Shared Memory

1. shmget - Link 1(Highly recommended)
   - `int shmget(key_t key, size_t size, int shmflg)`
   - The shmget() system call returns the shared memory identifier associated with the value of the *key* argument. It may be used either to obtain the identifier of a previously created shared memory segment or to create a new one with size equal to the value of *size* rounded up to a multiple of PAGE_SIZE. The *shmflg* field can be modified using bitwise operations to use the function in different ways. For example, shmget(SHM_KEY, BUF_SIZE, 0644 | IPC_CREAT)
     Will create a new shared memory segment with id SHM_KEY if it already does not exist.
2. shmat - Link 1
   - `void *shmat(int shmid, const void *shmaddr, int shmflg)`
   - The *shmat*() function attaches the shared memory segment associated with the shared memory identifier specified by *shmid* to the address space of the calling process. *shmflg* is used to determine the operation to be performed by the *shmat()* if *shmaddr* is not null or for reading.
3. shmdt - Link 1
   - `int shmdt(const void *shmaddr)`
   - The *shmdt*() function detaches the shared memory segment located at the address specified by *shmaddr* from the address space of the calling process.
4. shmtcl - Link 1, Link 2(Highly recommended)
   - `int shmctl(int shmid, int cmd, struct shmid_ds *buf)`
   - The *shmctl*() function provides a variety of shared memory control operations as specified by *cmd*. Refer to Link 2, as mentioned above, for more information on what each value of *cmd* would do. Information about *buf* can be found in Link 1, but the field can be kept NULL or 0 depending on the purpose for which the *shmtcl()* function is used.
5. ftok - Link 1
   - `key_t ftok(const char *pathname, int proj_id)`
   - The ftok() function uses the identity of the file named by the given *pathname* (which must refer to an existing, accessible file)and the least significant 8 bits of *proj_id* (which must be nonzero) to generate a *key_t* type System V IPC key. The resulting value is the same for all pathnames that name the same file when the same value of *proj_id* is used.
6. strcpy - Link 1
   - `char* strcpy(char* destination, const char* source)`
   - The strcpy() function copies the string pointed by *source* (including the null character) to the *destination*. The *strcpy()* function also returns the copied string.
7. fgets - Link 1
   - `char *fgets (char *str, int n, FILE *stream)`
   - *str* is a pointer to an array of chars where the string read is copied. *n* is the maximum number of characters to be copied into *str*(including the terminating null character). **stream* is a pointer to a FILE object that identifies an input

stream. The fgets() function returns a pointer to the string where the input is stored.

Problem 0
Write a C program that creates a child process and sends a message to it using shared memory.

Solution

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#define SHM_KEY 0x1234
#define BUF_SIZE 8

int main() {
  int shmid;
  char *shmPtr;

  if (fork() == 0) {
     sleep(3);  // To wait for the parent to write
     // Get the shared memory ID
     shmid = shmget(SHM_KEY, BUF_SIZE, 0644);
     if (shmid == -1) {
     perror("Shared memory");
     return 1;
     }

     // Attach to the segment to get a pointer to it.
     shmPtr = shmat(shmid, NULL, 0);

     if (shmPtr == (void *)-1) {
     perror("Shared memory attach");
     return 1;
     }
     printf("Child: received message \"%s\"\n", shmPtr);

     printf("Child: Reading Done, Detaching Shared Memory\n");

     if (shmdt(shmPtr) == -1) {
     perror("shmdt");
     return 1;
     }
  } else {
     shmid = shmget(SHM_KEY, BUF_SIZE, 0644 | IPC_CREAT);
```

```
        if (shmid == -1) {
        perror("Shared memory");
        return 1;
        }

        // Attach to the segment to get a pointer to it.
        shmPtr = shmat(shmid, NULL, 0);
        if (shmPtr == (void *)-1) {
        perror("Shared memory attach");
        return 1;
        }
        sprintf(shmPtr, "%s", "Hello.");
        printf("Parent: Writing Done, waiting for child\n");

        wait(NULL);
        if (shmdt(shmPtr) == -1) {
        perror("shmdt");
        return 1;
        }

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

    return 0;
}
```