İstanbul
Bilgi University

LAUREATE INTERNATIONAL UNIVERSITIES

CMPE 312 HOMEWORK2

System Calls

Sadiq Tijjani Umar
119200113

SHMAT() System call and it's applications

1)Provide Background Knowledge Related to the system call you are responsible with.

A pre-cursor to the details of a particular type of system call is the concept of a system call, what is a system call? . A system call provides an essential interface between a computer process and the operating system, to summarize the system briefly, a process may request a service from the operating system, the only way of asking for a service is though the system kernel, the kernel then provides an interface whereby the user process can request a service from the operating system. The system call may provide many services, including Main Memory Management, File Access, directory and networking etc. Now that this foundation has been set, we can now delve into the Shmat system call in particular.

The Shmat system call is an XSI(X/Open System Interfaces) shared memory attach operation, what it does is it merges the XSI shared memory partition indicated by shmid to the address region of the calling process. The XSI is a shared memory mechanism for shared memory segments also known as "System V" shared memory, it uses support functions such as Shmat, Shmget, Shmdt etc. Shared Memory is the most efficient multi-process dialogue mechanism, it is done when the Operating System check marks memory partitions in the address space of multiple processes such that several processes can write in and read from a memory partition without calling OS functions, In simple form, it is a way for several programs to exchange data without extra steps of computation from communications processes, the use of shared memory is helpful in the production of web applications and authorization modules for web servers because it is a low level tool for defining processes as separate(but interacting) entities.

2) Explain The System Call You Are Responsible With

The Brief Synopsis of the Shmat system call is presented below:

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

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

Description

- Shmat() system call merges the XSI shared memory partition which is identified by shmid to the address space of the calling process(shmid is the value returned by the shmget() system call which we will see breifly). Shmaddr will specify the merger address, in the case that shmaddr is NULL, the system will select a free and/or un-used address

to merge the partition. Given that shmaddr is not NULL and SHM_RND(which rounds off address to SHMBLA) is defined in shmflg, the merger occurs at the address identical to the address of the nearest multiple of SHMBLA(Lower Boundry Address), otherwise the system will choose a page aligned address to merge the memory partition. The third argument shmflg, specifies the necessary shared memory flags such as; SHM_RND, SHM_EXEC which allows the contents of the partition to be executed(given that the system call has executive permission on the partition), SHM_RDONLY which merges the partition for read only purposes(the process must always have read and write permission for the partition) and SHM_REMAP which specifies the replacement of the already existing mapping in the address range starting at shmaddr and continuing to the end of the partition(if shmaddr is NULL, then an EINVAL error will occur, which means that a mapping already exists in the address range).
A successful shmat system call will update the members of the shmid_ds structure(which specifies a shared memory partition associated with a shared memory partition ID), the members being: shm_dtime(which is the time of the last shmdt() operation) , shm_lpid(Process ID of the last process that performed shmat() system call) and shm_natchh(number of processes that share this particular partition).

Return Values

If successfully executed, the Shmat() system call will return the address of the merged shared memory partition.
If it is not executed successfully, Shmat() returns (void*) -1 , the errno variable or the perror() function will then indicate the error.

Error Cases For Shmat()

- EACESS: Indicates that the calling process has neither the necessary permissions for the requested merger type nor the CAP_IPC_OWNER capability which governs IPC(Inter Process Communication) namespace.
- EIDRM: Indicates that shmid is pointing to a removed(or missing) identifier.
- EINVAL: Indicates that a page is not aligned or a segment cannot be attached shmaddr value.
- ENOMEM: Indicates memory allocation error.

3) Demonstrate the usage of the system call you are responsible for, with an example code

An example of the shmat() system call in action would be two programs trying to exchange a string using a shared memory partition, where a reader program creates an XSI(System V) shared memory partition and an XSI semaphore set after which it will then proceed to print out the ID's of the created objects and wait for a signal from the writer program to change semaphore value. The writer program on the other hand takes in 3 cmd(command) line arguments(ID of partition, semaphore set and a string) and them merges the shared memory partition with it's address region, after which it will then decrease semaphore value in order to signal for the reader program to check the contents of shared memory. Below is the demonstration of the of the two programs:

/*Gotten from : https://man7.org/linux/man-pages/man2/shmat.2.html *\

Reader Program

```
/* svshm_string_read.c

              Licensed under GNU General Public License v2 or later.
            */
             Section 1
            #include "svshm_string.h"

            int
            main(int argc, char *argv[])
            {
                int semid, shmid;
                union semun arg, dummy;
                struct sembuf sop;
                char *addr;
                section 2
                /* Create shared memory and semaphore set containing one
                   semaphore. */

                shmid = shmget(IPC_PRIVATE, MEM_SIZE, IPC_CREAT | 0600);
                if (shmid == -1)
                    errExit("shmget");

                semid = semget(IPC_PRIVATE, 1, IPC_CREAT | 0600);
                if (shmid == -1)
                    errExit("shmget");
                section 3
                /* Attach shared memory into our address space. */

                addr = shmat(shmid, NULL, SHM_RDONLY);
```

```
    if (addr == (void *) -1)
        errExit("shmat");
```

section 4

```
    /* Initialize semaphore 0 in set with value 1. */

    arg.val = 1;
    if (semctl(semid, 0, SETVAL, arg) == -1)
        errExit("semctl");

    printf("shmid = %d; semid = %d\n", shmid, semid);
```

section 5

```
    /* Wait for semaphore value to become 0. */

    sop.sem_num = 0;
    sop.sem_op = 0;
    sop.sem_flg = 0;

    if (semop(semid, &sop, 1) == -1)
        errExit("semop");

    /* Print the string from shared memory. */

    printf("%s\n", addr);
```

section 6

```
    /* Remove shared memory and semaphore set. */

    if (shmctl(shmid, IPC_RMID, NULL) == -1)
        errExit("shmctl");
    if (semctl(semid, 0, IPC_RMID, dummy) == -1)
        errExit("semctl");

    exit(EXIT_SUCCESS);
}
```

Writer Program

```
/* svshm_string_write.c

   Licensed under GNU General Public License v2 or later.
*/
```

Section 1

```
#include "svshm_string.h"

int
main(int argc, char *argv[])
{
    int semid, shmid;
    struct sembuf sop;
    char *addr;
    size_t len;

    if (argc != 4) {
        fprintf(stderr, "Usage: %s shmid semid string\n",
argv[0]);
        exit(EXIT_FAILURE);
```

```
        }

        len = strlen(argv[3]) + 1;  /* +1 to include trailing '\0' */
        if (len > MEM_SIZE) {
            fprintf(stderr, "String is too big!\n");
            exit(EXIT_FAILURE);
        }
        Section 2
        /* Get object IDs from command-line. */

        shmid = atoi(argv[1]);
        semid = atoi(argv[2]);
        section 3
        /* Attach shared memory into our address space and copy string
           (including trailing null byte) into memory. */

        addr = shmat(shmid, NULL, 0);
        if (addr == (void *) -1)
            errExit("shmat");

        memcpy(addr, argv[3], len);
        section 4
        /* Decrement semaphore to 0. */

        sop.sem_num = 0;
        sop.sem_op = -1;
        sop.sem_flg = 0;

        if (semop(semid, &sop, 1) == -1)
            errExit("semop");

        exit(EXIT_SUCCESS);
    }
```

Bonus: Explain the source code of the system call assigned to you

Header File

- The header file is a document that simply instantiates all the necessary libraries to be used in the reader and writer programs(the reader, if curious, can find the document at : https://man7.org/linux/man-pages/man2/shmat.2.html), I have decided not to include the source code in the report but I will still explain the functions within it. The header file defines errExit(which calls the perror function to indicate an error), Union structure semun which will be used in semctl(which performs control operations defined by cmd on the System V semaphore collection identified by semid), and finally defines size of the heap memory(MEM_SIZE).

Reader Program

- **Section 1:** int main() takes in an integer and string argument, inside this function shmid, semid, sop and addr(which is a pointer) are all defined as their respective datatypes. An unsigned integer type(atleast 16 bits) is also assigned to len.
- **Section 2:** For creating a shared memory shmid is assigned the shmget() system call which returns the identifier of the XSI shared memory partition linked with the value of the argument key(in this case IPC_PRIVATE). When IPC_VALUE is used as the argument key, shmget leaves out all but the least significant nine bits of itself and then makes a new shared memory partition, shmget also takes in MEM_SIZE which is the size of the heap memory and lastly, IPC_CREAT which just creates a new partition, on failure of the system call, -1 is returned. To set up the semaphore set semid is assigned the semget() system call which returns an XSI semaphore set indicator(or identifier) given the associated argument key, it takes in IPC_PRIVATE as key and IPC_CREAT , int 1 represents number of semaphores. Incase of error in execution, -1 is returned and an error indication follows.
- **Section 3:** This section is for merging shared memory into the address region, to do this, the address region(addr) is assigned the shmat() system call which merges the XSI shared memory partition which is identified by shmid to the address region of the system calling process, the function takes in shmid which identifies the XSI shared memory partition to merged, NULL(which means the system will choose a free page aligned address to merge the partition) and SHM_RDONLY which will merge the partition for read only purposes. Given an unsuccessful execution, an error indication occurs.
- **Section 4:** In this section we need to initialize the 0 semaphore value to 1, to do this first of all, the arg(within union structure semun) is attached an integer value of 1, the semctl then performs control operations specified by cmd on the XSI semaphore taking in semid(which identifies semaphore set to be operated on), int 0(semnum value that identifies an exact semaphore within that set), SETVAL(which sets the semaphore value to arg.val for the particular semaphore within that set(semnum)) and the fourth argument arg which is of the type union semun. If the process control operation returns -1, then an error has occurred and errExit will indicate it.
- **Section 5:** In this section we want the program to wait for the semaphore value to become 0, firstly, the structure sembuf is one that defines operations to be performed on a single semaphore, as sop has been instantiated in this structure, sop.sem_num indicates semaphore number, sop.sem_op indicates semaphore operation and sop.sem_flg indicates operation flags. Semop() then performs operations on the particular semaphore in the set indicated by semid, it also takes in argument &sop(memory space of sop) and int 1 which is the semnum value, if the operation returns -1, an error indication is made, the string from shared memory is then printed.

- **Section 6:** In this section we want the program to remove shared memory and semaphore set, shmctl() performs control operations defined by cmd(in this case cmd is IPC_RMID(which marks the partition to be removed), it also takes in shmid(which identifies shared memory partition) and NULL , if the operation returns -1 then an error is indicated. Semctl() performs control operations specified by cmd on the XSI semaphore , taking in semid, int 0 as semnum value , IPC_RMID and dummy, if the process returns -1, an error is indicated, if process completed successfully an exit call is made and process terminates.

Writer Program

- **Section 1:** We want the writer program to take in three cmd line arguments(semaphore set, ID of shared memory partition, and a string) and attach the shared memory partition to it's address region, then decrement the semaphore value to 0, which then sends a signal to the reader. Firstly, int main() takes in an integer argument(argc) and a string argument of an array of memory spaces(argv),in this function typecasting is made, semid and shmid represent semaphore ID and Shared memory partition ID , the structure sembuf is one that defines operations to be performed on a single semaphore, addr is address region and size_t is an unsigned integer type. If argc is not equal to 4 then stderr is called(which is an error stream contained with every UNIX program) and a diagnostic output is made, then a call to exit and process termination is made. String length of the 4^{th} value in argv(+ 1 is added to trim trailing whitespace) is assigned to len. If len is greater than MEM_SIZE(heap memory), print error to show string is too big, an exit call is then made and process is terminated.
- **Section 2:** In this section we want the program to collect object IDs from cmd(command line), that atoi() function converts the initial portion of the 2^{nd} string in argv array to int and assigns it to shmid(Shared memory ID), it then also converts the initial portion of the 3^{rd} string in argv array to int and assigns it to semid(semaphore ID).
- **Section 3:** In this section we merge the shared memory partition into our address region and the copy string into memory. The shmat() call, as we have already seen, merges XSI shared memory partition with it's identified address region and returns the address of the merged shared memory partition. It takes in shmid(which identifies shared memory partition), NULL(which means the system will choose a free or unused page aligned address to merge the partition) and int 0(a value of 0 for the mode flag means 0_RDONLY which requests opening the file read only). If address returns (void*)-1, an error is indicated, if successful, copy address to len variable inside int main() function.

- **Section 4 :** Here we want to reduce semaphore value to 0. So we set sop.sem_num(semaphore number) to 0. sop.sem_op(semaphore operation) to -1 (decrement by 1) and sop.sem_flg(operation flag) to 0(read only). Semop() then performs operations on the particular semaphore in the set indicated by semid, it also takes in argument &sop(memory space of sop) and int 1 which is the semnum value, if the operation returns -1, an error indication is made, if executed successfully, an exit call is then made and process terminates.