the starting address (pointer) of the shared memory segment (similar to mmap(2))

When a process is finished with a shared memory segment,it detaches the segment by calling shmdt(2) system call.

Again, once the shared memory segment is created, it exists until it is explicitly removed by creator or superuser. (use shmctl(2) or ipcrm(1))

Example 1

```
// IPC communication between a child and a parent process using
// shared memory : the parent puts messages into the shared memory
// the child reads the shared memory and prints the messages

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

```
#define SHMSIZE 15   // maximum number of bytes in a message
#define MESGNUM 2    //number of messages

void cleanup (int shm_id, char *addr); // cleanup procedure
int shm_id; // ID of shared memory

int main (int argc, char *argv[])
{
        char message [SHMSIZE]; // the message to send/receive
        int i;
        int number_of_messages; // number to be sent
        int nbytes;          // number of bytes in a message

        int number_of_messages; // number to be sent
        int nbytes;          // number of bytes in a message
        int status;
        char *addr = (char *) malloc (SHMSIZE * sizeof (char));

        number_of_messages = MESGNUM;
        nbytes = SHMSIZE;

        // set up shared memory segment using PID as the key
        if ((shm_id = shmget ((key_t) getpid(), SHMSIZE, 0666 | IPC_CREAT)) == -
        1 ) {
                printf ("Error in shared memory region setup.\n");
                exit (2);
        } // if shared memory get failed

        // attach the shared memory segment
        addr = (char *) shmat (shm_id, (char *) 0, 0);
        if (fork ()) { // true if in parent process
                // create message of required length
                 for (i=0; i < nbytes; i++)
                 message [i] = i % 26 + 'a';
                 message [nbytes] = '\0';

                // send message using the shared memory segment
                for (i = 0; i < number_of_messages; i++) {
                        if (memcpy (addr, message, nbytes+1) == NULL) {
                                puts ("Error in memory copy");
                                cleanup (shm_id, addr);
                                exit (3);
                         } // end if error in memory copy
                } // end for as many messages as requested
                 wait (&status); // wait for child to return
                 // get the message sent by the child
                strcpy (message, addr);
                printf ("Parent - message from child: \n %s\n", message);
                strcpy (message, addr);
                printf ("Parent - message from child: \n %s\n", message);
```

```
                cleanup (shm_id, addr);
                exit(0);
        } // end parent process

        // in child process
         puts ("Child - messages from parent:");
         for (i = 0; i < number_of_messages; i++) {
                if (memcpy (message, addr, nbytes+1) == NULL) {
                        puts ("Error in memcpy");
                        cleanup (shm_id, addr);
                        exit (5);
                } // end if error in shared memory get
                else
                         puts (message);
        } // end for each message sent

        strcpy (addr, "I have received your messages!");
        exit (0);
} // end main program

// remove shared memory segment
void cleanup (int shm_id, char *addr)
{
        shmdt (addr);
        shmctl (shm_id, IPC_RMID, 0);
} // end cleanup function
```

RESULT:

```
Child - messages from parent:
abcdefghijklmno
abcdefghijklmno
Parent - message from child:
I have received your messages!
```

Example 2

We assume that the processes 1 & 2 are executed sequentially

Process1 : /* write hello", read world" */

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

typedef struct {
        char word[100];
} String;

main()
{
        int shmid;
        String *pointer;
        shmid = shmget(1000, sizeof(String), 0600|IPC_CREAT);
        pointer= (String *) shmat(shmid, 0, 0);
        strcpy(pointer->word,"hello\0");
        sleep(3);
        if ((strcmp(pointer->word,"world\0")) == 0)
        {
                printf("process 1 read world\n");
                shmdt(pointer);
        }
        shmctl(shmid, IPC_RMID,0);
}
```

Process2 : /* read hello", write world" */ Only partial program

```
shmid = shmget(1000, sizeof(String), 0600|IPC_CREAT);
pointer= (String *) shmat(shmid, 0, 0);
if ((strcmp(pointer->word,"hello\0")) == 0)
{
        printf("process 2 read hello\n");
        strcpy(pointer->word,"world\0");
}
shmdt(pointer);
```