TASK 2 REPORT - IT18350906 (I.R Aushan)

Introduction:

In this report, I go into detail on the whole process I went through in order to complete the said TASK 2 of the Secure Operating Systems Assignment for 2nd Year Cyber Security students. I'll be explaining the primary objectives of the assignment as I understood them and the methods I used to satisfy them, including the errors and obstacles I encountered and how I overcame them. And in the end I'll be referencing all the resources I used.

Objectives:

The Task 2 requests use to create a program which implements an application with two entities. Student and Lecturer. The two entities should access a shared location and must access a set of files. The Lecture program has a couple unique functions that only it can access. And none of these function should interrupt each other.

Attempt 2:

Shared Memory enables two processes to talk with one another. The communication is done via the shared memory itself, where changes made by one process is visible to other processes.

The header files, <sys/ipc.h> and <sys/shm.h> allows to access specific shared memory locations identified by a unique key value. And manipulate the data inside and delete the memory segment, using shmget(), shmat() and shmctl() calls.

My objective was to use this shared memory as the shared location between the two entities and perform the functions required by the assignment. Since the case studies are stored as files, Whenever the user uploads a file, the program reads it's content and the title, combines them together into a single string, which can be separated by using the two delimiter appended in the combining process, whenever the user wants to download it.

Although it viewing the shared memory requires a bit sophisticated method to work.

```
#define SHM_SIZE 1024  /* make it a 1K shared memory segment */

int itemKey = 6969;
int shareKey = 8800;
```

First of all I use 1 kilobyte memory segments to store the data. The itemKey is a fixed memory segment that acts as a counter for the number of case files that exist in the shared memory at any given instance. The shareKey is the initial key value which the records start.

```
jkey = shareKey; //Passing the initializer
count = atoi(getSharedMemory(itemKey)); //G
jkey += count; //Adding the number of recor
count++; //Increasing the count by 1
```

Pay attention to the code segment above, which is taken from the uploadCase() function.
Assume that there are two records available in the shared memory at this moment.
Key 8800 | casefile1
Key 8801 | casefile2
getSharedMemory() function returns whatever the content that's inside the memory segment identified by the unique key value passed in as the parameter.
Therefore getSharedMemory(itemKey) should return 2, and it's passed as an integer to count variable.
jkey variable holds the initial key value, which is 8800 in this particular case. With the jkey += count; the next key value is generated for the uploadCase(), (8802)
And the count variable is incremented by one.

```
/* Updating the count of records in shared memory */
  char string1[]="";
  sprintf(string1,"%d",count);
  addToSharedMemory(string1,itemKey);
/*Updating Finished*/
```

The new count value is passed into the itemKey memory segment, updating it.

```
void listAll()
{
  int item_count = atoi(getSharedMemory(itemKey));

  printf("Case ID  | Name of the Study\n");

  for(int i=shareKey;i<(shareKey+item_count);i++)
  {
    if (validateSharedMemory(i)==0){
      continue;
    }

  char *data = getSharedMemory(i);
  char *tag = getTitle(data);

  printf("%i        | %s\n", i, tag);
  }
}
```

In order to view the files,
First the number of records that exist in the memory is passed into item_count using the same methods as explained above.
And with the for loop, The program prints out the key values and the titles of each memory segment starting from the initial key value, shareKey. Until the last possible key value, which is the total of first value + number of values.
The if condition in the middle is used for the program to continue; in case of a deleted key value is met.

Other Important Notes:
***Linear mutex are used to lock uploadCase() and deleteCase() functions.**

The ‘update’ argument pushes any file that’s given to it, to a user specified case ID. Meaning you can update a case ID.
But in order to edit a caseID, use the ‘edit’ argument. And then use the update argument to push it.

References:

https://www.tutorialspoint.com/c_standard_library/c_function_strtok.htm
https://linux.die.net/man/2/execve
https://www.geeksforgeeks.org/ipc-shared-memory/
https://www.youtube.com/watch?v=SMeDw2GDMsE - C Programming in Linux Tutorial #036 - Shared Memory
https://www.codingame.com/playgrounds/14213/how-to-play-with-strings-in-c/string-split
https://www.programmingsimplified.com/c-program-list-files-in-directory
https://www.geeksforgeeks.org/c-program-list-files-sub-directories-directory/
https://computing.llnl.gov/tutorials/pthreads/
LAB SHEET 3 to 6
Semaphores in C
How to use POSIX semaphores in C language - GeeksforGeeks
C Programming in Linux Tutorial #046 - Semaphore pthread - YouTube
Server Client Using Semaphores and Shared memory | C Programming | Software Architecture
Client server program with shared memory and semaphores in C - Stack OverflowSynchronization With Semaphores - Multithreaded Programming Guide
C library function sprintf()
C library function - memcpy()
arrays - C copy char * to char[] - Stack Overflow
Shared Memory

https://www.youtube.com/playlist?list=PL-suslzEBiMrqFeagWE9MMWR9ZiYgWq89 - Dr. Brian Fraser's playlist of C programming on Linux ****
https://www.youtube.com/watch?v=CKNjXvMB0MY&list=PLypxmOPCOkHXbJhUgjRaV2pD9MJklArhg - ShellWave Youtube Playlist on C programming ****