# CPE 435: OPERATING SYSTEMS LABORATORY.

**Lab04**

**IPC Using Shared Memory.**

**Submitted by**: Dan Otieno.

**Date of Experiment**: 02/03/23.

**Report Deadline**: 02/10/23.

**Demonstration Deadline**: 02/10/23.

# Introduction:

The goal for this lab was to understand create two processes without using the parent-child relation, which is mostly the method that has been used for previous labs. This involves writing codes in three separate files; a header file to store variables, and two source files.

# Theory:

For this lab, to implement two processes without using a parent and child relationship, we used the concept of shared memory. We do so by setting up one process to create a shared segment, whose process ID can be accessed by other processes. We also generate a key that is used by other processes to interact with the shared segment that was created by the first process, in which they can read and write data. When done, each processes detaches from the shared segment. For this lab, we will have a shared region, a struct datatype containing 4 variables, sum, first and second numbers of type float, and an integer flag. The first process sets the values for first and second numbers, then it updates the value of flag to 1 if there is new data. The second process determines and displays the sum of the two numbers, and then updates the flag to 0 to indicate that the process is complete. The two processes continue running until the user sets a flag of -1 via input. Some of the theory topics explored in this lab are below:

- ## Shmget():
    - This is a function that returns the identifier of the shared memory segment associated with the value of the key. It may be used either to obtain the identifier of a previously created shared memory segment (shmflg is 0), or to create a new set. According to an IBM article, [A shared memory identifier, associated data structure and shared memory segment of at least size bytes, see <sys/shm.h>, are created for key if one of the following is true:
        - Argument key has a value of IPC_PRIVATE.
        - Argument key does not already have a shared memory identifier associated with it and the flag IPC_CREAT was specified.] - Source.

- ## Shmat():
    - This is a function that attaches the shared memory segment associated with the shared memory identifier, shmid, and returns the pointer in the calling process address space. [The segment is attached at the address specified by one of the following criteria:
        - If shmaddr is a NULL pointer, the segment is attached at the first available address as selected by the system.
        - If shmaddr is not a NULL pointer, and the flag, SHM_RND was specified, the segment is attached at the address given by (shmaddr-((prtdiff_t)shmaddr%SHMLBA)) where % is the 'C' language remainder operator.
        - If shmaddr is not a NULL pointer, and the flag, SHM_RND was not specified, the segment is attached at the address given by shmaddr.

- The segment is attached for reading if the flag, SHM_RDONLY, is specified with shmflg and the calling process has read permission. If the flag is not set and the process has both read and write permission, the segment is attached for reading and writing. The first attach of newly created __IPC_MEGA segment, as well as subsequent attaches, will have write access to the segment, regardless of the SHM_RDONLY option.
- All attaches to an __IPC_MEGA shared memory segment have the same Write or Read access authority. If a segment is enabled for writes then all attaches have the ability to read and write to the segment. If the segment is disabled for writes, then all attaches have the ability to read from the segment and cannot write to the segment. The first attach of newly created __IPC_MEGA segment, as well as subsequent attaches, will have write access to the segment, regardless of the SHM_RDONLY option. Write/Read access can be changed by the shmctl() function, Shared Memory Control Operations.] - Source.

## - Shmctl():

- This function is defined by the value of shmid. When shmid identifies a shared memory segment, shmctl performs the control operation in that segment. [The shmctl() function provides a variety of shared memory control operations on the shared memory segment identified by the argument, shmid. The argument cmd specifies the shared memory control operation and may be any of the following values:
  - IPC_STAT
    - This command obtains status information for the shared memory segment specified by the shared memory identifier, shmid. It places the current value of each member of the shmid_ds data structure associated with shmid into the structure pointed to by buf. The contents of this structure is defined in <sys/shm.h>. This command requires read permission.
  - IPC_SET
    - Set the value of the following members of the shmid_ds data structure associated with shmid to the corresponding value in the structure pointed to by buf:

      shm_perm.uid

      shm_perm.gid

      shm_perm.mode (only the low-order 9 bits).

      This command can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges or to the value of shm_perm.cuid or shm_perm.uid in the shmid_ds data structure associated with shmid.

- Using the IPC_SET function to change the IPC_MODE for an __IPC_MEGA shared memory segment will have an immediate effect on all attaches to the target segment. That is, the read and write access of all current attachers is immediately affected by the permissions specified in the new IPC_MODE. To determine how the new mode affects access, you must consider the effect of all three parts of the mode field (the owner permissions, group permissions and other permissions). If all three read and all three write permissions in the new mode are set off, then the access for all attachors is changed to read. If any of the three read permission bits is set on but the corresponding write permission bit is off, then the access for all attachors is changed to read. Otherwise, the access of all attachors is changed to write.

- IPC_RMID
    - Remove the shared memory identified specified by shmid from the system and destroy the shared memory segment and shmid_ds data structure associated with shmid. This command can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges or to the value of shm_perm.cuid or shm_perm.uid in the shmid_ds data structure associated with shmid. The remove will be completed asynchronous to the return from the shmctl() function, when the last attachment is detached. When IPC_RMID is processed, no further attaches will be allowed.
- The IPC_STAT option of shmctl() returns a structure named shmid_ds (mapped in shm.h). shmid_ds contains status information for the requested shared memory segment.

- As part of its own dump priority support, the USS Kernel will be adding a new field to shmid_ds. This field will contain the dump priority of the requested shared memory segment. The new field will be added to the shmid_ds structure in shm.h.] - Source.

- Shmdt():
    - [The shmdt() function detaches from the calling process's address space the shared memory segment located at the address specified by the argument shmaddr. Storage in the user address space for a segment with the __IPC_SHAREAS attribute is not cleaned up unless the segment is no longer attached to by other processes in the address space.] - Source.

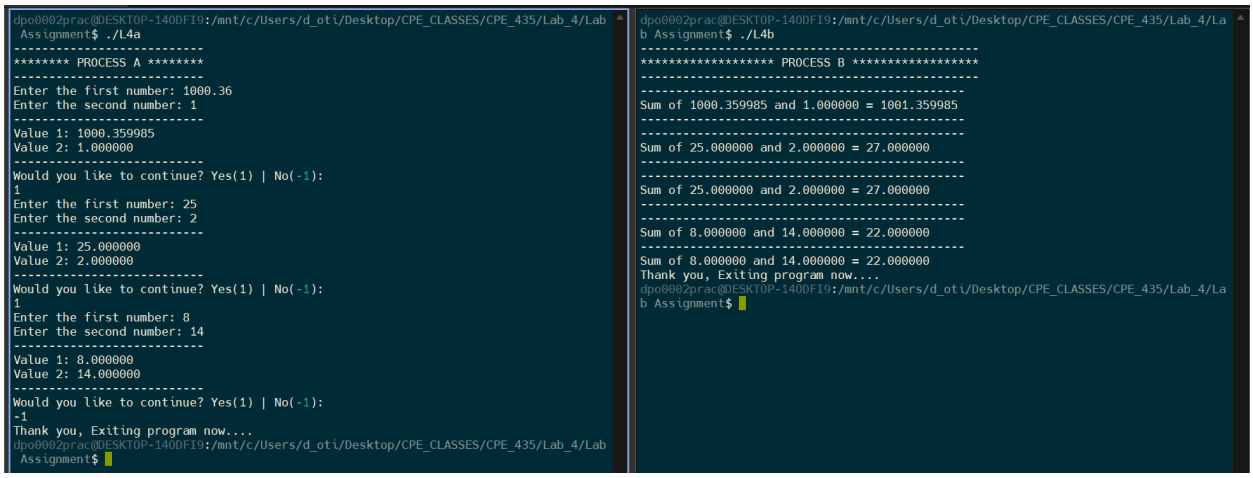
## Results & Observation:

### Assignment 1:

## Description:

The goal for this assignment was to use C/C++ to write a program that demonstrates the interaction between two separate processes that are not part of a parent-child relationship as described in theory above.

# Program Output:




This was an interesting lab, and well understood creating two processes without forking.

# Appendix:

**Assignment 1 code - Header.**

```
/*
DAN OTIENO
CPE 434-01
LAB 4
Exercise 1
*/

struct info {
        float num1, num2;
        float sum;
        int flag, choice;
};
#define  KEY ((key_t)(1243))
#define  MSIZ  sizeof(struct info)
```

## Assignment 1 code – Process A.

```
/*
DAN OTIENO
CPE 434-01
LAB 4
Exercise 1
*/
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <unistd.h>
#include "lab4.h"
int main(void)
{
   int  i, id  ;
   struct info *ctrl;
   if ( (id = shmget( KEY,MSIZ,IPC_CREAT | 0666) ) < 0 )
   {
            //error…;
            printf("An error has occurred...closing program\n");
            exit(1) ;
   }

          // attach a local pointer to the shared memory segment created
          // .. exit if fails
   if ( (ctrl = (struct info *) shmat( id, 0, 0)) <= (struct info *) (0) )
          {
```

```
            //error … ;
    printf("Another error has occurred...closing program\n");
            exit(1) ;
    }
    printf("--------------------------\n");
    printf("******** PROCESS A ********\n");
    printf("--------------------------\n");
    while(1)
    {
            printf("Enter the first number: ");
            scanf("%f", &ctrl->num1);
            printf("Enter the second number: ");
            scanf("%f", &ctrl->num2);
            printf("--------------------------\n");
            printf("Value 1: %f\n", ctrl->num1);
            printf("Value 2: %f\n", ctrl->num2);
            printf("--------------------------\n");
            fflush(stdout);
            ctrl->flag = 1;
            printf("Would you like to continue? Yes(1) | No(-1): \n");
            scanf("%i", &ctrl->choice);
            if(ctrl->choice == 1)
            {
                    ctrl->flag = 1;
            }

            if(ctrl->choice == -1)
            {
```

```
                        ctrl->flag = -1;
                        printf("Thank you, Exiting program now....\n");
                        fflush(stdout);
                        //now detach the pointer from the shared memory
                        shmdt(ctrl);
                        //let us delete the shared memory
                        shmctl(id,IPC_RMID,NULL);
                        exit(0);
                }
        }
}
```

## Assignment 1 code – Process B.

```
/*
DAN OTIENO
CPE 434-01
LAB 4
Exercise 1
*/
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include "lab4.h"

int main(void)
{
        int id;
        struct info* ctrl;
```

```
        if ((id = shmget(KEY, MSIZ, 0)) < 0)
        {
                printf("An error has occurred...closing program\n");
                exit(1);
        }

        ctrl = (struct info*) shmat(id, 0, 0);

        if (ctrl <= (struct info*)(0))
        {
                printf("An error has occurred...closing program\n");
                exit(1);
        }

        printf("-------------------------------------------------\n");
        printf("****************** PROCESS B *****************\n");
        printf("-------------------------------------------------\n");

        while(1)
        {
                while(ctrl->flag == 0 && ctrl->flag != -1);
                if(ctrl->choice == 1)
                {
                        ctrl->sum = ctrl->num1 + ctrl->num2; // calculate and
store sum
                        printf("-----------------------------------------------
---\n");
                        printf("Sum of %f and %f = %f\n", ctrl->num1, ctrl-
>num2, ctrl->sum);
                        printf("-----------------------------------------------
---\n");
                        ctrl->flag = 0; // set the flag for process A
```

```
                }
                else if(ctrl->choice == -1)
                {
                        ctrl->sum = ctrl->num1 + ctrl->num2; // calculate and
store sum
                        printf("Sum of %f and %f = %f\n", ctrl->num1, ctrl-
>num2, ctrl->sum);
                        printf("Thank you, Exiting program now....\n");
                        exit(0);
                }
        }

        shmdt(ctrl);
        exit(0);
}
```