CENG328 Operating Systems
Laboratory Chapter 6

1 Interprocess Communication II

In Laboratory Chapter 3, how communication between multiple processes using pipes can be established has been explained. In this chapter an another interprocess communication method, **shared memory** will be explained.

A Shared memory segment is a location in memory which is independent from address spaces of any processes. It can be created or destroyed by processes, and its contents can be used by processes. Shared memory segments are in fact very similar to memory regions allocated by malloc() function; the major difference is they are independent from memory spaces of processes, while segments allocated by malloc() are in heap memory of existing processes. Shared memory segments may stay in memory after all processes that access them get terminated too.

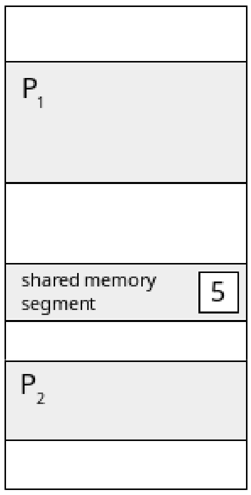

Figure 1: Shared Memory

Steps to use a shared memory segment is as follows:

1. Allocate space (if it is being created for the first time)
2. Attach to the allocated space
3. Access contents of the shared memory segment
4. Detach from the allocated space
5. Destroy (if it won't be used anymore)

Study the following code (shm1.c):

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

int main(int argc, char **argv) {
  pid_t child;
  int shmid;   // shared memory id
  int* shmptr; // pointer to shared memory

  shmid = shmget((key_t) 1234, 3 * sizeof(int), 0666 | IPC_CREAT);
  if (shmid == -1) {
    perror("shmget");
    return -1;
  }

  shmptr = (int*) shmat(shmid, NULL, 0);
  if (shmptr == (void*) -1) {
    perror("shmat");
    return -1;
  }

  shmptr[2] = 0; // we are going to calculate sum of numbers,
                 // therefore set initial value to 0

  child = fork();
  if (child == -1) {
    perror("fork");
    return -1;
  } if (child > 0) {
    waitpid(child, NULL, 0);
    int s = shmptr[0], e = shmptr[1], r = shmptr[2];
    printf("Sum of numbers from %d to %d is %d.\n", s, e, r);
  } else if (child == 0) {
    int i;
    printf("Enter start and end numbers: ");
    scanf("%d %d", &shmptr[0], &shmptr[1]);
    for (i = shmptr[0]; i <= shmptr[1]; i++)
      shmptr[2] += i;
    return 0;
  }

  if (shmdt(shmptr) == -1) {
    perror("shmdt");
    return -1;
  }

  if (shmctl(shmid, IPC_RMID, 0) == -1) {
    perror("shm remove");
    return -1;
  }

  return 0;
}
```

In this code, four functions that are associated with shared memory concept have been used:

- **shmget:** allocates and returns id of a shared memory segment
- **shmat:** attaches to the identified shared memory segment
- **shmdt:** detaches from the identified shared memory segment
- **shmctl:** shared memory control operations (remove, etc)

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

shmget returns id of a shared memory segment that is identified by the **key** variable. If no shared memory segments match with the given key, a new one is created. If key is a numeric value, any processes knowing the key may get its id. If key is defined as **IPC_PRIVATE**, only the creating process may get the id. If a new segment is to be created, its size is defined by the **size** variable (in bytes). **shmflg** defines flags such as read/write permissions of this segment, etc.

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

shmat returns starting address of a shared memory segment which is identified by **shmid**. **shmaddr** specifies which address to attach the segment. If it is NULL, it is attached to the first available address. **shmflg** specifies flags for the segment, such as read-only, etc.

At this moment, we have a pointer (shmptr) to a location in memory which is just enough to store 3 integers. This pointer can be used as any pointers you have used so far.

```
int shmdt(const void *shmaddr);
```

When all tasks related to this shared memory segment ends, the program can be detached with the help of **shmdt**. shmdt detaches from the shared memory segment specified by **shmptr**.

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

shmctl performs different tasks depending on **cmd** on the shared memory segment specified by **shmid**. For example if we want to remove a shared memory segment from the memory, cmd must be set to **IPC_RMID**.

Shared memory can be used with different programs that do not share code as well. Study the given code (shm2.c):

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

int main(int argc, char **argv) {
  int shmid;
  char* shmptr;

  shmid = shmget((key_t) 1235, sizeof(char) * 50, IPC_CREAT | 0666);
  shmptr = shmat(shmid, NULL, 0);

  printf("Enter a message: ");
  scanf("%s", shmptr);

  shmdt(shmptr);
  return 0;
}
```

This code attaches to a shared memory segment (which is 50 characters long) with 1235 as key and stores a message in it. The code below (shm3.c) attaches to the same segment, reads the message in it and destroys it:

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

int main(int argc, char **argv) {
  int shmid;
  char* shmptr;

  shmid = shmget((key_t) 1235, sizeof(char) * 50, IPC_CREAT | 0666);
  shmptr = shmat(shmid, NULL, 0);

  printf("Message stored is: %s\n", shmptr);

  shmdt(shmptr);
  shmctl(shmid, IPC_RMID, 0);
  return 0;
}
```

It is also possible to share semaphores between multiple processes using shared memory segments. Study both codes below (shm4.c, shm5.c):

```
// shm4.c
#include <stdio.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <semaphore.h>

int main(int argc, char **argv) {
  int shmid;
  sem_t* mutex;

  shmid = shmget((key_t) 1236, sizeof(sem_t), IPC_CREAT | 0666);
  mutex = shmat(shmid, NULL, 0);
  sem_init(mutex, 1, 0); // 2nd parameter denotes this semaphore
                         // will be shared between processes
                         // it is initially locked (value = 0)

  int i, limit;
    printf("How many numbers do you want to generate? ");
    scanf("%d", &limit);
  FILE* fp = fopen("list.txt", "w");
  for (i = 0; i < limit; i++)
    fprintf(fp, "%d ", rand() % 100);
  fclose(fp);

  sem_post(mutex); // file is ready, now let the other process
                   // to do its job

  shmdt(mutex);
  return 0;
}
```

```
// shm5.c
#include <stdio.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <semaphore.h>

int main(int argc, char **argv) {
  int shmid;
  sem_t* mutex;

  shmid = shmget((key_t) 1236, sizeof(sem_t), IPC_CREAT | 0666);
  mutex = shmat(shmid, NULL, 0);
  sem_wait(mutex); // wait until file is created

  FILE* fp = fopen("list.txt", "r");
  int evens = 0, odds = 0, n;
  while (!feof(fp)) {
    fscanf(fp, "%d ", &n);
    if (n % 2 == 0)
      evens++;
    else
      odds++;
  }
  printf("File has %d even and %d odd numbers.\n", evens, odds);
  fclose(fp);

  shmdt(mutex);
  shmctl(shmid, IPC_RMID, 0);
  return 0;
}
```

In the first code, a shared memory segment that is just big enough to store a semaphore variable is allocated. Then by setting the 2nd parameter of sem_init function to 1, we have specified that our semaphore variable will be shared among multiple processes (remember semaphores will be shared among threads if 2nd parameter was 0).

Open two terminal windows and compile both programs. Simultaneously run both programs at the same time, one program per terminal window. When you enter a number in 1st program and hit Enter key, you'll notice the 2nd program continues execution and prints statistics about the file that has been created by the first program.

In Linux terminal, **ipcs** command can be used to view a list of existing message queues (will not be explained in labs), shared memory segments and semaphores:

```
$ ipcs
------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x000004d3 21463047   ceng328    666        50         0
0x000004d2 21659684   ceng328    666        12         2
0x000004d4 37126193   ceng328    666        32         0
```

ipcrm command removes existing message queues, shared memory segments and semaphores:

```
$ ipcrm -m 21463047
$ ipcrm -M 1234
$ ipcs
------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x000004d4 37126193   ceng328    666        32         0
```

2 Exercises

1. Read man pages for the following library functions: shmget, shmat, shmdt, shmctl.
2. Read man pages for the following command line utilities: ipcs, ipcrm.
3. Set 2nd parameter of shm_init function in shm4.c to 0 and see the difference it causes during execution.