Chat Box Application

Linux Shared Memory studies

Group Member

Nguyen Tuan Dung - 20194427

Nguyen Huy Hoang - 20194433

Nguyen Van Thanh Tung - 20190090

Tran Duc Duy - 20205181

Table Of Contents

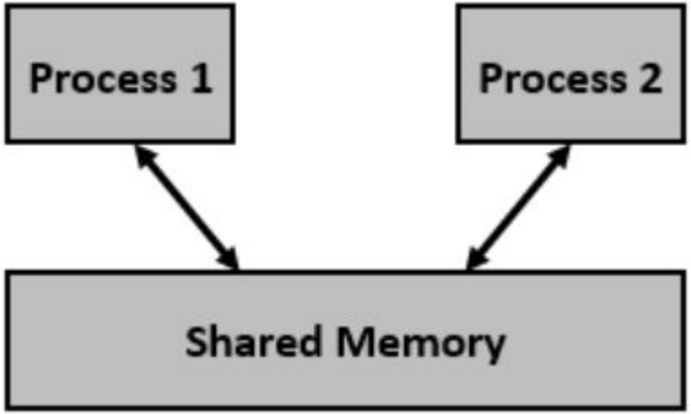
1. Introduction
   a. Shared Memory
   b. Semaphore
2. Chat Box Application - Components
   a. Scenario
   b. Shared Memory Area
   c. Semaphore Lock
   d. Messages & Display
3. Chat Box Application - Combination & Proof
   a. Combination
   b. Proof
4. Conclusion

Introduction

Shared Memory

Shared Memory

- A technique used for IPC
- A memory management technique in the Linux operating system
- Allow multiple processes to share, access a portion of memory
- Allows for the creation of a common data region that can be accessed by multiple process

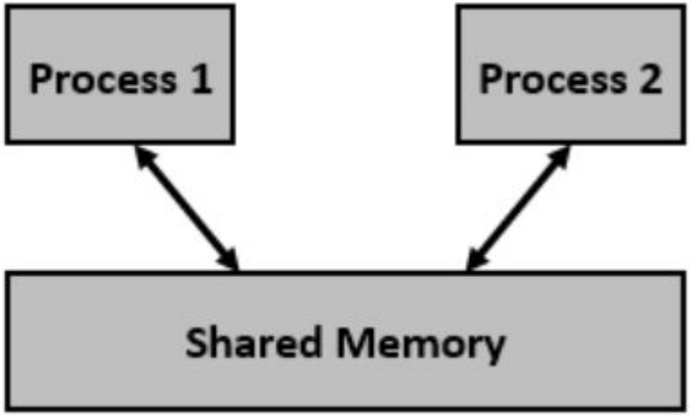


Shared Memory - Linux System calls

shmget()	Create a new shared memory segment or to access an existing shared memory segment
shmat()	Attach a shared memory segment to a process's address space
shmdt()	Detach a shared memory segment from a process's address space
shmctl()	Control the shared memory segment

Shared Memory

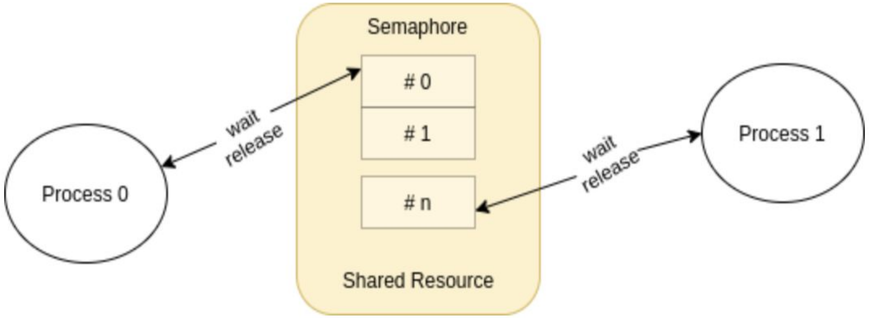
- A memory management technique in the Linux operating system
- Allow multiple processes to share, access a portion of memory
- Allows for the creation of a common data region that can be accessed by multiple process
- How to synchronize the accesses?
  - **Need a synchronization mechanism!!!**



Semaphore

Semaphore

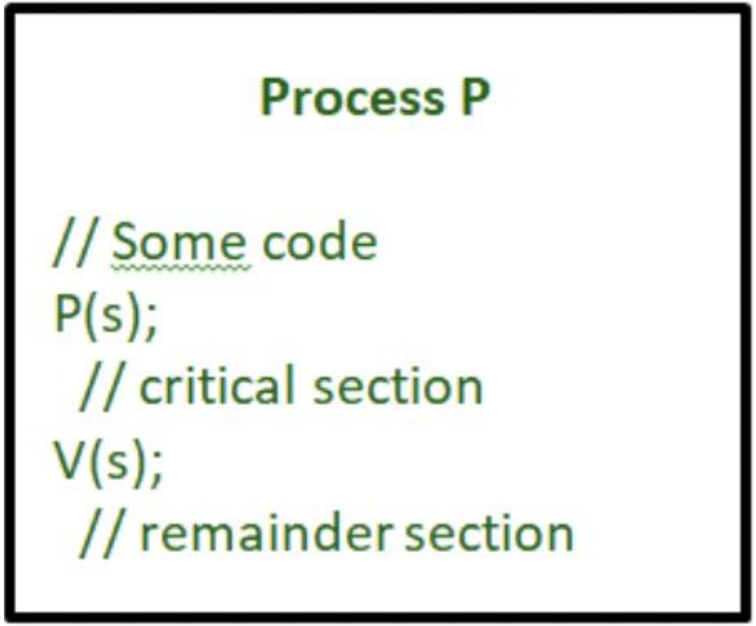
- Synchronization object
- Used to control access to shared resource in multi-threaded/process environment



Semaphore

- Typically has two operation:
    - **wait() - P()**
    - **signal() - V()**
- Semaphore value > 0: it decrements the value and continues the critical section
- Semaphore value = 0: the process is blocked until other process finished with the resource, signals the semaphore value, wake up the blocked process

→ **Semaphore can be used to synchronize Shared Memory Area**



Process P

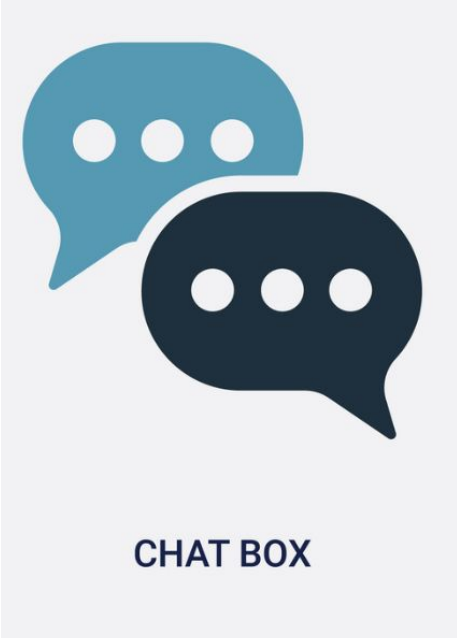
```
// Some code
P(s);
 // critical section
V(s);
 // remainder section
```

Chat Box Application Components

Scenario

Chat Box - Scenario

- The chat box is where the user can chat with other users
- Each user have a unique name
- Each user can send message to the chat box, display received messages on the screen

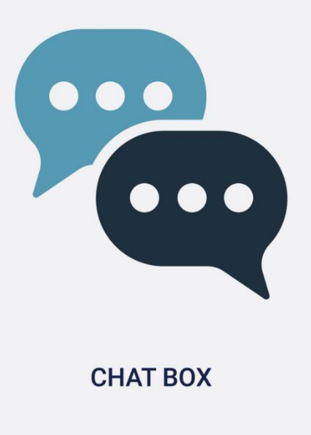


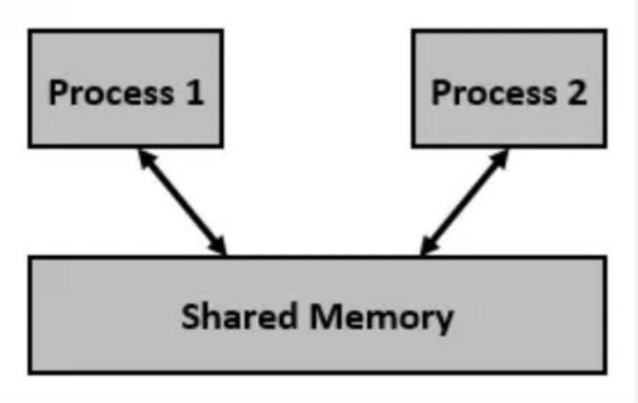
Chat Box - Scenario

- The chat box is where the user can chat with other users
- Each user have a unique name
- Each user can send message to the chat box, display received messages on the screen

→ Basically,

- The chat box is a shared memory area
- Each user can create a separate process to join the chat box




Process 1

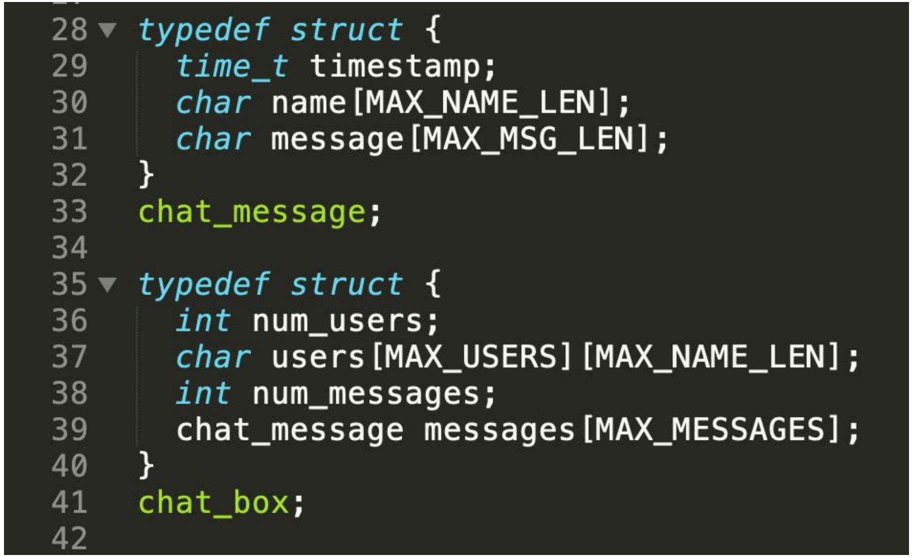
Process 2

Shared Memory

Shared Memory Area

Chat Box - Shared Memory Area

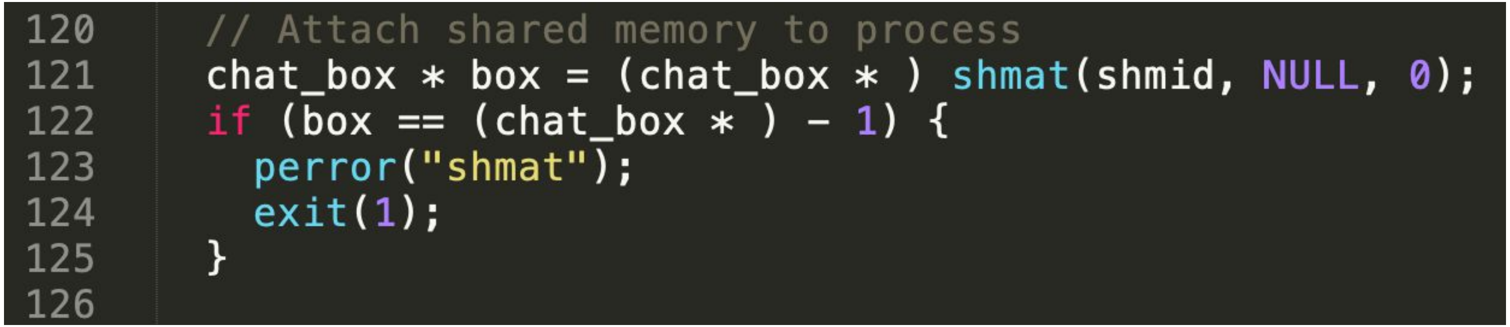
- We declared the chat box with users and messages
- The chat box has been pre-allocated an amount of memory by the size of these constant:
  - **MAX_USERS**
  - **MAX_NAME_LEN**
  - **MAX_MESSAGES**
- Messages are identified by the time of sending, the sender, and the content of the message



```c
typedef struct {
    time_t timestamp;
    char name[MAX_NAME_LEN];
    char message[MAX_MSG_LEN];
}
chat_message;

typedef struct {
    int num_users;
    char users[MAX_USERS][MAX_NAME_LEN];
    int num_messages;
    chat_message messages[MAX_MESSAGES];
}
chat_box;
```

Chat Box - Shared Memory Area

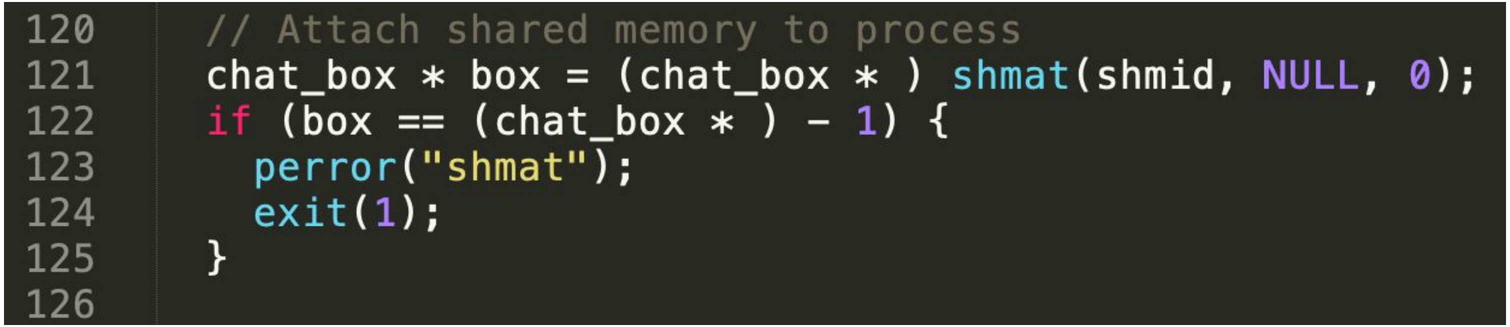
When a user wanna join the box chat: attach the shared memory area which declared the chat_box to the user's chat process



```
// Attach shared memory to process
chat_box * box = (chat_box * ) shmat(shmid, NULL, 0);
if (box == (chat_box * ) - 1) {
  perror("shmat");
  exit(1);
}

```

Chat Box - Shared Memory Area

When a user wanna join the box chat: attach the shared memory area which declared the chat_box to the user’s chat process



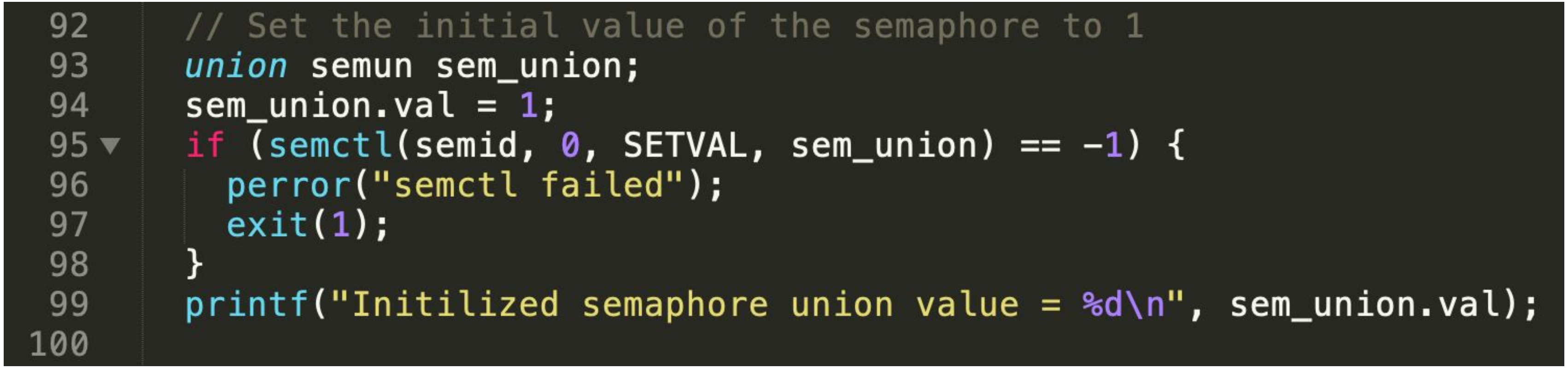
```c
// Attach shared memory to process
chat_box * box = (chat_box * ) shmat(shmid, NULL, 0);
if (box == (chat_box * ) - 1) {
  perror("shmat");
  exit(1);
}

```

And detach when exit from the chat box: `**shmdt(box)**`

Semaphore Lock

Semaphore Lock

- Messages and users data saved in shared memory area
- JUST 1 PROCESS CAN MODIFY IT AT ONE TIME
- We use binary semaphore to lock the access of processes to the shared memory area
    - **0 <= sem_union.val <= 1**



```c
// Set the initial value of the semaphore to 1
union semun sem_union;
sem_union.val = 1;
if (semctl(semid, 0, SETVAL, sem_union) == -1) {
  perror("semctl failed");
  exit(1);
}
printf("Initilized semaphore union value = %d\n", sem_union.val);

```

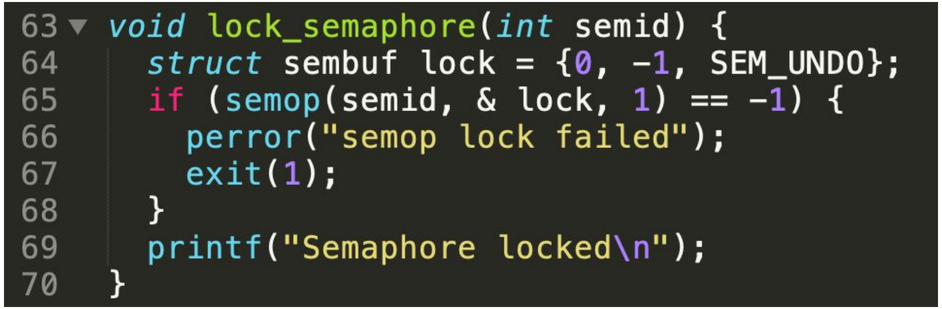
Semaphore Lock - Unlock

We also provide lock and unlock function for the purpose of easy to use in processes

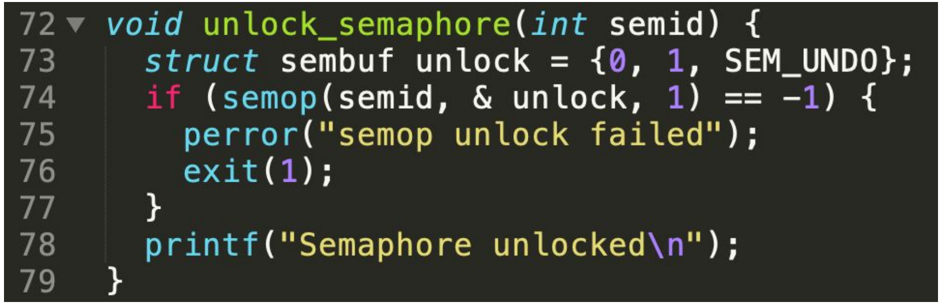
When ever process access to shared memory area:

- Add a user
- Save a message

We will lock then unlock after the modification done successfully



```c
void lock_semaphore(int semid) {
  struct sembuf lock = {0, -1, SEM_UNDO};
  if (semop(semid, & lock, 1) == -1) {
    perror("semop lock failed");
    exit(1);
  }
  printf("Semaphore locked\n");
}
```



```c
void unlock_semaphore(int semid) {
  struct sembuf unlock = {0, 1, SEM_UNDO};
  if (semop(semid, & unlock, 1) == -1) {
    perror("semop unlock failed");
    exit(1);
  }
  printf("Semaphore unlocked\n");
}
```

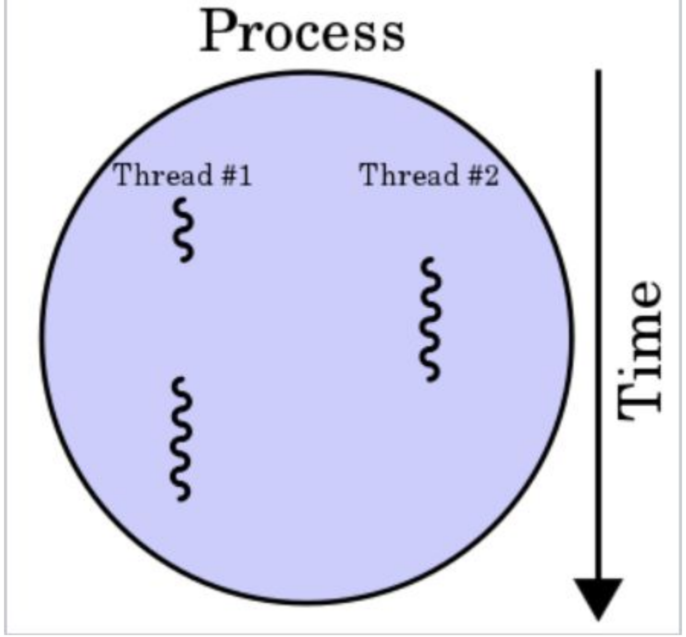
Messages & Display

Chat Box - Messages & Display

As each user represented by one process and each process have 1 main thread, we create another thread:

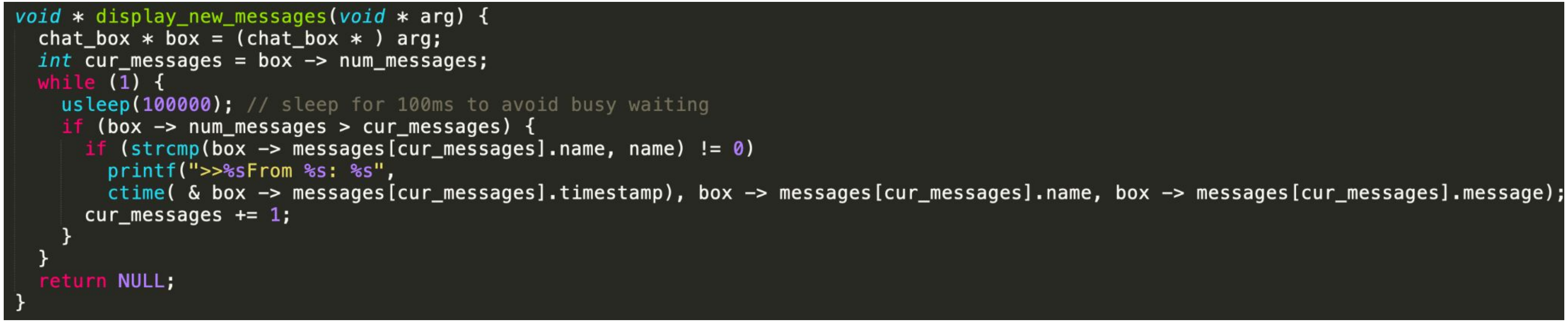
- The main thread for the purposes of sending message
- The sub thread for the purposes of display received message

→ Allow multi-tasking



Chat Box - Display Thread

- Store a pointer point to the latest messages displayed
- Not display the message of owner



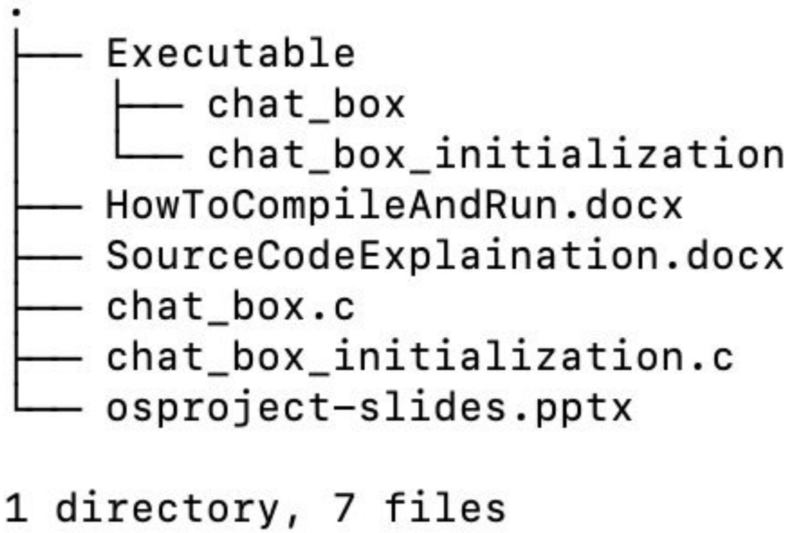
```c
void * display_new_messages(void * arg) {
  chat_box * box = (chat_box * ) arg;
  int cur_messages = box -> num_messages;
  while (1) {
    usleep(100000); // sleep for 100ms to avoid busy waiting
    if (box -> num_messages > cur_messages) {
      if (strcmp(box -> messages[cur_messages].name, name) != 0)
        printf(">>%sFrom %s: %s",
        ctime( & box -> messages[cur_messages].timestamp), box -> messages[cur_messages].name, box -> messages[cur_messages].message);
      cur_messages += 1;
    }
  }
  return NULL;
}
```

Chat Box Application Combination & Proof

Final Products - Combination

Final Products

- Combine above components, we executed, extract 2 executable files in the executable folder
- The `chat_box_initialzization`: for the main purpose of create new chat box or delete existing chat box and create new one
- The `chat_box`: user can run this file to access the chat box



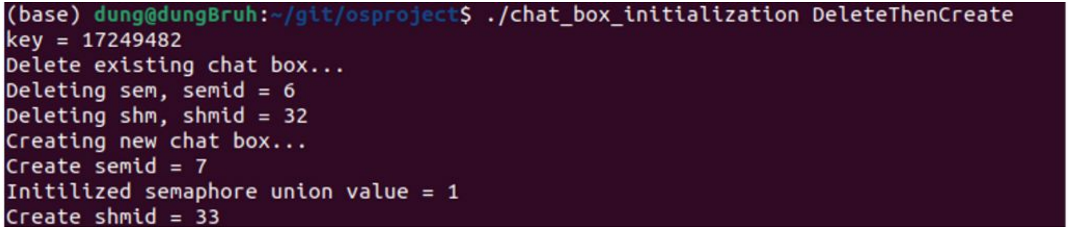
```
.
├── Executable
│   ├── chat_box
│   └── chat_box_initialization
├── HowToCompileAndRun.docx
├── SourceCodeExplaination.docx
├── chat_box.c
├── chat_box_initialization.c
└── osproject-slides.pptx

1 directory, 7 files
```

Proof

Proof - Chat Box Initialization

We deleted existing chat_box…

Then create new one with new `semid`, `shmid`

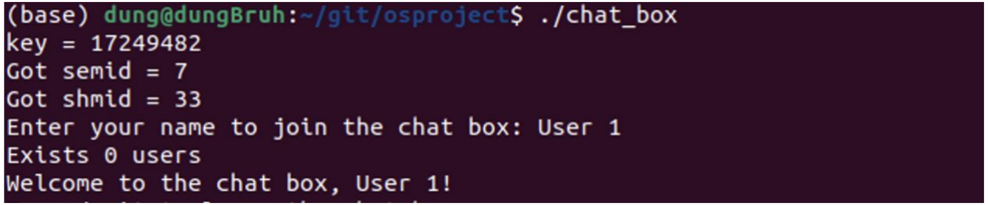


```
(base) dung@dungBruh:~/git/osproject$ ./chat_box_initialization DeleteThenCreate
key = 17249482
Delete existing chat box...
Deleting sem, semid = 6
Deleting shm, shmid = 32
Creating new chat box...
Create semid = 7
Initilized semaphore union value = 1
Create shmid = 33
```

Proof - User Join

User join the chat box by key, with the same `semid`, `shmid` as initialized

User name = `User 1`



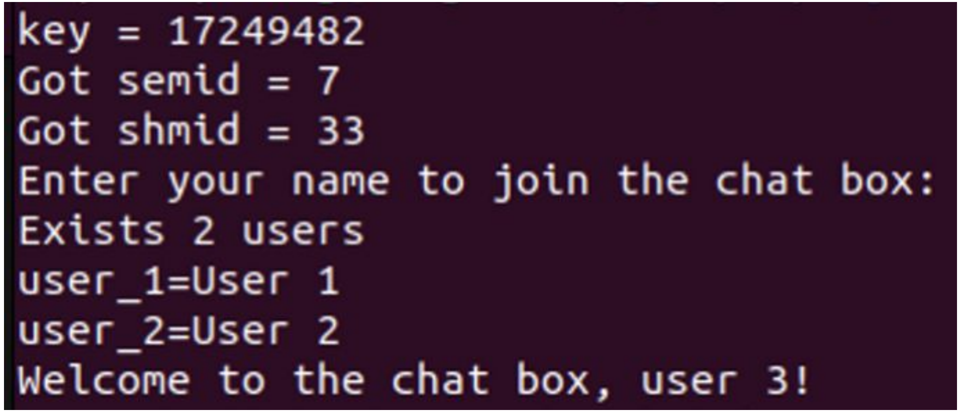
```
(base) dung@dungBruh:~/git/osproject$ ./chat_box
key = 17249482
Got semid = 7
Got shmid = 33
Enter your name to join the chat box: User 1
Exists 0 users
Welcome to the chat box, User 1!
```

Proof - Users Join

More users join the chat box by key, with the same `semid`, `shmid` as initialized

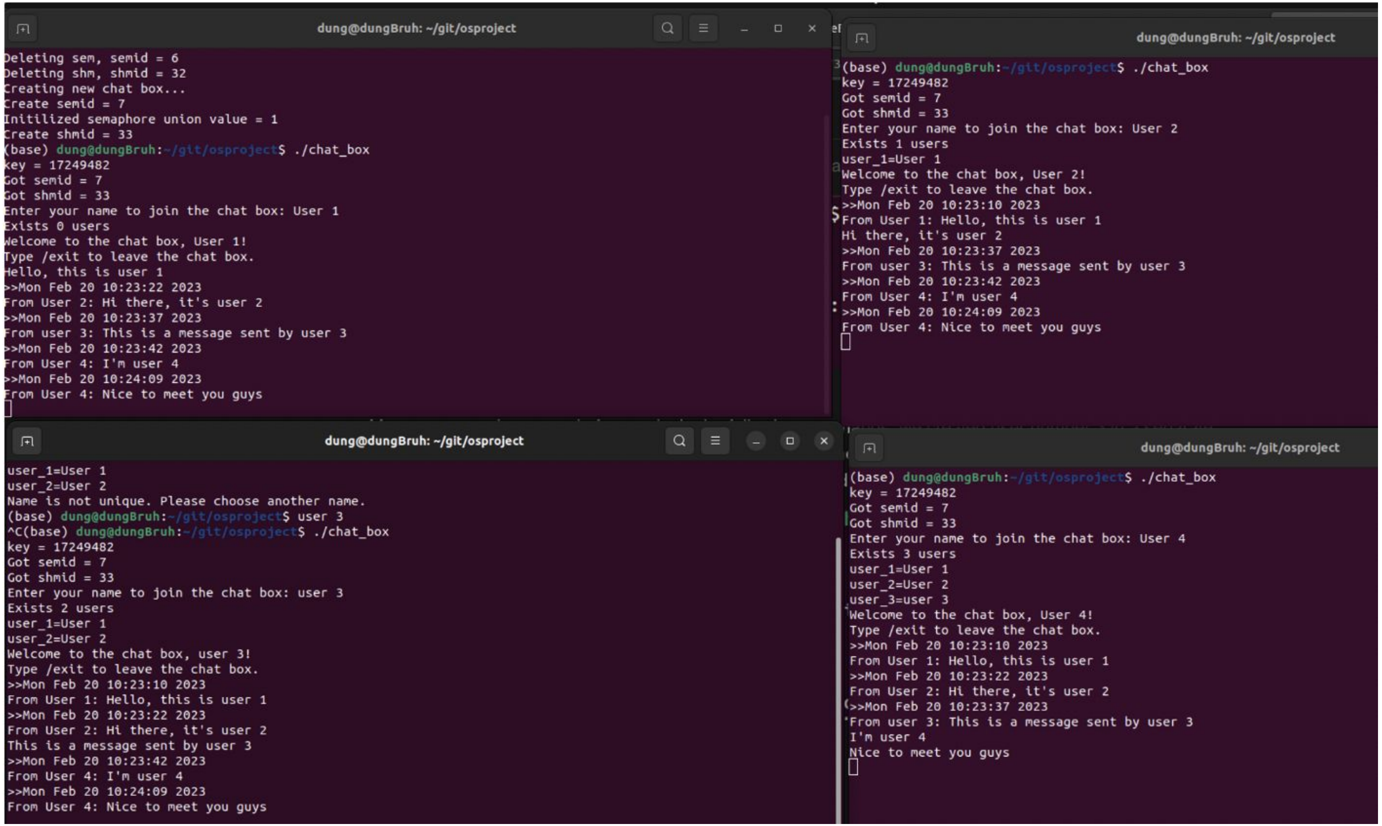
User name = `user 3`

Exists 2 users: `User 1`, `User 2`



```
key = 17249482
Got semid = 7
Got shmid = 33
Enter your name to join the chat box:
Exists 2 users
user_1=User 1
user_2=User 2
Welcome to the chat box, user 3!
```

Proof - Chat Box


dung@dungBruh: ~/git/osproject
dung@dungBruh: ~/git/osproject
dung@dungBruh: ~/git/osproject
dung@dungBruh: ~/git/osproject

```
Deleting sem, semid = 6
Deleting shm, shmid = 32
Creating new chat box...
Create semid = 7
Initilized semaphore union value = 1
Create shmid = 33
(base) dung@dungBruh:~/git/osproject$ ./chat_box
key = 17249482
Got semid = 7
Got shmid = 33
Enter your name to join the chat box: User 1
Exists 0 users
Welcome to the chat box, User 1!
Type /exit to leave the chat box.
Hello, this is user 1
>>Mon Feb 20 10:23:22 2023
From User 2: Hi there, it's user 2
>>Mon Feb 20 10:23:37 2023
From user 3: This is a message sent by user 3
>>Mon Feb 20 10:23:42 2023
From User 4: I'm user 4
>>Mon Feb 20 10:24:09 2023
From User 4: Nice to meet you guys
```

```
(base) dung@dungBruh:~/git/osproject$ ./chat_box
key = 17249482
Got semid = 7
Got shmid = 33
Enter your name to join the chat box: User 2
Exists 1 users
user_1=User 1
Welcome to the chat box, User 2!
Type /exit to leave the chat box.
>>Mon Feb 20 10:23:10 2023
From User 1: Hello, this is user 1
Hi there, it's user 2
>>Mon Feb 20 10:23:37 2023
From user 3: This is a message sent by user 3
>>Mon Feb 20 10:23:42 2023
From User 4: I'm user 4
>>Mon Feb 20 10:24:09 2023
From User 4: Nice to meet you guys
```

```
user_1=User 1
user_2=User 2
Name is not unique. Please choose another name.
(base) dung@dungBruh:~/git/osproject$ user 3
^C(base) dung@dungBruh:~/git/osproject$ ./chat_box
key = 17249482
Got semid = 7
Got shmid = 33
Enter your name to join the chat box: user 3
Exists 2 users
user_1=User 1
user_2=User 2
Welcome to the chat box, user 3!
Type /exit to leave the chat box.
>>Mon Feb 20 10:23:10 2023
From User 1: Hello, this is user 1
>>Mon Feb 20 10:23:22 2023
From User 2: Hi there, it's user 2
This is a message sent by user 3
>>Mon Feb 20 10:23:42 2023
From User 4: I'm user 4
>>Mon Feb 20 10:24:09 2023
From User 4: Nice to meet you guys
```

```
(base) dung@dungBruh:~/git/osproject$ ./chat_box
key = 17249482
Got semid = 7
Got shmid = 33
Enter your name to join the chat box: User 4
Exists 3 users
user_1=User 1
user_2=User 2
user_3=user 3
Welcome to the chat box, User 4!
Type /exit to leave the chat box.
>>Mon Feb 20 10:23:10 2023
From User 1: Hello, this is user 1
>>Mon Feb 20 10:23:22 2023
From User 2: Hi there, it's user 2
>>Mon Feb 20 10:23:37 2023
From user 3: This is a message sent by user 3
I'm user 4
Nice to meet you guys
```

Conclusion

Conclusion

- In conclusion, our group had have a deeper understanding about Shared Memory in Linux operating system
- Shared memory provides a fast and efficient mechanism for inter-process communication by allowing multiple processes to share the same region of memory
- However, it is important to note that shared memory is a low-level mechanism and requires careful management to ensure that it is used correctly and efficiently
- In addition, we implemented a simulated program that allows two process to exchange messages with inter-process synchronization technique included.

Conclusion - Future Work

- The chat box application still have some aspects to extends:
    - Add some GUI to the application for user - friendly interface
    - Add the feature of sending file through the chat box
- Maybe, we will implement this for the purpose of deploy product in the future?
- Maybe …

Thank you for listening!!!