BiT
Bahir Dar University
Bahir Dar Institute of
Technology

Department of Software Engineering

Faculty of Computing

Operating Systems and System Programming Individual Assignment

System Call

Prepared By: Dagim Gizachew
ID No. – BDU1307320

Submitted to: Instructor Wendimu Baye
Submission Date: 17 - 11 – 2014

1. What / Why / How, this system call?

The Syntax for the system call is:

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

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

The `shmat` basically means ***attach shared memory.***

Before talking about the system call, let's briefly see what a shared memory is. Shared memory is a memory shared between two or more processes. Each process has its own address space, if any process wants to communicate with some information from its own address space to other processes, then it is only possible with IPC (inter process communication) techniques.

Usually, inter-related process communication is performed using Pipes or Named Pipes. Unrelated processes communication can be performed using Named Pipes or through popular IPC techniques of **Shared Memory** and Message Queues. For this document we will be focusing on shared memory.

The `shmat()`function operates on XSI shared memory. It attaches to the shared memory segment specified by `shmid` and returns the address of the shared memory segment.

The address specified by `shmaddr` is only used when `shmat()` is called from a program that uses data model ***LLP64** and attaches to a teraspace shared memory segment. Otherwise, the address specified by `shmaddr` is ignored and the actual shared memory segment address is returned regardless of the value of `shmaddr`.

The `shmaddr` argument and the setting of the `SHM_RND` bit in the `shmflg` bit-mask argument control how the segment is attached. And The segment is attached at the address specified by one of the following criteria:

- If `shmaddr` is a null pointer, the segment is attached at the first available address as selected by the kernel. This is the preferred method of attaching a segment.
- If `shmaddr` is not a null pointer and (`shmflg` & `SHM_RND`) is non-zero, the segment is attached at the address given by `shmaddr -((uintptr_t) shmaddr %SHMLBA)`.
  `SHMLBA` – is *shared memory low boundary address* and it is used to round down to the nearest multiple of a constant.
  This constant is equal to some constant multiple of the system page size. Attaching a segment at an address that is a multiple of SHMLBA is necessary on some architectures in order to improve CPU cache performance.
  `uintptr_t` – is unsigned integer type used for storing pointer addresses.
- If `shmaddr` is not a null pointer and (`shmflg` & `SHM_RND`) is 0 (not set), the segment is attached at the address given by `shmaddr`.
- The segment is attached for reading if (`shmflg` & `SHM_RDONLY`) is non-zero and the calling process has read permission; otherwise, if it is 0 and the calling process has read and write permission, the segment is attached for reading and writing.

The return values for this function are:

- Value: when it is successfully executed. The value returned is a pointer to the shared memory segment associated with the specified identifier.
- -1: when it is not successful. The *errno* variable is set to indicate the error.

2. Briefly describe about the list of parameters and flags

The parameters mentioned above in the syntax are the following:

➲ shmid
➲ shmaddr
➲ shmflg

Their explanation is stated here below.

⦿ shmid

(Input) Shared memory identifier, a positive integer. It is returned by the shmget() function and used to identify the shared memory segment.

⦿ shmaddr

(Input) Shared memory address. The address at which the calling thread would like the shared memory segment attached.

⦿ shmflg

(Input) Operations flags. The value of the shmflg parameter is either zero or is obtained by performing an OR operation on one or more of the following constants:

The required shared memory flags are:

- SHM_RDONLY - attaches the segment for read-only purpose, by default it is read-write
- SHM_RND - rounding off address to SHMLBA(*shared memory low boundary address*)
- SHM_REMAP - replaces the existing mapping in the range specified by shmaddr and continuing till the end of segment

3. List the flags, their purpose with code implementation (give Example source code with output)

The flags are SHM_RND, SHM_RDONLY, SHM_REMAP. Their use is written in the above question.

Now let's see the implementation. To implement the shmat() system call we need to use other system calls like shmget(), shmdt(), shmctl().

As mentioned in the first question, shmat() returns the address at which the shared memory segment is attached. This value can be treated like a normal C pointer; the segment looks just like any other part of the process's virtual memory.

Typically, we assign the return value from `shmat()` to a pointer to some programmer-defined structure, in order to impose that structure on the segment.

To attach a shared memory segment for read-only access, we specify the flag `SHM_RDONLY` in `shmflg`. Attempts to update the contents of a read-only segment result in a segmentation fault. If `SHM_RDONLY` is not specified, the memory can be both read and modified.

To attach a shared memory segment, a process requires read and write permissions on the segment, unless `SHM_RDONLY` is specified, in which case only read permission is required.

It is possible to attach the same shared memory segment multiple times within a process, and even to make one attach read-only while another is read-write. The contents of the memory at each attachment point are the same, since the different entries of the process virtual memory page tables are referring to the same physical pages of memory.

One final value that may be specified in `shmflg` is `SHM_REMAP`. In this case, `shmaddr` must be non-NULL. This flag requests that the `shmat()` call replace any existing shared memory attachment or memory mapping in the range starting at `shmaddr` and continuing for the length of the shared memory segment. Normally, if we try to attach a shared memory segment at an address range that is already in use, the error EINVAL results. `SHM_REMAP` is a nonstandard Linux extension.

When a process no longer needs to access a shared memory segment, it can call `shmdt()` to detach the segment from its virtual address space. The `shmaddr` argument identifies the segment to be detached. It should be a value returned by a previous call to `shmat().`

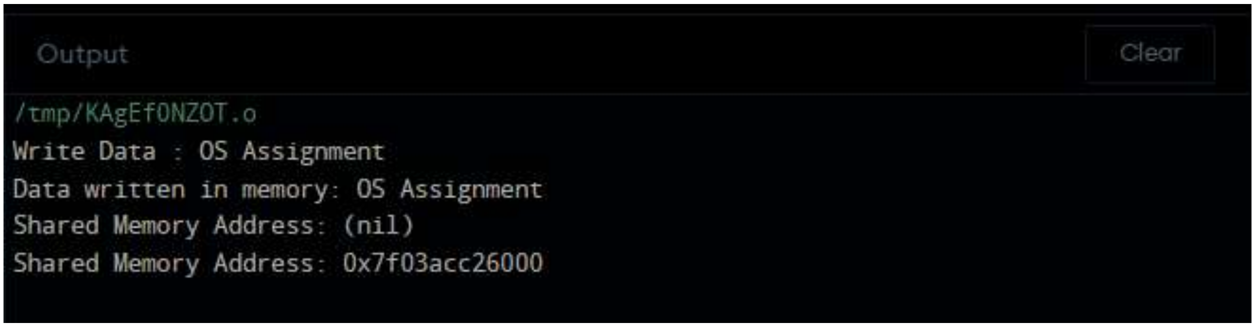
Regarding the implementation, the last two flags are difficult to implement and get a result. The system will do it for us as an abstraction.

Source code implementation.

I. Default Read/Write

- SHARED MEMORY FOR WRITER PROCESS

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

int main()
{
    // ftok to generate unique key
    key_t key = ftok("shmfile",65);

    // shmget returns an identifier in shmid
    int shmid = shmget(key,1024,0666|IPC_CREAT);

    int *shmaddr=(void*)0; //shared memory address set to null pointer
    // shmat to attach to shared memory
    char *str = (char*) shmat(shmid,shmaddr,0);


    printf("Write Data : ");
    gets(str);

    printf("Data written in memory: %s\n",str);

    printf("Shared Memory Address: %p\n",shmaddr);

    printf("Shared Memory Address: %p\n",str);
    //detach from shared memory
    shmdt(str);

    return 0;
}
```

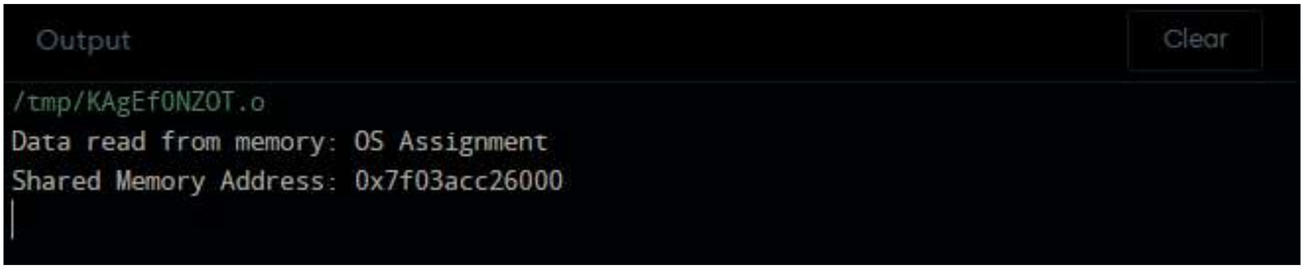


```
/tmp/KAgEf0NZOT.o
Write Data : OS Assignment
Data written in memory: OS Assignment
Shared Memory Address: (nil)
Shared Memory Address: 0x7f03acc26000
```

As we stated in the criteria, when we first give the `shmaddr` a null pointer the segment is attached at the first available address as selected by the kernel.

- SHARED MEMORY FOR READER PROCESS

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

int main()
{
    // ftok to generate unique key
    key_t key = ftok("shmfile",65);

    int *shmaddr=0x7f03acc26000;
    // shmget returns an identifier in shmid
    int shmid = shmget(key,1024,0666|IPC_CREAT);

    // shmat to attach to shared memory
    char *str = (char*) shmat(shmid,shmaddr,0);

    printf("Data read from memory: %s\n",str);

    printf("Shared Memory Address: %p\n",str);
    //detach from shared memory
    shmdt(str);

    // destroy the shared memory
    shmctl(shmid,IPC_RMID,NULL);

    return 0;
}
```

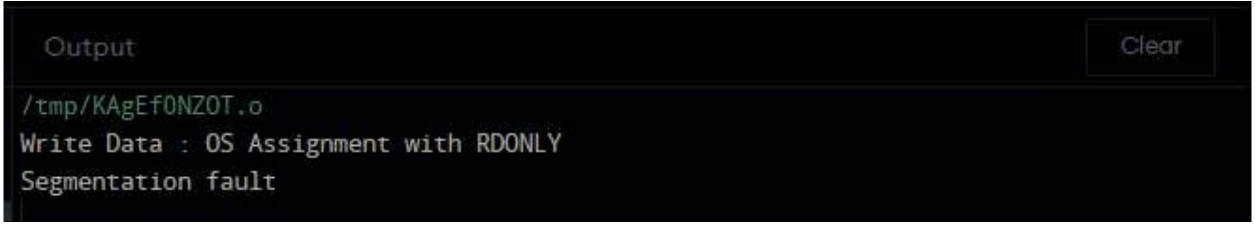


```
/tmp/KAgEf0NZOT.o
Data read from memory: OS Assignment
Shared Memory Address: 0x7f03acc26000
```

II. With `SHM_RDONLY` Flag

- SHARED MEMORY FOR WRITER PROCESS

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

int main()
{
    // ftok to generate unique key
    key_t key = ftok("shmfile",65);

    // shmget returns an identifier in shmid
    int shmid = shmget(key,1024,0666|IPC_CREAT);

    int *shmaddr=(void*)0; //shared memory address set to null pointer
    // shmat to attach to shared memory
    char *str = (char*) shmat(shmid,shmaddr,SHM_RDONLY);


    try{printf("Write Data : ");
    gets(str);

    printf("Data written in memory: %s\n",str);

    printf("Shared Memory Address: %p\n",shmaddr);

    printf("Shared Memory Address: %p\n",str);
    }
    catch
    //detach from shared memory
    shmdt(str);

    return 0;
}
```



```
/tmp/KAgEf0NZOT.o
Write Data : OS Assignment with RDONLY
Segmentation fault
```

The output will be an error because we choose to make the flag `SHM_RDONLY`.