SOSAssignment1_Task2

ID_Number: IT19022666

Name: P.T.D. Thissera

Introduction

This report contains a detailed description of the overall process I undertaken to complete those two tasks in the Secure Operating System Assignment for Second-Year Cybersecurity Students. This report describes the main objectives of the assignment, the methods I used to satisfy them, the errors, and constraints I faced, and how I overcame them. In the end, I will also mention all the resources used to complete this assignment.

Task2 requires you to create a program that will run an application with two users. Namely, a student and a lecturer. Both users must be able to access a shared location and be able to access a group of files. There are several unique functions that lecturer can access. A student cannot access them. Thus, the student and the lecturer should be given separate privileges. And none of these tasks should be interrupted.

Attempt

Shared memory allows two processes to communicate with each other. Communication is done through the use of shared memory so that one process can see changes made by other processes The <sys / ipc.h> and <sys / shm.h> header file allows access to specific shared memory locations identified by a unique key value. Use the shmget (), shmat () and shmctl () functions to manipulate the data inside and delete the memory partition.

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 subjects are stored as text 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.

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

```
#define SHMSIZE 1024  //1K shared memory segment for each record

int iKey = 6969;//Item_Key
int sKey = 5500;//Share_Key

//char SEM_NAME[]= "vik";
```

Pay attention to the code segment above, which is taken from the **uploadSubject()** function. Assume that there are three records available in the shared memory at this moment.

```
root@kali:~# ./a view
Subject_ID   | Name of the Subject
5500         |
5501         |SOS
5502         |SNP
root@kali:~#
```

get_Shared_Memory() function returns whatever the content that is inside the memory segment identified by the unique key value passed in as the parameter. Therefore, **get_Shared_Memory(iKey)** should return, and it is passed as an integer to count variable. **skey** variable holds the initial key value, which is 7700 in this particular case. With the **skey += count;** the next key value is generated for the **upload(),** and the count variable is incremented by one.

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

```
//Updating the count of records in shared memory
        char string1[]="";
        sprintf(string1,"%d",count);
        add_To_Shared_Memory(string1,iKey);
```

In order to view the files, First the number of records that exist in the memory is passed into **s_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.

```
void all() //This section is explained in the report -- ****** read_it ********
{
        int item_count = atoi(get_Shared_Memory(iKey));

  printf("Subject_ID   | Name of the Subject\n");

        for(int i = sKey; i < (sKey + item_count); i++)
        {
                if (validate_Shared_Memory(i) == 0){
      continue;
    }

        char *data = get_Shared_Memory(i);
        char *tag = get_Title(data);

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

Outputs of the task 2

Lecture

```
root@kali:~# gcc Lecturer.c -o a -lpthread
root@kali:~# ./a upload
Enter path to Subject to upload: Subject.txt
Enter the name for case study : ICS
You are UPLOAD the data successfully!!!!!!!

Subject_ID is - 5500
Please use the Subject_ID to access the files
root@kali:~# ./a upload
Enter path to Subject to upload: Subject.txt
Enter the name for case study : SOS
You are UPLOAD the data successfully!!!!!!!

Subject_ID is - 5501
Please use the Subject_ID to access the files
root@kali:~# ./a upload
Enter path to Subject to upload: Subject.txt
Enter the name for case study : SNP
You are UPLOAD the data successfully!!!!!!!

Subject_ID is - 5502
Please use the Subject_ID to access the files
```

```
root@kali:~# gcc Lecturer.c -o a -lpthread
root@kali:~# ./a update
Enter Subject_ID to update Subject: 5501
Enter path to the edited file:
Subject1.txt
Enter the Subject title: DMSS
You are UPDATE data successfully!!!!!!
```

```
root@kali:~# ./a view
Subject_ID   | Name of the Subject
5500         |ICS
5501         |DMSS
5502         |SNP
root@kali:~#
```

```
root@kali:~# ./a download
Subject_ID: 5502
File Downloaded Successfully!!!
root@kali:~#
```

```
root@kali:~# ./a delete
Enter Subject_ID to delete: 5501
Subject Deleted Successfully
root@kali:~# ./a view
Subject_ID   | Name of the Subject
5500         |
5502         |SNP
root@kali:~#
```

Student

```
root@kali:~# vi Student.c
root@kali:~# gcc Student.c -o a -lpthread
root@kali:~# ./a view
Subject_ID   | Name of the Subject
5500         |
5502         | SNP
root@kali:~#
```

```
root@kali:~# ./a download
Enter The subject_ID: 5500
File Downloaded Successfully
root@kali:~#
```