OPERATING SYSTEM LABORATORY

LAB EXERCISE 4:

XV6 MINI PROJECT

MAHIMA S

2022115018

ABOUT XV6:

Xv6 is a simple Unix-like operating system developed by MIT as a reimplementation of the Unix Sixth Edition in order to facilitate teaching and research on operating systems. It is lightweight, easy to understand, and well-suited for educational purposes.

KEY FEATURES OF XV6:

Simplicity: Xv6 is designed to be simple and easy to understand, making it an excellent tool for teaching operating system concepts.

Modularity: The codebase is modular and well-organized, allowing for easy exploration and experimentation with different components.

Compatibility: Xv6 closely resembles Unix v6, providing a platform for studying historical Unix systems while also incorporating some modern features.

Extensibility: Xv6 can be extended and modified to implement additional features or experiment with new concepts, making it a versatile tool for research and exploration.

STEPS TO CREATE NEW USER DEFINED SYSTEM CALLS IN XV6:

- Determine the functionality and interface of the new system call.
- Implement the functionality in **syscall.c**.
- Update **syscall.h** with the system call number and any related definitions.
- Add a case statement in **syscall()** to handle the new system call.
- Declare the system call prototype in **user.h**
- Define the system call stub in assembly language in **usys.S.**
- Add the wrapper function for the system call in **syscall.c**.
- Modify user-space programs to use the new system call.
- Update the Makefile to compile the new kernel code and link it with user-space programs.
- Ensure user-space programs are recompiled after modifying the system call interface.
- Compile and run XV6 to test the new system call.

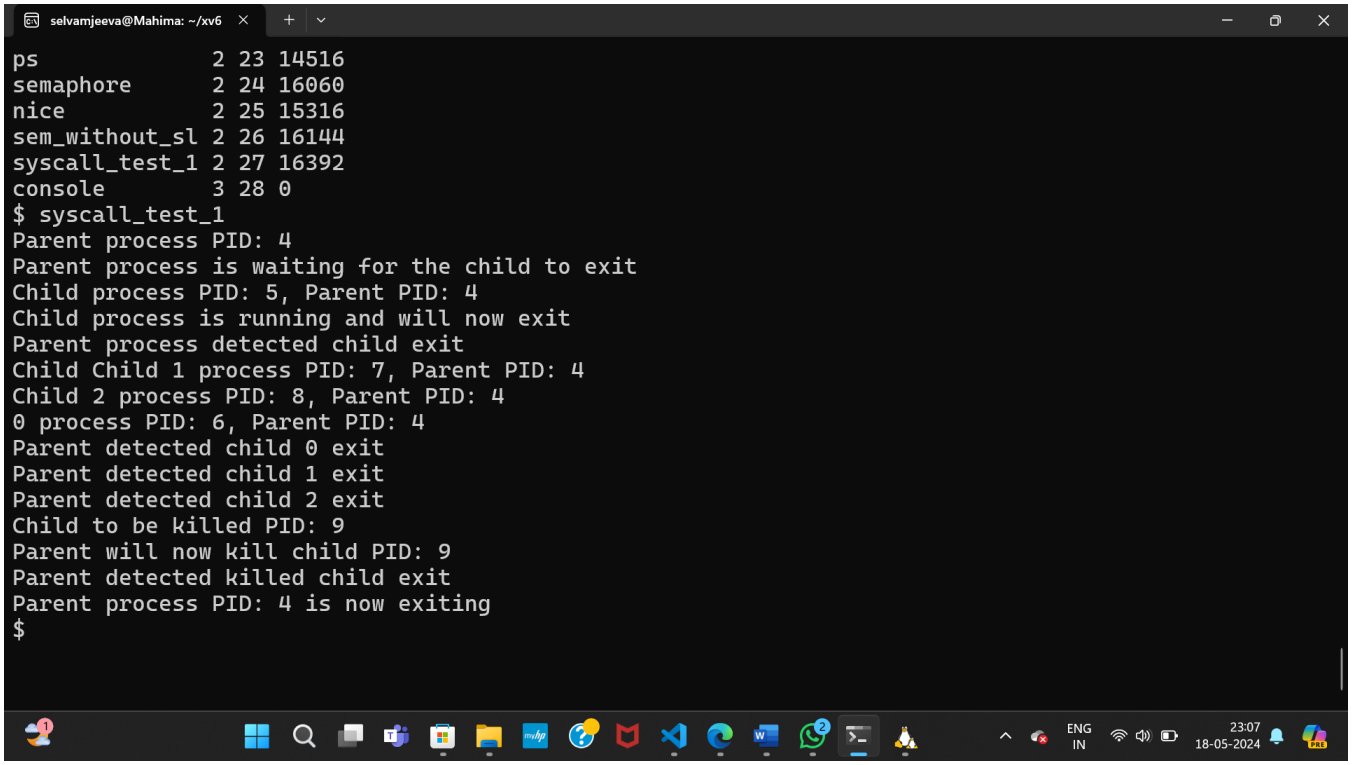
1) Write a c program that uses and test new system calls in xv6:

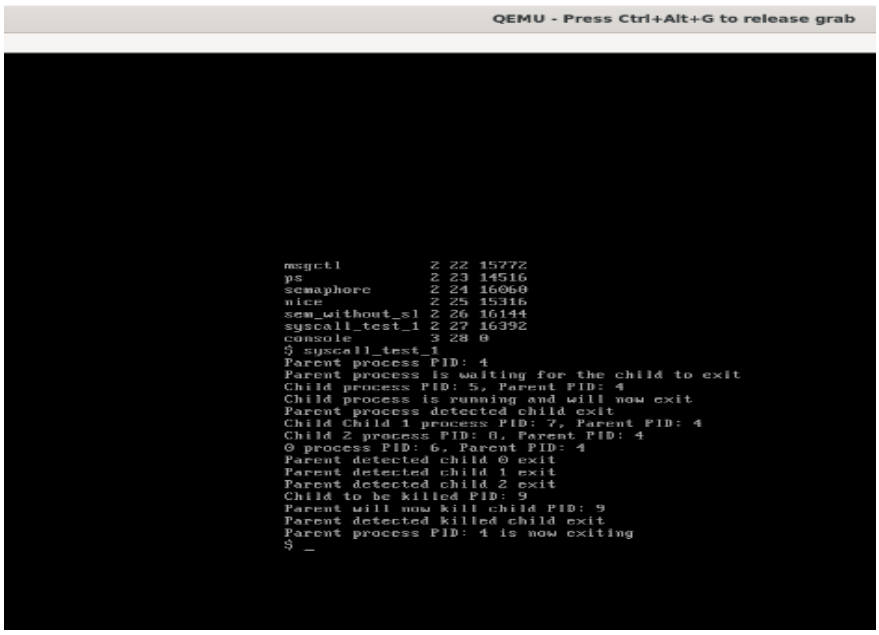
Filename: syscall_test_1.c

```c
#include "types.h"
#include "stat.h"
#include "user.h"

int main(void) {
    int pid;

    printf(1, "Parent process PID: %d\n", getpid());

    pid = fork();
    if (pid < 0) {
        printf(1, "Fork failed\n");
        exit();
    } else if (pid == 0) {
        // Child process
        printf(1, "Child process PID: %d, Parent PID: %d\n", getpid(), getppid());
        printf(1, "Child process is running and will now exit\n");
        exit();
    } else {
        // Parent process
        printf(1, "Parent process is waiting for the child to exit\n");
        wait();
        printf(1, "Parent process detected child exit\n");
    }

    // Demonstrate creating multiple child processes
    for (int i = 0; i < 3; i++) {
        pid = fork();
        if (pid < 0) {
            printf(1, "Fork failed\n");
```

```
            exit();
        } else if (pid == 0) {
            // Child process
            printf(1, "Child %d process PID: %d, Parent PID: %d\n", i, getpid(),
getppid());
            sleep(50);  // Sleep to simulate work
            exit();
        }
    }

    // Parent process waits for all children
    for (int i = 0; i < 3; i++) {
        wait();
        printf(1, "Parent detected child %d exit\n", i);
    }

    // Demonstrate killing a child process
    pid = fork();
    if (pid < 0) {
        printf(1, "Fork failed\n");
        exit();
    } else if (pid == 0) {
        // Child process
        printf(1, "Child to be killed PID: %d\n", getpid());
        sleep(100);  // Sleep to simulate work
        exit();
    } else {
        // Parent process
        sleep(10);  // Sleep a bit to ensure the child starts
        printf(1, "Parent will now kill child PID: %d\n", pid);
        kill(pid);
        wait();  // Ensure the parent waits for the killed child
        printf(1, "Parent detected killed child exit\n");
    }

    printf(1, "Parent process PID: %d is now exiting\n", getpid());
    exit();
}
```

OUTPUT :

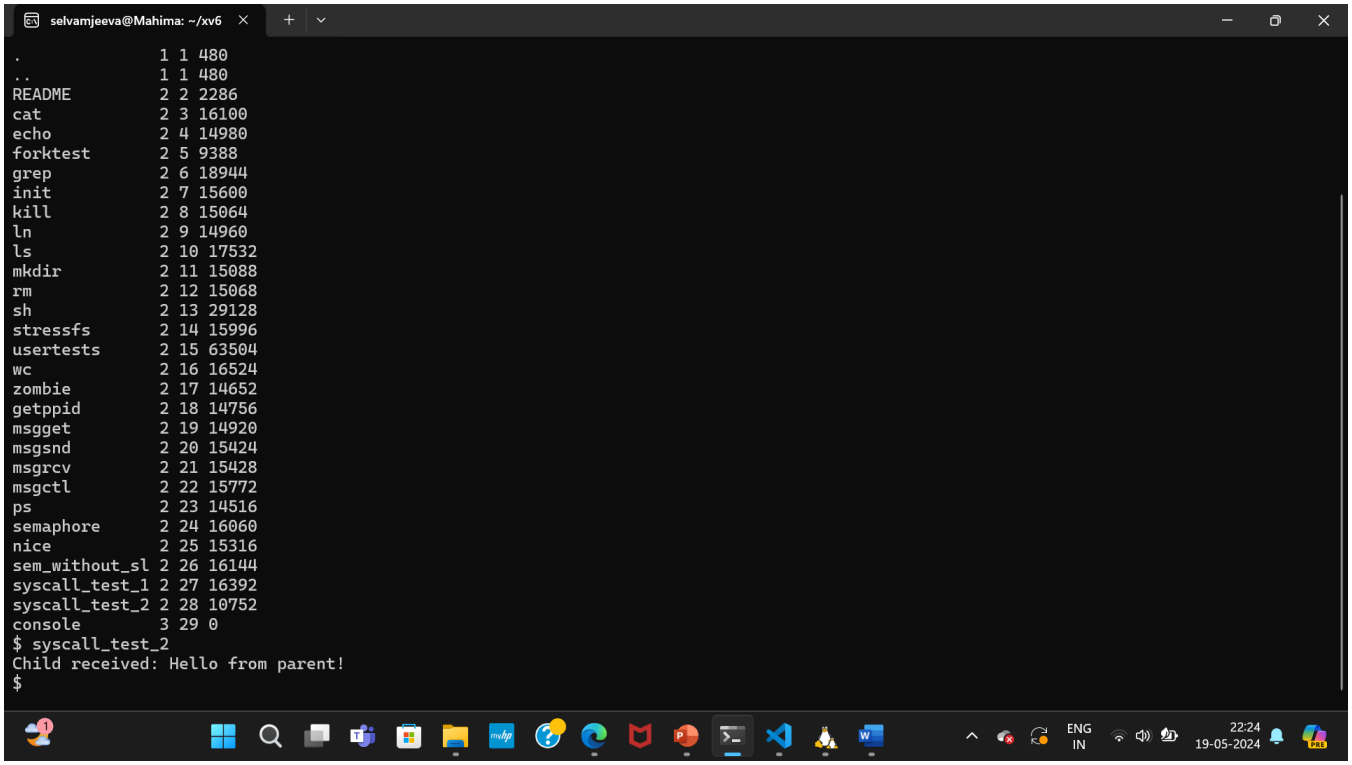

selvamjeeva@Mahima: ~/xv6
ps 2 23 14516
semaphore 2 24 16060
nice 2 25 15316
sem_without_sl 2 26 16144
syscall_test_1 2 27 16392
console 3 28 0
$ syscall_test_1
Parent process PID: 4
Parent process is waiting for the child to exit
Child process PID: 5, Parent PID: 4
Child process is running and will now exit
Parent process detected child exit
Child Child 1 process PID: 7, Parent PID: 4
Child 2 process PID: 8, Parent PID: 4
0 process PID: 6, Parent PID: 4
Parent detected child 0 exit
Parent detected child 1 exit
Parent detected child 2 exit
Child to be killed PID: 9
Parent will now kill child PID: 9
Parent detected killed child exit
Parent process PID: 4 is now exiting
$
ENG
IN
23:07
18-05-2024


QEMU - Press Ctrl+Alt+G to release grab
msgctl 2 22 15772
ps 2 23 14516
semaphore 2 24 16060
nice 2 25 15316
sem_without_sl 2 26 16144
syscall_test_1 2 27 16392
console 3 28 0
$ syscall_test_1
Parent process PID: 4
Parent process is waiting for the child to exit
Child process PID: 5, Parent PID: 4
Child process is running and will now exit
Parent process detected child exit
Child Child 1 process PID: 7, Parent PID: 4
Child 2 process PID: 8, Parent PID: 4
0 process PID: 6, Parent PID: 4
Parent detected child 0 exit
Parent detected child 1 exit
Parent detected child 2 exit
Child to be killed PID: 9
Parent will now kill child PID: 9
Parent detected killed child exit
Parent process PID: 4 is now exiting
$ _

Filename: syscall_test_2.c

```c
#include "types.h"
#include "stat.h"
#include "user.h"

int main(void) {
    int pipefd[2];
    char buf[512];
    int pid;

    if (pipe(pipefd) < 0) {
        printf(1, "Error: pipe failed\n");
        exit();
    }

    pid = fork();
    if (pid < 0) {
        printf(1, "Error: fork failed\n");
        exit();
    } else if (pid == 0) {

        close(pipefd[1]);
        read(pipefd[0], buf, sizeof(buf));
        printf(1, "Child received: %s\n", buf);
        close(pipefd[0]);
        exit();
    } else {

        close(pipefd[0]);
        char *msg = "Hello from parent!";
        write(pipefd[1], msg, strlen(msg) + 1);
        close(pipefd[1]);
        wait();
    }

    exit();
}
```

OUTPUT :


selvamjeeva@Mahima: ~/xv6
.   1 1 480
..   1 1 480
README   2 2 2286
cat   2 3 16100
echo   2 4 14980
forktest   2 5 9388
grep   2 6 18944
init   2 7 15600
kill   2 8 15064
ln   2 9 14960
ls   2 10 17532
mkdir   2 11 15088
rm   2 12 15068
sh   2 13 29128
stressfs   2 14 15996
usertests   2 15 63504
wc   2 16 16524
zombie   2 17 14652
getppid   2 18 14756
msgget   2 19 14920
msgsnd   2 20 15424
msgrcv   2 21 15428
msgctl   2 22 15772
ps   2 23 14516
semaphore   2 24 16060
nice   2 25 15316
sem_without_sl 2 26 16144
syscall_test_1 2 27 16392
syscall_test_2 2 28 10752
console   3 29 0
$ syscall_test_2
Child received: Hello from parent!
$
ENG
IN
22:24
19-05-2024

2)Minimum of 5 system calls to be created and tested in XV6:

1. **getppid() :** return the pid of parent

```
#include "types.h"
#include "x86.h"
#include "defs.h"
#include "date.h"
#include "param.h"
#include "memlayout.h"
#include "mmu.h"
#include "proc.h"
#include "spinlock.h"
#include "msg.h"
#include "sem.h"
int
sys_getppid(void)
{
    return myproc()->parent ? myproc()->parent->pid : myproc()->pid;
}
```

getppid.c

```
#include "user.h"

#include "types.h"

#include "stat.h"

#include "fcntl.h"

int main(void) {

   int ppid = getppid();

   printf(1, "Parent PID: %d\n", ppid);

   exit();}
```

2)nice() system call : which change the process priority

```
#include "types.h"
#include "x86.h"
#include "defs.h"
#include "date.h"
#include "param.h"
#include "memlayout.h"
#include "mmu.h"
#include "proc.h"
#include "spinlock.h"
#include "msg.h"
#include "sem.h"

int
sys_nice(void)
{
  int inc;
  if(argint(0, &inc) < 0)
    return -1;

  struct proc *curproc = myproc();

  if (inc < 0 || inc > 40) {
    return -1;
  }

  curproc->quantum += inc;

  return 0;
}
```

nice.c

#include "types.h"

#include "user.h"

Int main(int argc, char *argv[]){

if (argc < 2) {

```
        printf(2, "Usage: %s <priority_increment>\n", argv[0]);
        exit();
    }

    int priority_increment = atoi(argv[1]); // Convert argument to integer
    if (priority_increment < 0 || priority_increment > 40) {
        printf(2, "Invalid priority increment. Please provide a value between 0 and 40.\n");
        exit();
    }

    if (nice(priority_increment) < 0) {
        printf(2, "Failed to adjust process priority.\n");
        exit();
    }

    printf(1, "Process priority adjusted successfully.\n");
    exit();
}
```

msg.h :

```
#ifndef _MSG_QUEUE_H_
#define _MSG_QUEUE_H_

#include "types.h"

// IPC permissions structure
struct ipc_perm {
    key__t key;
    uid__t uid;
```

```
    gid__t gid;
    uid__t cuid;
    gid__t cgid;
    unsigned short mode;
};

// Message queue data structure
struct msqid_ds {
    struct ipc_perm msg_perm;
    uint msg_qnum;
    uint msg_qbytes;
    pid__t msg_lspid;
    pid__t msg_lrpid;
    time__t msg_stime;
    time__t msg_rtime;
    time__t msg_ctime;
};

#endif // _MSG_QUEUE_H_
```

3)msgget() :

```
#include "types.h"
#include "x86.h"
#include "defs.h"
#include "date.h"
#include "param.h"
#include "memlayout.h"
#include "mmu.h"
#include "proc.h"
#include "spinlock.h"
#include "msg.h"
#include "sem.h"

#define BUF_SIZE 512
#define MSGMAX 10
#define MSGQMAX 10
```

```c
static struct msgq msg_queues[MSGQMAX];

static struct spinlock msgq_lock;

void init_msg_queue(void) {
    initlock(&msgq_lock, "msgq_lock");
    for (int i = 0; i < MSGQMAX; i++) {
        msg_queues[i].key = 0;
        msg_queues[i].start = 0;
        msg_queues[i].end = 0;
    }
}

int sys_msgget(void) {
    int key, msgflg;

    if (argint(0, &key) < 0 || argint(1, &msgflg) < 0)
        return -1;

    acquire(&msgq_lock);
    for (int i = 0; i < MSGQMAX; i++) {
        if (msg_queues[i].key == key) {
            release(&msgq_lock);
            return i;
        }
    }
    for (int i = 0; i < MSGQMAX; i++) {
        if (msg_queues[i].key == 0) {
            msg_queues[i].key = key;
            release(&msgq_lock);
            return i;
        }
    }
    release(&msgq_lock);
    return -1;
}
```

msgget.c

```c
#include "types.h"

#include "stat.h"

#include "user.h"
```

```
#define IPC_CREAT 01000


int main(void) {
   int msgqid = msgget(123, 0666 | IPC_CREAT);
   if (msgqid < 0) {
      printf(1, "Failed to create/get message queue\n");
   } else {
      printf(1, "Message queue ID: %d\n", msgqid);
   }
   exit();
}
```

4) msgsnd() :

```
#include "types.h"
#include "x86.h"
#include "defs.h"
#include "date.h"
#include "param.h"
#include "memlayout.h"
#include "mmu.h"
#include "proc.h"
#include "spinlock.h"
#include "msg.h"
#include "sem.h"

#define BUF_SIZE 512
#define MSGMAX 10
#define MSGQMAX 10
```

```
int sys_msgsnd(void) {

  int msqid;
  struct msgbuf *msgp;
  int msgsz, msgflg;
```

```
    if (argint(0, &msqid) < 0 || argptr(1, (void*)&msgp, sizeof(struct msgbuf)) < 0 ||
        argint(2, &msgsz) < 0 || argint(3, &msgflg) < 0)
        return -1;

    acquire(&msgq_lock);
    if (msqid < 0 || msqid >= MSGQMAX || msg_queues[msqid].key == 0) {
        release(&msgq_lock);
        return -1;
    }

    if (msg_queues[msqid].end - msg_queues[msqid].start >= MSGMAX) {
        release(&msgq_lock);
        return -1;
    }

    msg_queues[msqid].messages[msg_queues[msqid].end % MSGMAX] = *msgp;
    msg_queues[msqid].end++;
    release(&msgq_lock);
    return 0;
}
```

msgsnd.c :

```
#include "types.h"

#include "stat.h"

#include "user.h"


struct mess {

   long mtype;

   char mtext[100];

};


#define IPC_CREAT 01000
```

```
int main(void) {
    int msgqid = msgget(123, 0666 | IPC_CREAT);
    if (msgqid < 0) {
        printf(1, "Failed to create/get message queue\n");
        exit();
    }

    struct mess msg;
    msg.mtype = 1;
    strcpy(msg.mtext, "Hello, xv6!");

    if (msgsnd(msgqid, &msg, sizeof(msg.mtext), 0) < 0) {
        printf(1, "Failed to send message\n");
    } else {
        printf(1, "Message sent\n");
    }
    exit();
}
```

5) msgrcv () :

```
int sys_msgrcv(void) {
    int msqid;
    struct msgbuf *msgp;
    int msgsz, msgtyp, msgflg;

    if (argint(0, &msqid) < 0 || argptr(1, (void*)&msgp, sizeof(struct msgbuf)) < 0 ||
```

```
        argint(2, &msgsz) < 0 || argint(3, &msgtyp) < 0 || argint(4, &msgflg) < 0)
        return -1;

    acquire(&msgq_lock);
    if (msqid < 0 || msqid >= MSGQMAX || msg_queues[msqid].key == 0) {
        release(&msgq_lock);
        return -1;
    }

    if (msg_queues[msqid].start == msg_queues[msqid].end) {
        release(&msgq_lock);
        return -1;
    }

    *msgp = msg_queues[msqid].messages[msg_queues[msqid].start % MSGMAX];
    msg_queues[msqid].start++;
    release(&msgq_lock);
    return 0;
}
```

msgrcv.c

```
#include "types.h"
#include "stat.h"
#include "user.h"

#define IPC_CREAT 01000
struct mess {
    long mtype;
    char mtext[100];
};

int main(void) {
    int msgqid = msgget(123, 0666 | IPC_CREAT);
```

```
    if (msgqid < 0) {
        printf(1, "Failed to create/get message queue\n");
        exit();
    }

    struct mess msg;
    memset(&msg, 0, sizeof(msg));

    if (msgrcv(msgqid, &msg, sizeof(msg.mtext), 1, 0) < 0) {
        printf(1, "Failed to receive message\n");
    } else {
        printf(1, "Received message: %s\n", msg.mtext);
    }
    exit();
}
```

6. msgctl() :

```
int sys_msgctl(void) {
    int msqid, cmd;
    struct msqid_ds *buf;

    if (argint(0, &msqid) < 0 || argint(1, &cmd) < 0 || argptr(2, (void*)&buf,
sizeof(struct msqid_ds)) < 0)
        return -1;

    acquire(&msgq_lock);
    if (msqid < 0 || msqid >= MSGQMAX || msg_queues[msqid].key == 0) {
        release(&msgq_lock);
        return -1;
    }
```

```
    switch (cmd) {
    case IPC_RMID:
        msg_queues[msqid].key = 0;
        msg_queues[msqid].start = 0;
        msg_queues[msqid].end = 0;
        break;
    case IPC_STAT:
        buf->msg_perm.key = msg_queues[msqid].key;
        buf->msg_qnum = msg_queues[msqid].end - msg_queues[msqid].start;
        buf->msg_qbytes = sizeof(struct msgbuf) * MSGMAX;
        buf->msg_lspid = -1;
        buf->msg_lrpid = -1;
        buf->msg_stime = 0;
        buf->msg_rtime = 0;
        buf->msg_ctime = 0;
        break;
    case IPC_SET:

        release(&msgq_lock);
        return -1;
    default:
        release(&msgq_lock);
        return -1;
    }

    release(&msgq_lock);
    return 0;
}
```

msgctl.c

#include "types.h"

#include "stat.h"

#include "user.h"

#include "msg.h"

#define IPC_STAT 0

```
int main(void) {
    int msgqid;
    struct msqid_ds buf;

    msgqid = msgget(123, 0666);
    if (msgqid < 0) {
        printf(1, "Failed to get message queue\n");
        exit();
    }

    if (msgctl(msgqid, IPC_STAT, &buf) < 0) {
        printf(1, "Failed to get message queue status\n");
    } else {
        printf(1, "Message queue status:\n");
        printf(1, "  msg_qnum: %d\n", buf.msg_qnum);
        printf(1, "  msg_qbytes: %d\n", buf.msg_qbytes);
        printf(1, "  msg_lspid: %d\n", buf.msg_lspid);
        printf(1, "  msg_lrpid: %d\n", buf.msg_lrpid);
    }

    exit();
}
```

7) ps() : system call

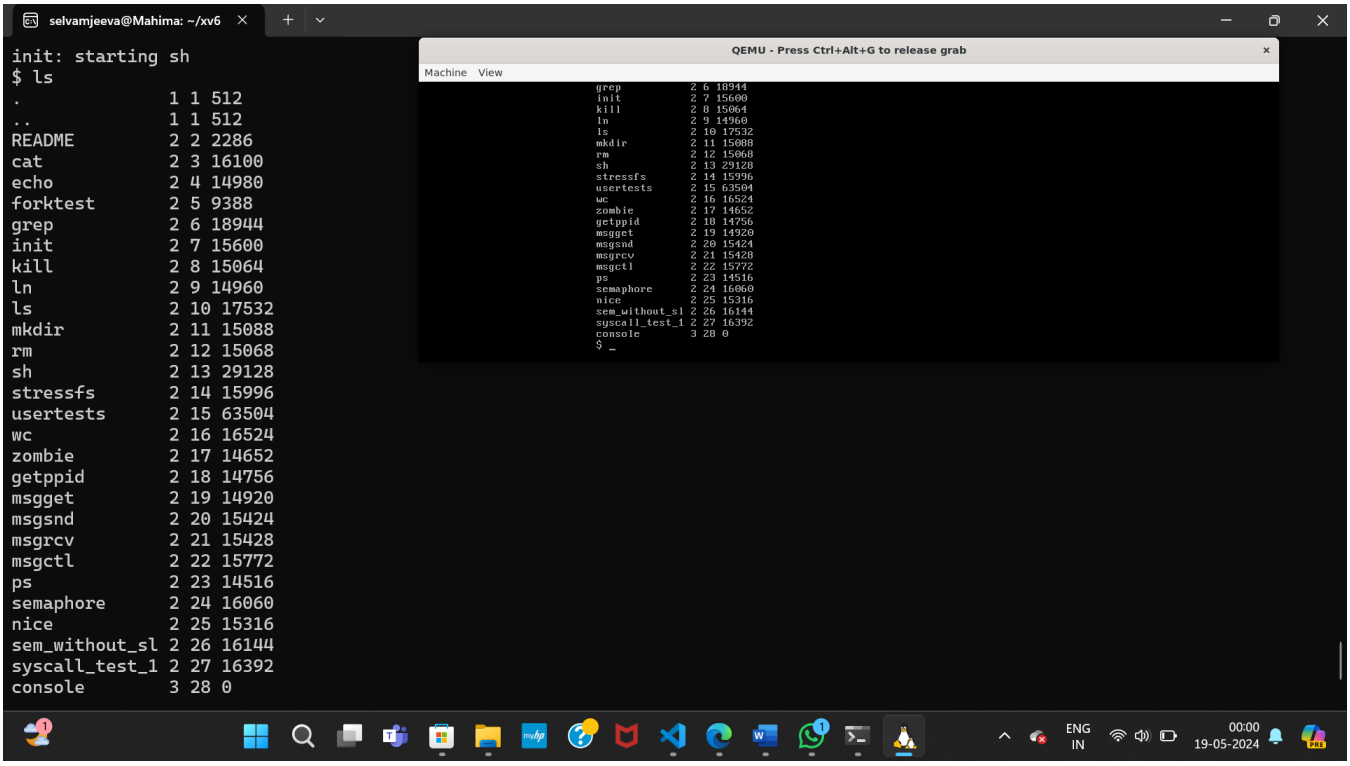
```
#include "types.h"
#include "x86.h"
#include "defs.h"
#include "date.h"
#include "param.h"
#include "memlayout.h"
#include "mmu.h"
#include "proc.h"
#include "spinlock.h"
#include "msg.h"
#include "sem.h"

void sys_ps(void) {
    struct proc *p;

    cprintf("PID\tName\tState\n");

    for (p = proc; p < &proc[NPROC]; p++) {

        cprintf("%d\t%s\t", p->pid, p->name);
        switch(p->state) {
            case UNUSED:
                cprintf("UNUSED\n");
                break;
            case EMBRYO:
                cprintf("EMBRYO\n");
                break;
            case SLEEPING:
                cprintf("SLEEPING\n");
                break;
            case RUNNABLE:
                cprintf("RUNNABLE\n");
                break;
            case RUNNING:
                cprintf("RUNNING\n");
                break;
            case ZOMBIE:
                cprintf("ZOMBIE\n");
                break;
            default:
                cprintf("UNKNOWN\n");
```

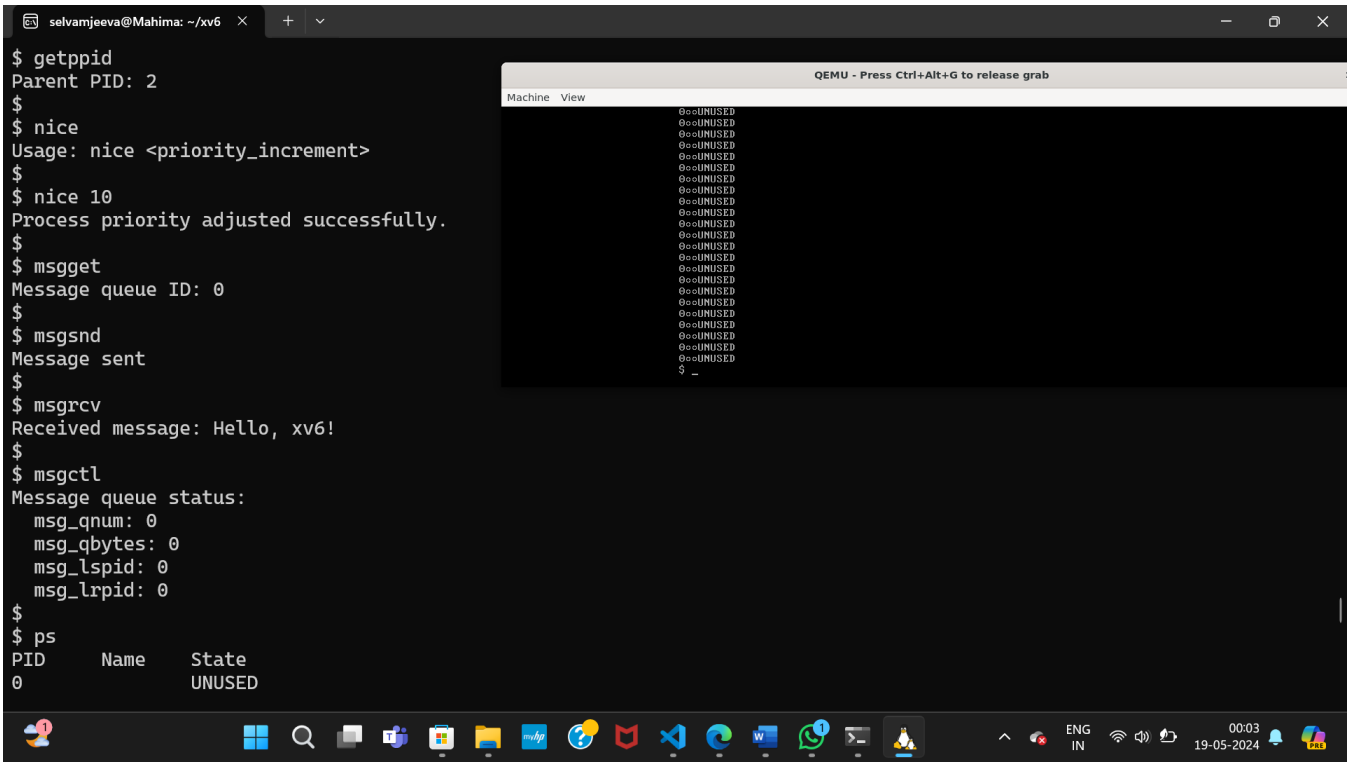
```
                break;
        }
    }
}
```

ps.c

```
#include "types.h"

#include "stat.h"

#include "user.h"


int

main(void)

{

   ps();

   exit();

}
```

OUTPUT:


selvamjeeva@Mahima: ~/xv6
init: starting sh
$ ls
.   1 1 512
..   1 1 512
README   2 2 2286
cat   2 3 16100
echo   2 4 14980
forktest   2 5 9388
grep   2 6 18944
init   2 7 15600
kill   2 8 15064
ln   2 9 14960
ls   2 10 17532
mkdir   2 11 15088
rm   2 12 15068
sh   2 13 29128
stressfs   2 14 15996
usertests   2 15 63504
wc   2 16 16524
zombie   2 17 14652
getppid   2 18 14756
msgget   2 19 14920
msgsnd   2 20 15424
msgrcv   2 21 15428
msgctl   2 22 15772
ps   2 23 14516
semaphore   2 24 16060
nice   2 25 15316
sem_without_sl 2 26 16144
syscall_test_1 2 27 16392
console   3 28 0
QEMU - Press Ctrl+Alt+G to release grab
Machine View
grep 2 6 18944
init 2 7 15600
kill 2 8 15064
ln 2 9 14960
ls 2 10 17532
mkdir 2 11 15088
rm 2 12 15068
sh 2 13 29128
stressfs 2 14 15996
usertests 2 15 63504
wc 2 16 16524
zombie 2 17 14652
getppid 2 18 14756
msgget 2 19 14920
msgsnd 2 20 15424
msgrcv 2 21 15428
msgctl 2 22 15772
ps 2 23 14516
semaphore 2 24 16060
nice 2 25 15316
sem_without_sl 2 26 16144
syscall_test_1 2 27 16392
console 3 28 0
$ _
ENG IN
00:00
19-05-2024


selvamjeeva@Mahima: ~/xv6
$ getppid
Parent PID: 2
$
$ nice
Usage: nice <priority_increment>
$
$ nice 10
Process priority adjusted successfully.
$
$ msgget
Message queue ID: 0
$
$ msgsnd
Message sent
$
$ msgrcv
Received message: Hello, xv6!
$
$ msgctl
Message queue status:
  msg_qnum: 0
  msg_qbytes: 0
  msg_lspid: 0
  msg_lrpid: 0
$
$ ps
PID   Name   State
0   UNUSED
QEMU - Press Ctrl+Alt+G to release grab
Machine View
0 <> UNUSED
$ _
ENG IN
00:03
19-05-2024

3) Implement semaphore system calls in xv6 with and with out using sleep and wakeup system calls:

SEMAPHORE:

sem.h :

```
#ifndef _SEMAPHORE_H_
#define _SEMAPHORE_H_

#include "spinlock.h"

#define MAX_SEMAPHORES 10


struct semaphore {
    int value;
    int active;
    int waiting_count;
    int waiting_processes[20];
    struct spinlock lock;
};

#endif // _SEMAPHORE_H
```

1) With using sleep() and wakeup() system calls:

```
#include "types.h"
#include "x86.h"
#include "defs.h"
#include "date.h"
#include "param.h"
```

```c
#include "memlayout.h"
#include "mmu.h"
#include "proc.h"
#include "spinlock.h"
#include "msg.h"
#include "sem.h"

#define MAX_SEMAPHORES 10

//semaphore

struct semaphore semaphores[MAX_SEMAPHORES];


int sys_sem_init(void) {
    int value;
    if (argint(0, &value) < 0)
        return -1;

    int i;
    for (i = 0; i < MAX_SEMAPHORES; i++) {
        if (!semaphores[i].active) {
            semaphores[i].value = value;
            semaphores[i].active = 1;
            initlock(&semaphores[i].lock, "semaphore");
            return i; // Return semaphore ID
        }
    }
    return -1;
}

int sys_sem_wait(void) {
    int sem_id;
    if (argint(0, &sem_id) < 0)
        return -1;

    if (sem_id < 0 || sem_id >= MAX_SEMAPHORES || !semaphores[sem_id].active)
        return -1; // Invalid semaphore ID

    acquire(&semaphores[sem_id].lock);
    semaphores[sem_id].value--;
    if (semaphores[sem_id].value < 0) {

        sleep(myproc(), &semaphores[sem_id].lock);
    }
```

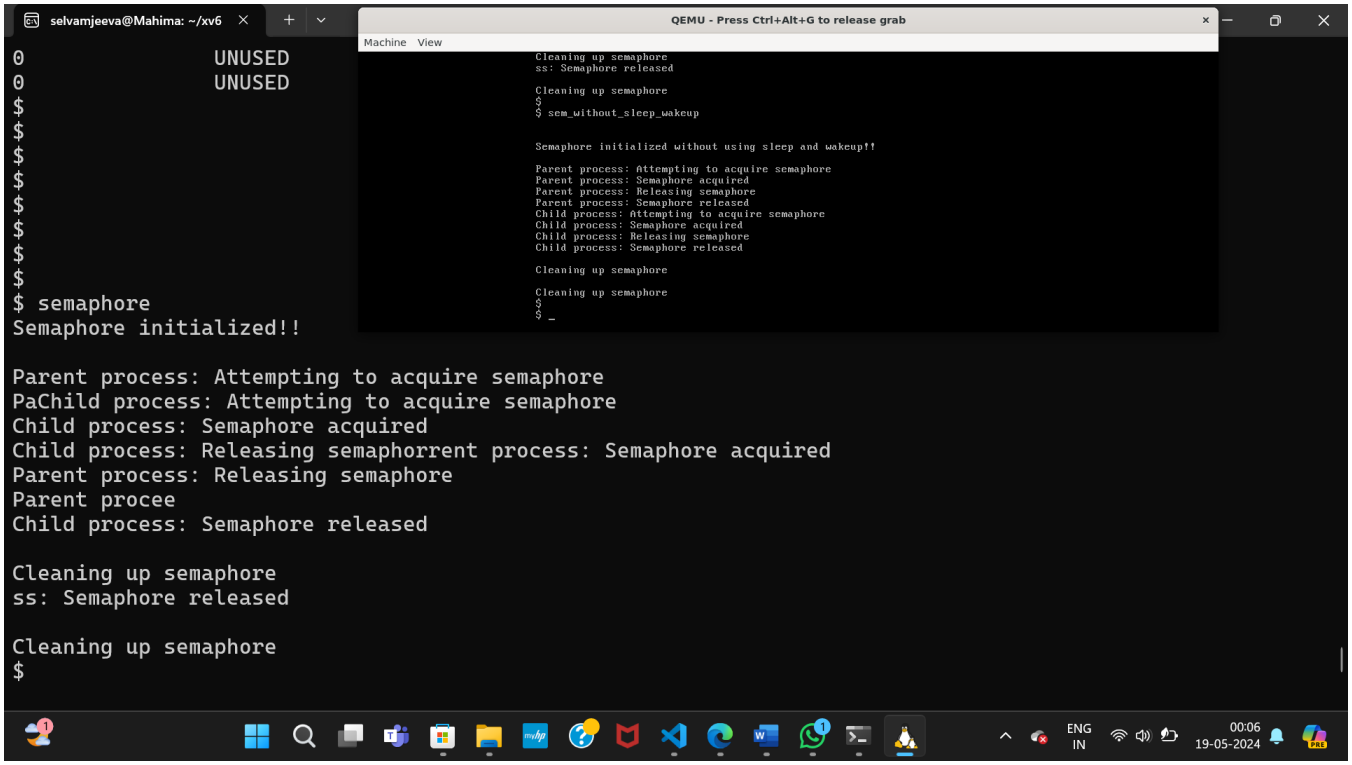
```
    release(&semaphores[sem_id].lock);
    return 0;
}

int sys_sem_signal(void) {
    int sem_id;
    if (argint(0, &sem_id) < 0)
        return -1;

    if (sem_id < 0 || sem_id >= MAX_SEMAPHORES || !semaphores[sem_id].active)
        return -1; // Invalid semaphore ID

    acquire(&semaphores[sem_id].lock);
    semaphores[sem_id].value++;
    if (semaphores[sem_id].value <= 0) {

        wakeup(&semaphores[sem_id]);
    }
    release(&semaphores[sem_id].lock);
    return 0;
}

int
sys_sem_destroy(void)
{
    int sem_id;
    if (argint(0, &sem_id) < 0)
        return -1;

    return sem_destroy(sem_id);
}

int sem_destroy(int sem_id) {
    if (sem_id < 0 || sem_id >= MAX_SEMAPHORES || !semaphores[sem_id].active)
        return -1;

    acquire(&semaphores[sem_id].lock);
    semaphores[sem_id].active = 0;
    release(&semaphores[sem_id].lock);

    return 0;
}
```

semaphore.c

```
#include "types.h"
#include "user.h"

#define SEM_VALUE 3

int main(int argc, char *argv[]) {
    // Initialize a semaphore with an initial value
    int sem_id = sem_init(SEM_VALUE);
    if (sem_id < 0) {
        printf(1, "Semaphore initialization failed\n");
        exit();
    }
    printf(1,"Semaphore initialized!!\n");

    // Fork child processes to test semaphore wait and signal
    int pid = fork();
    if (pid < 0) {
        printf(1, "Fork failed\n");
        exit();
    } else if (pid == 0) {
        // Child process
        printf(1, "Child process: Attempting to acquire semaphore\n");
        sem_wait(sem_id);
```

```
        printf(1, "Child process: Semaphore acquired\n");
        printf(1, "Child process: Releasing semaphore\n");
        sem_signal(sem_id);
        printf(1, "Child process: Semaphore released\n");
    } else {
        // Parent process
        printf(1, "\nParent process: Attempting to acquire semaphore\n");
        sem_wait(sem_id);
        printf(1, "Parent process: Semaphore acquired\n");
        printf(1, "Parent process: Releasing semaphore\n");
        sem_signal(sem_id);
        printf(1, "Parent process: Semaphore released\n");
        wait();
    }

    // Clean up the semaphore
    printf(1, "\nCleaning up semaphore\n");
    sem_destroy(sem_id);

    exit();
}
```

OUTPUT:



2 . Without using sleep() and wakeup() system calls:

```c
#include "types.h"
#include "x86.h"
#include "defs.h"
#include "date.h"
#include "param.h"
#include "memlayout.h"
#include "mmu.h"
#include "proc.h"
#include "spinlock.h"
#include "msg.h"
#include "sem.h"

#define MAX_SEMAPHORES 10

//semaphore without sleep and wakeup calls
```

```
int sys_sem_init_new(void) {
    int value;
    struct semaphore *sem;
    if (argint(0, &value) < 0 || value < 0)
        return -1;
    sem = (struct semaphore*)kalloc();
    if (sem == 0)
        return -1;
    sem->value = value;
    sem->active=1;
    initlock(&sem->lock, "sem_lock");

    return (int)sem;
}
int sys_sem_wait_new(void) {
    struct semaphore *sem;
    if (argint(0, (int*)&sem) < 0)
        return -1;
    acquire(&sem->lock);
    while (sem->value <= 0) {
        release(&sem->lock);
        yield();  // Yield the CPU to allow other processes to run
        acquire(&sem->lock);
    }
    sem->value--;
    release(&sem->lock);
    return 0;
}

int sys_sem_signal_new(void) {
    struct semaphore *sem;

    if (argint(0, (int*)&sem) < 0)
        return -1;

    acquire(&sem->lock);
    sem->value++;
    release(&sem->lock);

    return 0;
}

int sys_sem_destroy_new(void) {
    struct semaphore *sem;
```

```
    if (argint(0, (int*)&sem) < 0)
        return -1;
    sem->active=0;
    kfree((char*)sem);

    return 0;
}
```

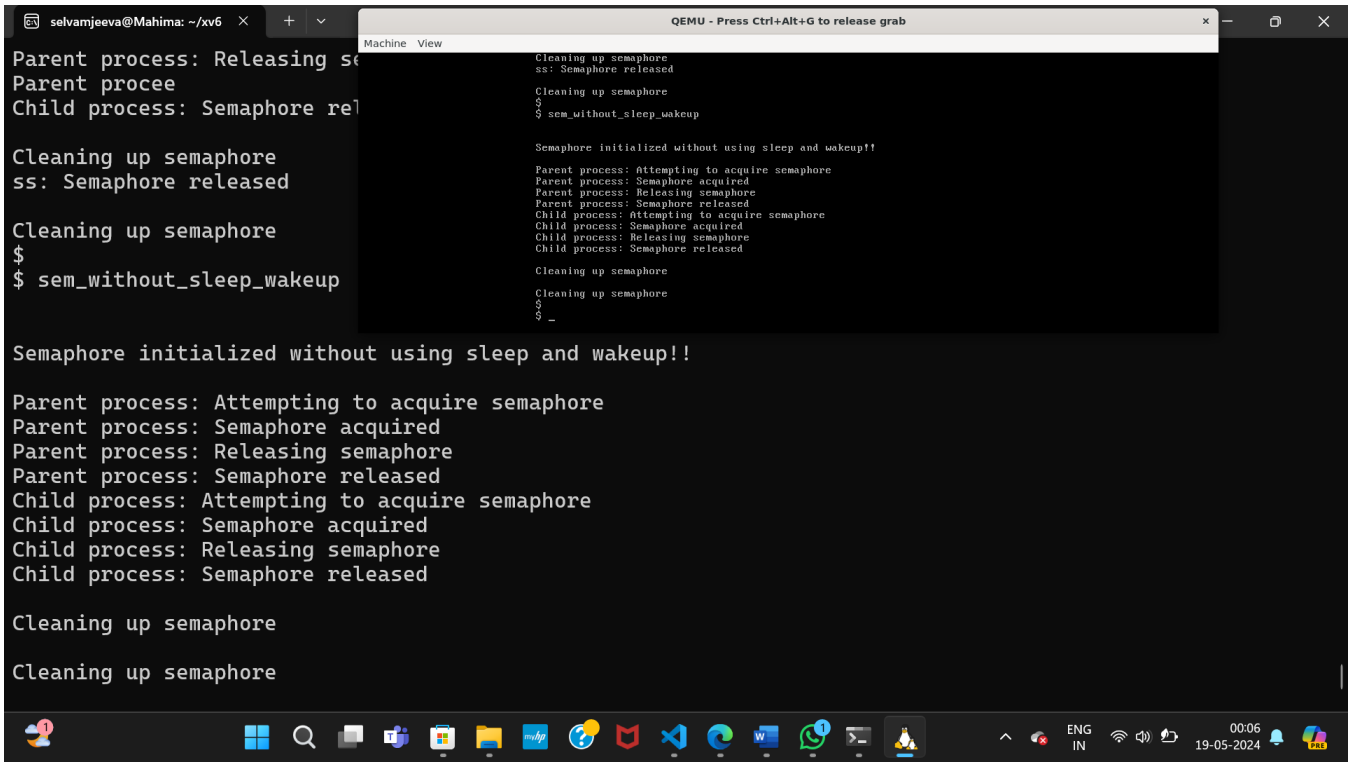
sem_without_sleep_wakeup.c

```
#include "types.h"
#include "user.h"



#define SEM_VALUE 3


int main(int argc, char *argv[]) {
   int sem_id = sem_init_new(SEM_VALUE);
   if (sem_id ==-1) {
      printf(1, "Semaphore initialization failed\n");
      exit();
   }
   printf(1, "\n\nSemaphore initialized without using sleep and wakeup!!\n");



   int pid = fork();
```

```
if (pid < 0) {
    printf(1, "Fork failed\n");
    exit();
} else if (pid == 0) {

    printf(1, "Child process: Attempting to acquire semaphore\n");
    sem_wait_new(sem_id);
    printf(1, "Child process: Semaphore acquired\n");
    printf(1, "Child process: Releasing semaphore\n");
    sem_signal_new(sem_id);
    printf(1, "Child process: Semaphore released\n");
} else {

    printf(1, "\nParent process: Attempting to acquire semaphore\n");
    sem_wait_new(sem_id);
    printf(1, "Parent process: Semaphore acquired\n");
    printf(1, "Parent process: Releasing semaphore\n");
    sem_signal_new(sem_id);
    printf(1, "Parent process: Semaphore released\n");
    wait();
}


printf(1, "\nCleaning up semaphore\n");
sem_destroy_new(sem_id);
```

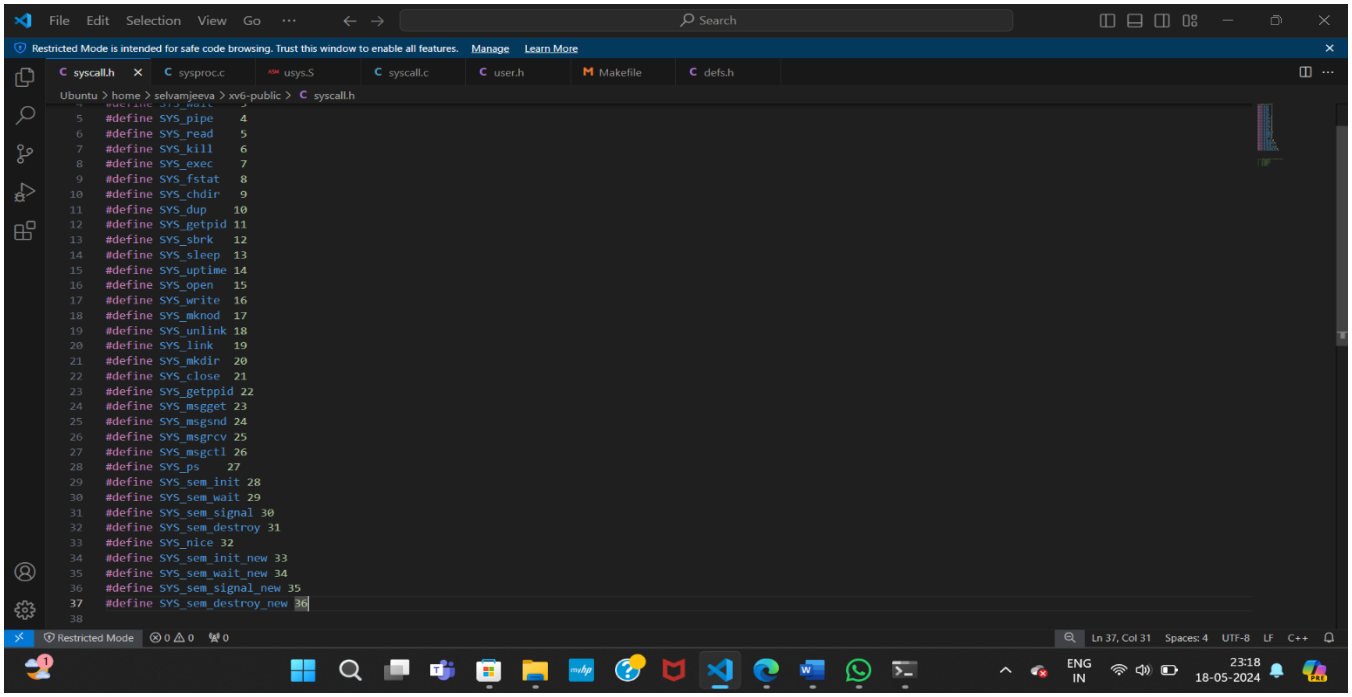
```
    exit();
}
```

OUTPUT:

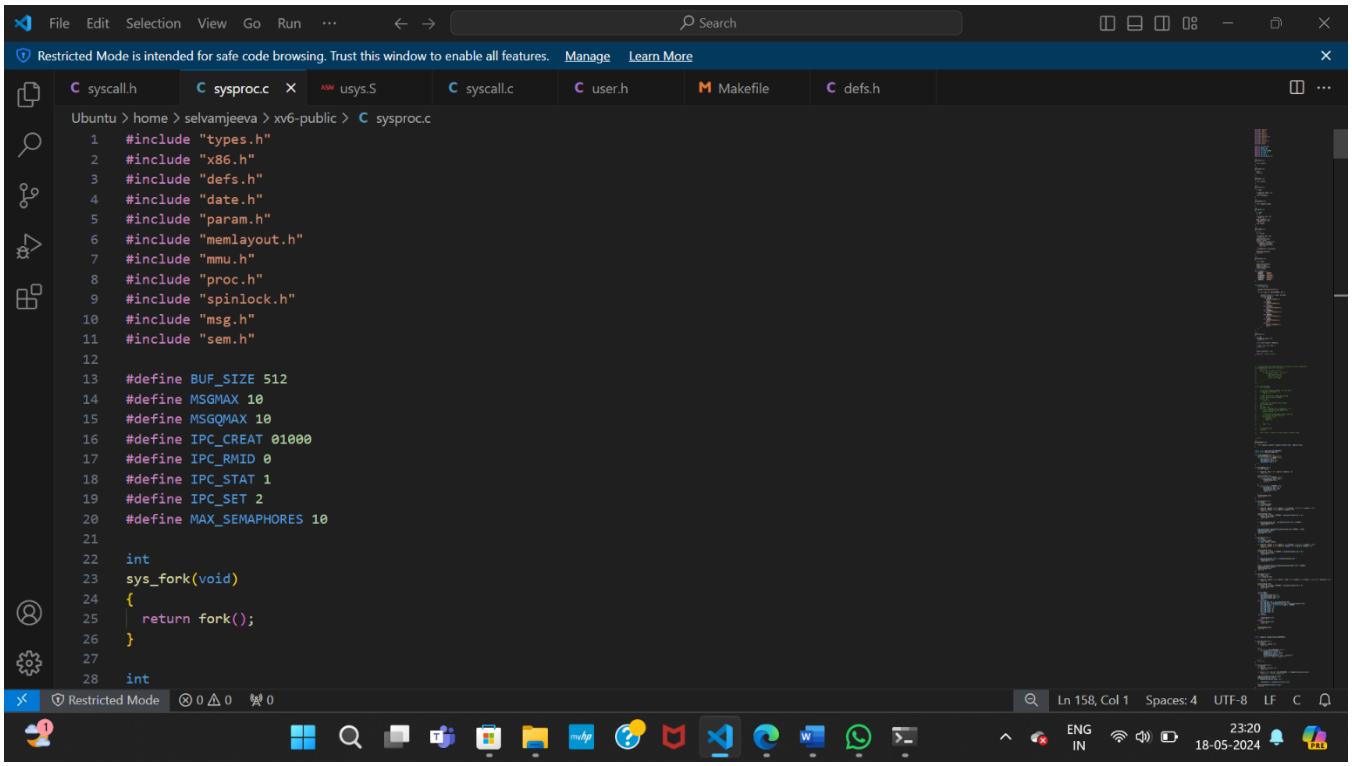

selvamjeevera@Mahima: ~/xv6
QEMU - Press Ctrl+Alt+G to release grab
Machine View
Parent process: Releasing se
Parent procee
Child process: Semaphore rel
Cleaning up semaphore
ss: Semaphore released
Cleaning up semaphore
$
$ sem_without_sleep_wakeup
Semaphore initialized without using sleep and wakeup!!
Parent process: Attempting to acquire semaphore
Parent process: Semaphore acquired
Parent process: Releasing semaphore
Parent process: Semaphore released
Child process: Attempting to acquire semaphore
Child process: Semaphore acquired
Child process: Releasing semaphore
Child process: Semaphore released
Cleaning up semaphore
Cleaning up semaphore
$
$ _
Semaphore initialized without using sleep and wakeup!!
Parent process: Attempting to acquire semaphore
Parent process: Semaphore acquired
Parent process: Releasing semaphore
Parent process: Semaphore released
Child process: Attempting to acquire semaphore
Child process: Semaphore acquired
Child process: Releasing semaphore
Child process: Semaphore released
Cleaning up semaphore
Cleaning up semaphore
ENG
IN
00:06
19-05-2024

XV6 – MODIFIED FILE IMAGES :

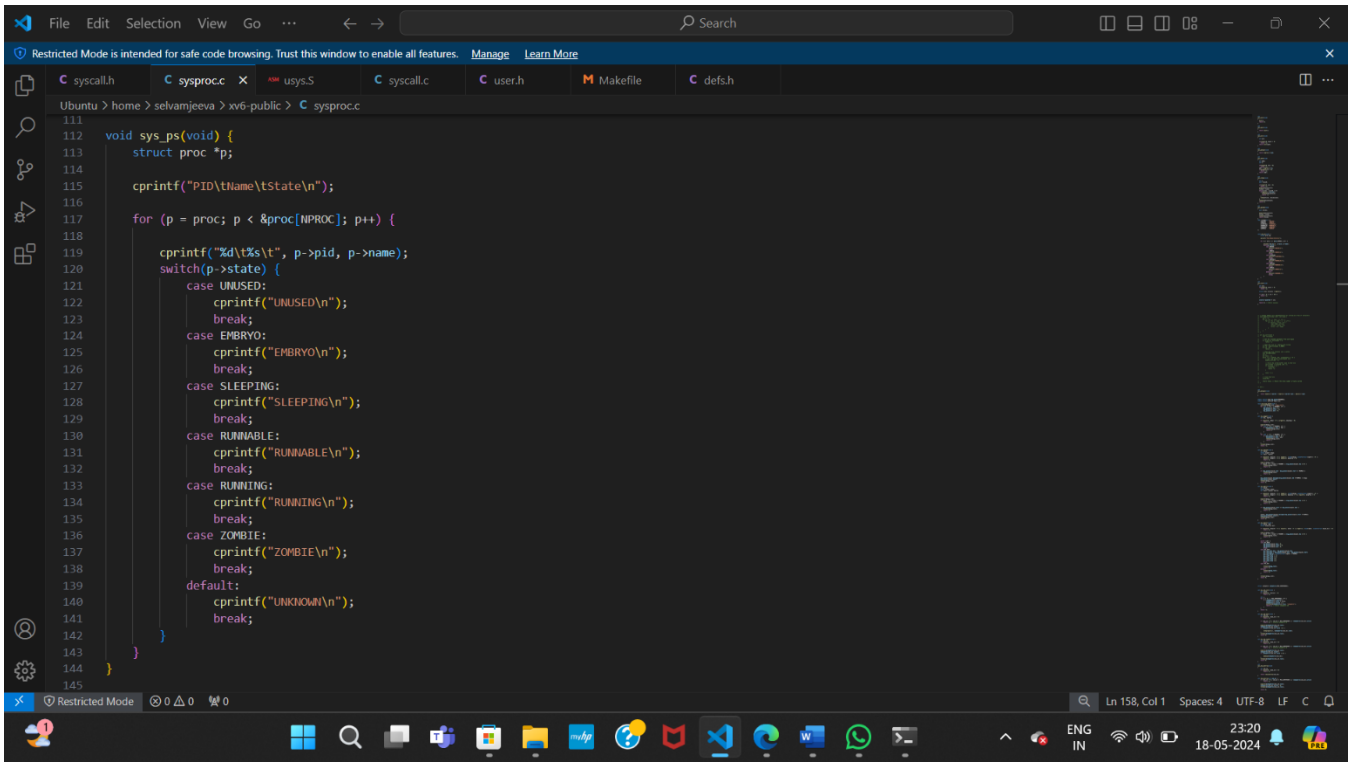
Syscall.h :

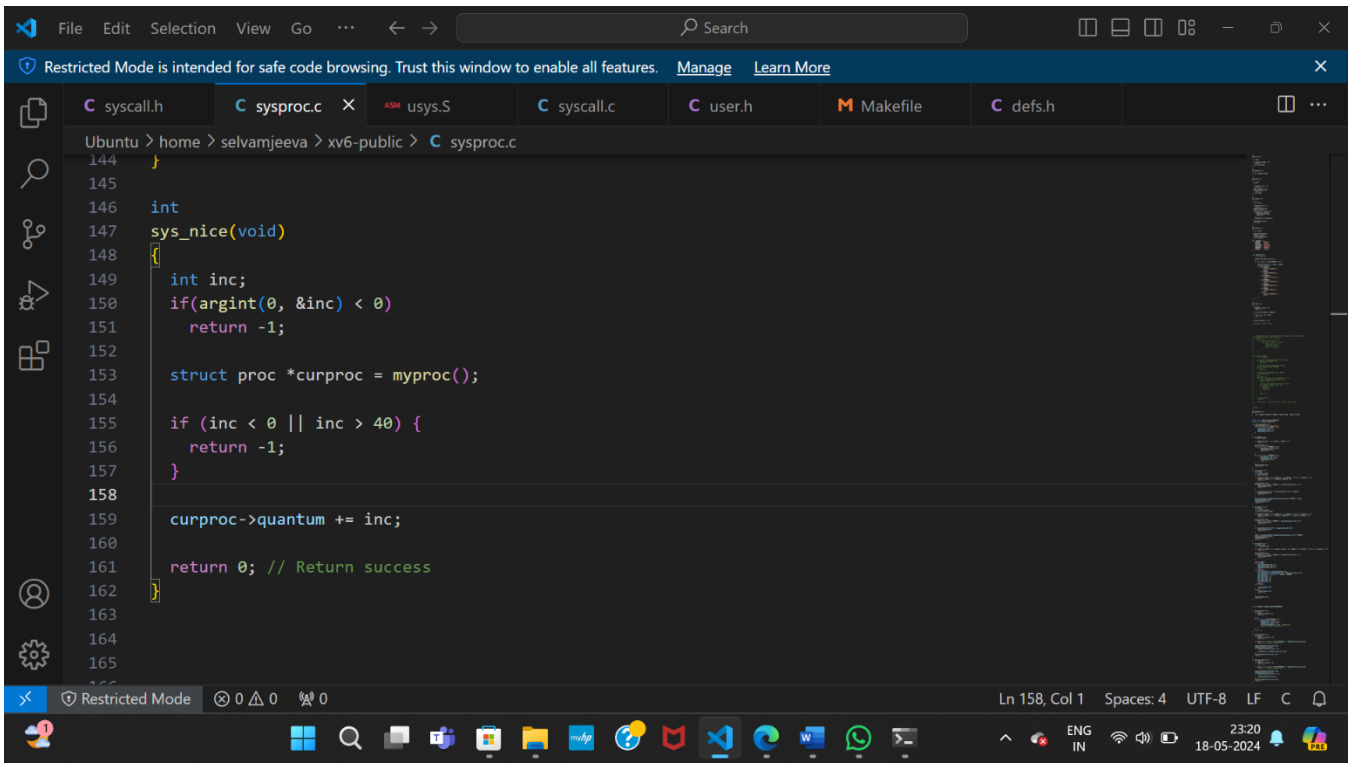


Sysproc.c :

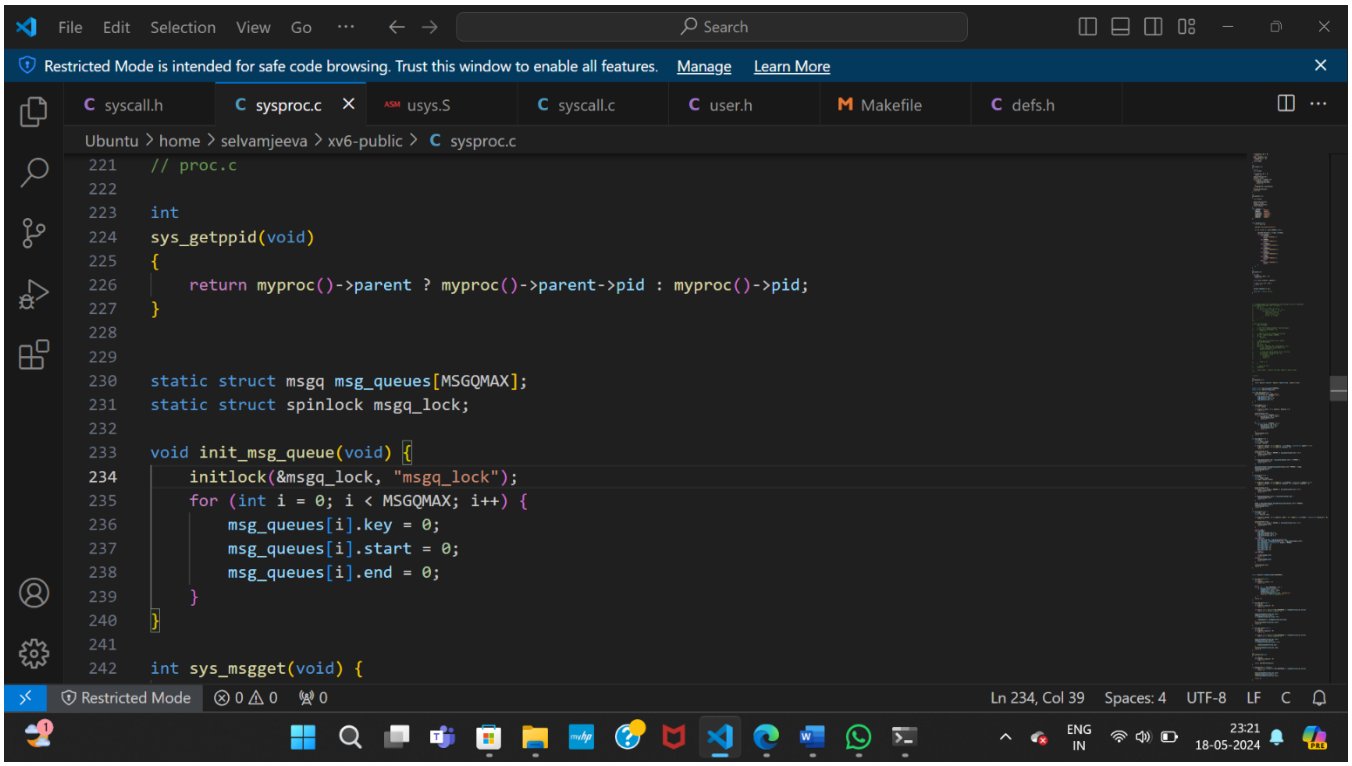


sysproc.c

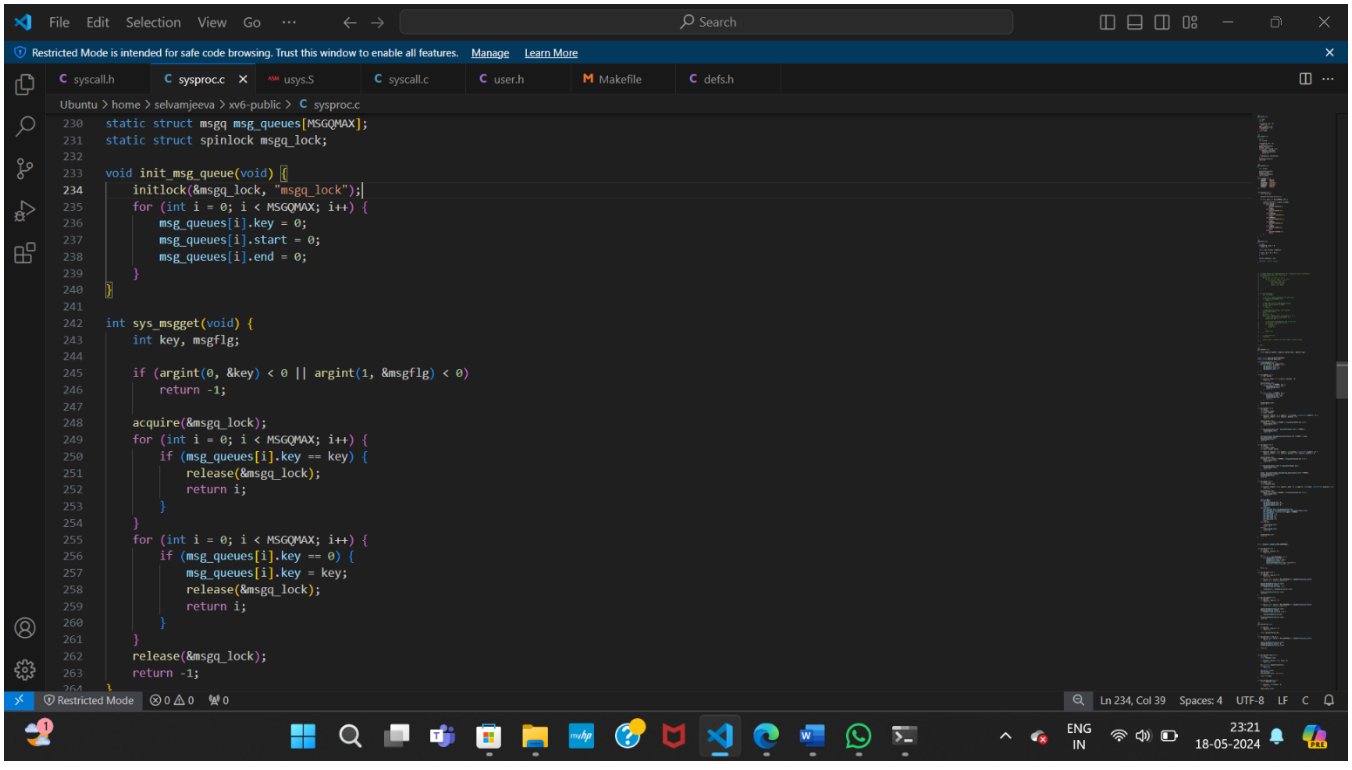

File Edit Selection View Go
Search
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
syscall.h
sysproc.c
usys.S
syscall.c
user.h
Makefile
defs.h
Ubuntu > home > selvamjeeva > xv6-public > sysproc.c
void sys_ps(void) {
struct proc *p;
cprintf("PID\tName\tState\n");
for (p = proc; p < &proc[NPROC]; p++) {
cprintf("%d\t%s\t", p->pid, p->name);
switch(p->state) {
case UNUSED:
cprintf("UNUSED\n");
break;
case EMBRYO:
cprintf("EMBRYO\n");
break;
case SLEEPING:
cprintf("SLEEPING\n");
break;
case RUNNABLE:
cprintf("RUNNABLE\n");
break;
case RUNNING:
cprintf("RUNNING\n");
break;
case ZOMBIE:
cprintf("ZOMBIE\n");
break;
default:
cprintf("UNKNOWN\n");
break;
}
}
}
Restricted Mode
Ln 158, Col 1 Spaces: 4 UTF-8 LF C
ENG IN
23:20 18-05-2024


File Edit Selection View Go
Search
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
syscall.h
sysproc.c
usys.S
syscall.c
user.h
Makefile
defs.h
Ubuntu > home > selvamjeeva > xv6-public > sysproc.c
}
int
sys_nice(void)
{
int inc;
if(argint(0, &inc) < 0)
return -1;
struct proc *curproc = myproc();
if (inc < 0 || inc > 40) {
return -1;
}
curproc->quantum += inc;
return 0; // Return success
}
Restricted Mode
Ln 158, Col 1 Spaces: 4 UTF-8 LF C
ENG IN
23:20 18-05-2024

sysproc.c

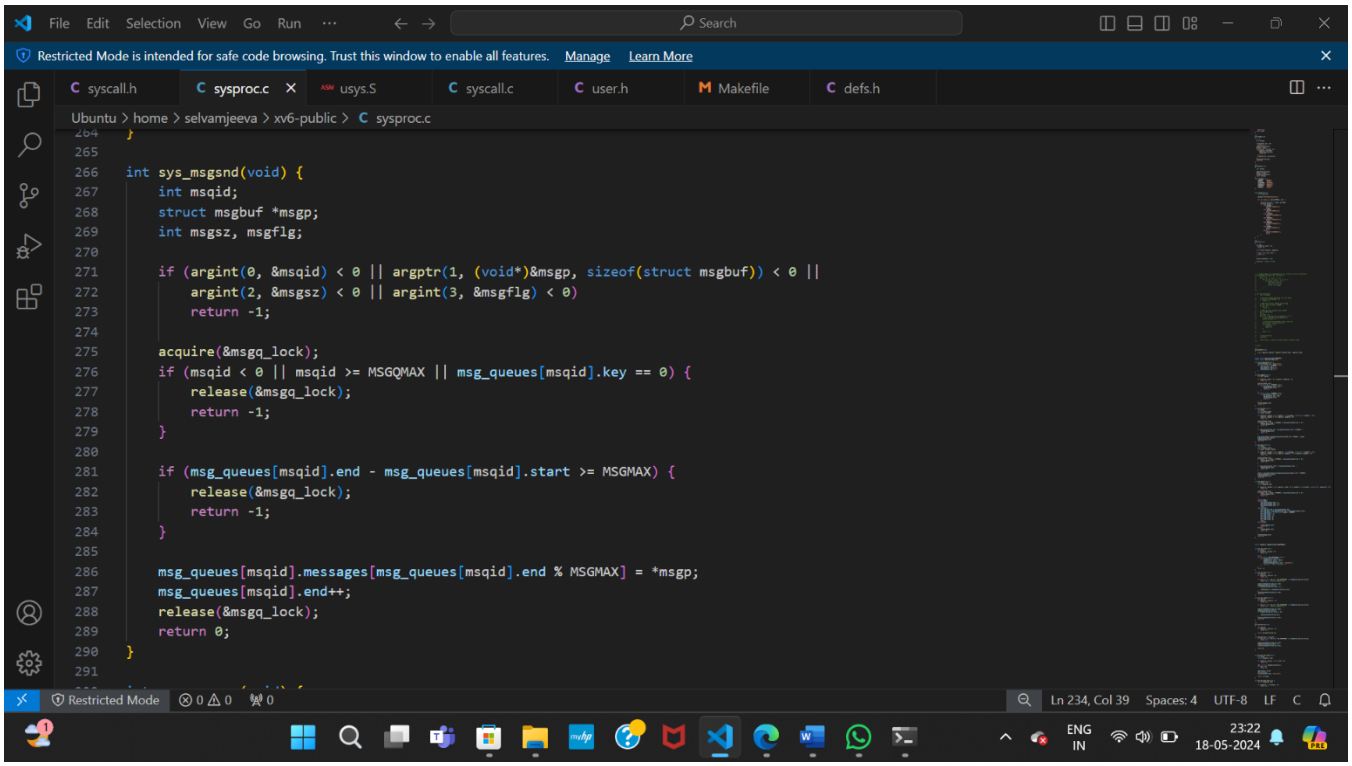


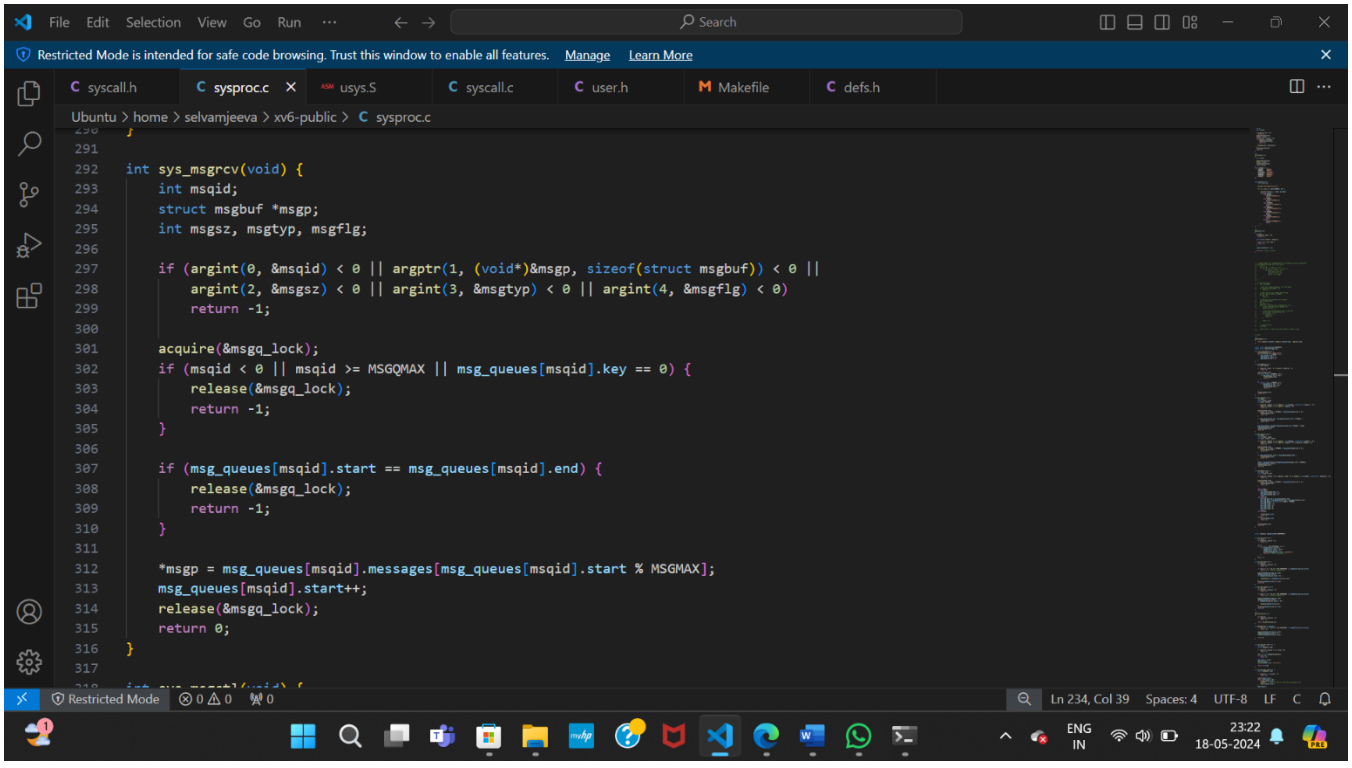
```
// proc.c

int
sys_getppid(void)
{
    return myproc()->parent ? myproc()->parent->pid : myproc()->pid;
}


static struct msgq msg_queues[MSGQMAX];
static struct spinlock msgq_lock;

void init_msg_queue(void) {
    initlock(&msgq_lock, "msgq_lock");
    for (int i = 0; i < MSGQMAX; i++) {
        msg_queues[i].key = 0;
        msg_queues[i].start = 0;
        msg_queues[i].end = 0;
    }
}

int sys_msgget(void) {
```



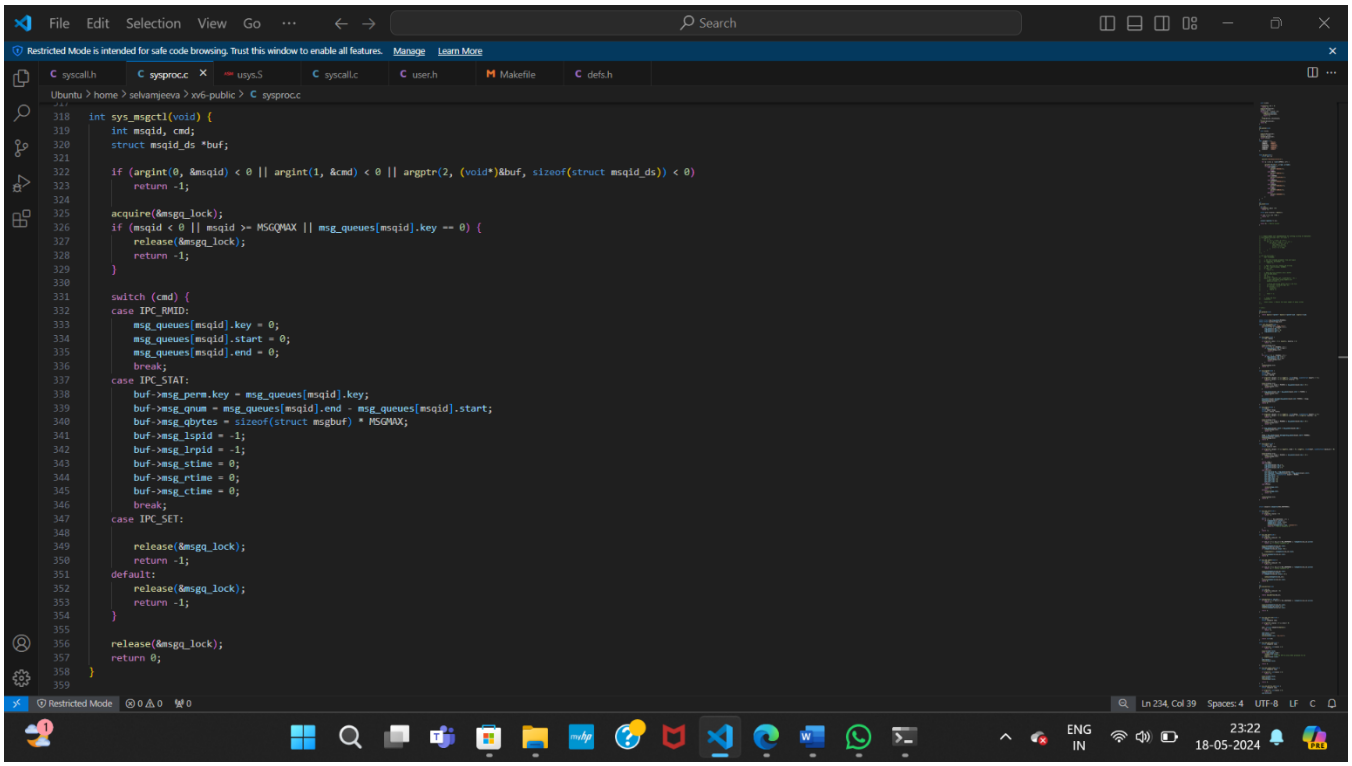
```
static struct msgq msg_queues[MSGQMAX];
static struct spinlock msgq_lock;

void init_msg_queue(void) {
    initlock(&msgq_lock, "msgq_lock");
    for (int i = 0; i < MSGQMAX; i++) {
        msg_queues[i].key = 0;
        msg_queues[i].start = 0;
        msg_queues[i].end = 0;
    }
}

int sys_msgget(void) {
    int key, msgflg;

    if (argint(0, &key) < 0 || argint(1, &msgflg) < 0)
        return -1;

    acquire(&msgq_lock);
    for (int i = 0; i < MSGQMAX; i++) {
        if (msg_queues[i].key == key) {
            release(&msgq_lock);
            return i;
        }
    }
    for (int i = 0; i < MSGQMAX; i++) {
        if (msg_queues[i].key == 0) {
            msg_queues[i].key = key;
            release(&msgq_lock);
            return i;
        }
    }
    release(&msgq_lock);
    return -1;
}
```

sysproc.c

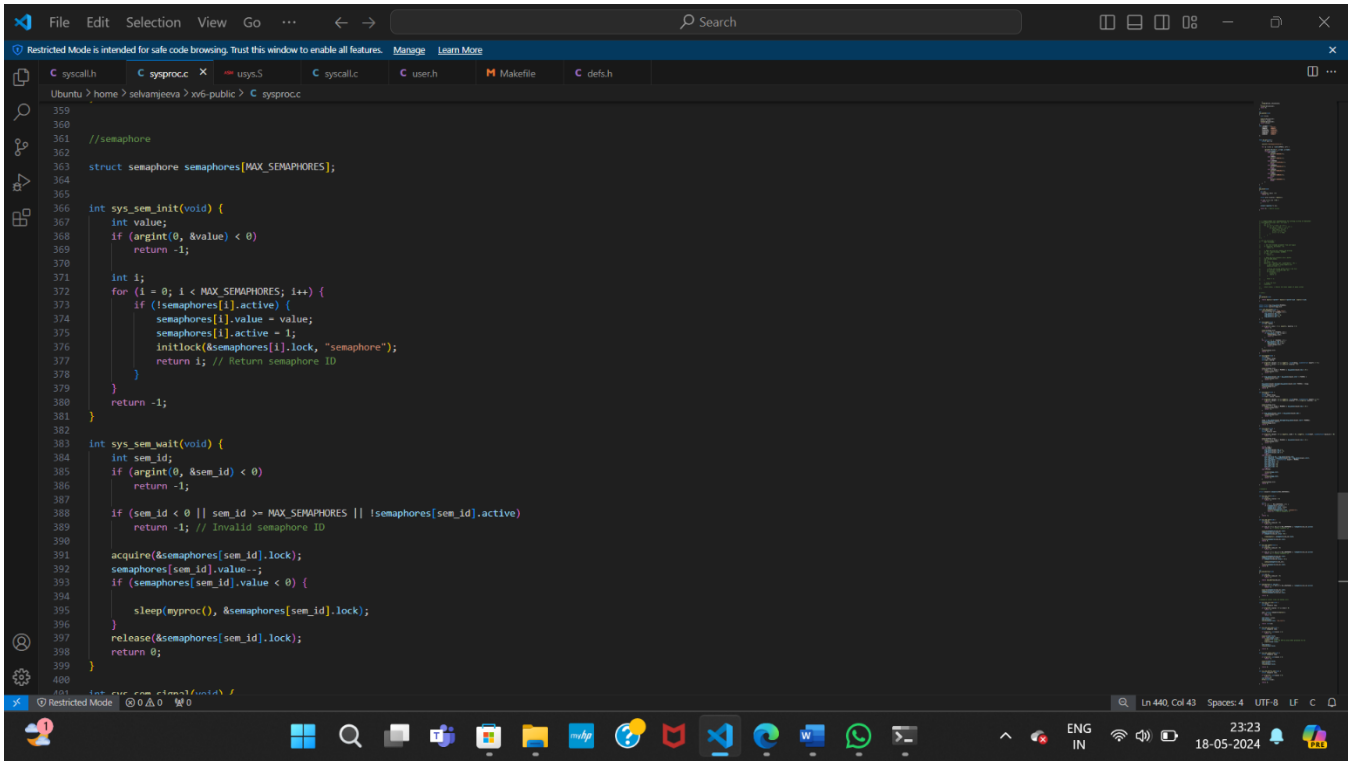

File Edit Selection View Go Run
Search
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
syscall.h sysproc.c usys.S syscall.c user.h Makefile defs.h
Ubuntu > home > selvamjeeva > xv6-public > sysproc.c
int sys_msgsnd(void) {
int msqid;
struct msgbuf *msgp;
int msgsz, msgflg;
if (argint(0, &msqid) < 0 || argptr(1, (void*)&msgp, sizeof(struct msgbuf)) < 0 ||
argint(2, &msgsz) < 0 || argint(3, &msgflg) < 0)
return -1;
acquire(&msgq_lock);
if (msqid < 0 || msqid >= MSGQMAX || msg_queues[msqid].key == 0) {
release(&msgq_lock);
return -1;
}
if (msg_queues[msqid].end - msg_queues[msqid].start >= MSGMAX) {
release(&msgq_lock);
return -1;
}
msg_queues[msqid].messages[msg_queues[msqid].end % MSGMAX] = *msgp;
msg_queues[msqid].end++;
release(&msgq_lock);
return 0;
}
Restricted Mode
Ln 234, Col 39 Spaces: 4 UTF-8 LF C
ENG IN
23:22 18-05-2024


File Edit Selection View Go Run
Search
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
syscall.h sysproc.c usys.S syscall.c user.h Makefile defs.h
Ubuntu > home > selvamjeeva > xv6-public > sysproc.c
int sys_msgrcv(void) {
int msqid;
struct msgbuf *msgp;
int msgsz, msgtyp, msgflg;
if (argint(0, &msqid) < 0 || argptr(1, (void*)&msgp, sizeof(struct msgbuf)) < 0 ||
argint(2, &msgsz) < 0 || argint(3, &msgtyp) < 0 || argint(4, &msgflg) < 0)
return -1;
acquire(&msgq_lock);
if (msqid < 0 || msqid >= MSGQMAX || msg_queues[msqid].key == 0) {
release(&msgq_lock);
return -1;
}
if (msg_queues[msqid].start == msg_queues[msqid].end) {
release(&msgq_lock);
return -1;
}
*msgp = msg_queues[msqid].messages[msg_queues[msqid].start % MSGMAX];
msg_queues[msqid].start++;
release(&msgq_lock);
return 0;
}
Restricted Mode
Ln 234, Col 39 Spaces: 4 UTF-8 LF C
ENG IN
23:22 18-05-2024

sysproc.c

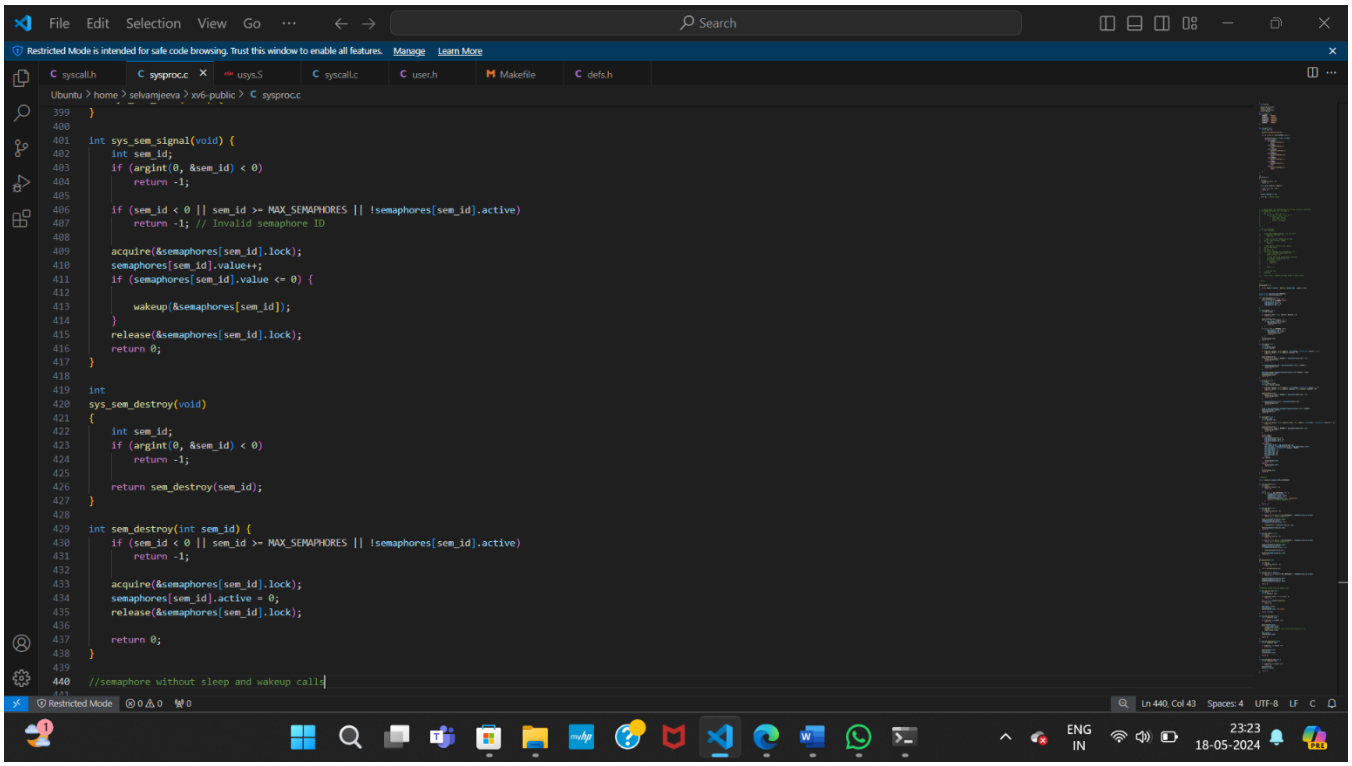


```
int sys_msgctl(void) {
    int msqid, cmd;
    struct msqid_ds *buf;

    if (argint(0, &msqid) < 0 || argint(1, &cmd) < 0 || argptr(2, (void*)&buf, sizeof(struct msqid_ds)) < 0)
        return -1;

    acquire(&msgq_lock);
    if (msqid < 0 || msqid >= MSGQMAX || msg_queues[msqid].key == 0) {
        release(&msgq_lock);
        return -1;
    }

    switch (cmd) {
    case IPC_RMID:
        msg_queues[msqid].key = 0;
        msg_queues[msqid].start = 0;
        msg_queues[msqid].end = 0;
        break;
    case IPC_STAT:
        buf->msg_perm.key = msg_queues[msqid].key;
        buf->msg_qnum = msg_queues[msqid].end - msg_queues[msqid].start;
        buf->msg_qbytes = sizeof(struct msgbuf) * MSGMAX;
        buf->msg_lspid = -1;
        buf->msg_lrpid = -1;
        buf->msg_stime = 0;
        buf->msg_rtime = 0;
        buf->msg_ctime = 0;
        break;
    case IPC_SET:

        release(&msgq_lock);
        return -1;
    default:
        release(&msgq_lock);
        return -1;
    }

    release(&msgq_lock);
    return 0;
}
```

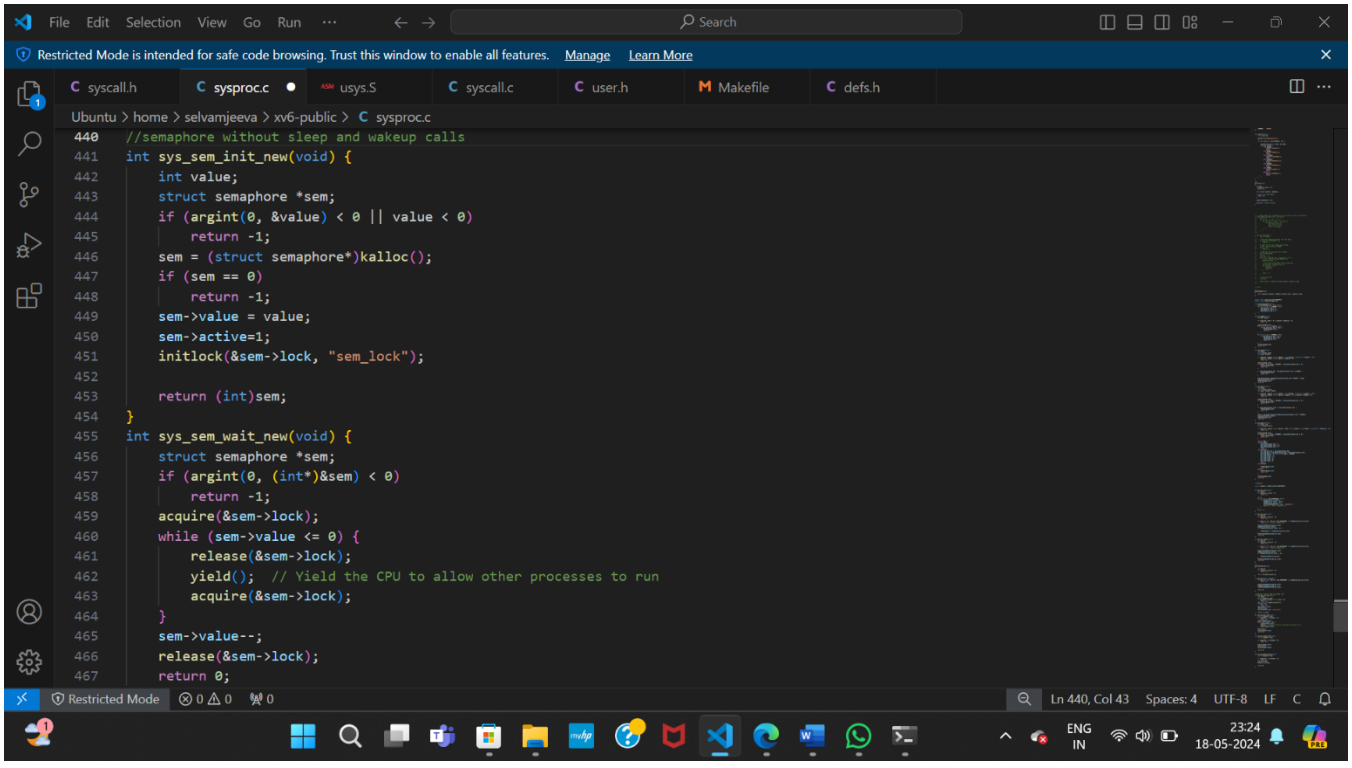


```
//semaphore

struct semaphore semaphores[MAX_SEMAPHORES];


int sys_sem_init(void) {
    int value;
    if (argint(0, &value) < 0)
        return -1;

    int i;
    for (i = 0; i < MAX_SEMAPHORES; i++) {
        if (!semaphores[i].active) {
            semaphores[i].value = value;
            semaphores[i].active = 1;
            initlock(&semaphores[i].lock, "semaphore");
            return i; // Return semaphore ID
        }
    }
    return -1;
}

int sys_sem_wait(void) {
    int sem_id;
    if (argint(0, &sem_id) < 0)
        return -1;

    if (sem_id < 0 || sem_id >= MAX_SEMAPHORES || !semaphores[sem_id].active)
        return -1; // Invalid semaphore ID

    acquire(&semaphores[sem_id].lock);
    semaphores[sem_id].value--;
    if (semaphores[sem_id].value < 0) {

        sleep(myproc(), &semaphores[sem_id].lock);
    }
    release(&semaphores[sem_id].lock);
    return 0;
}
```

sysproc.c

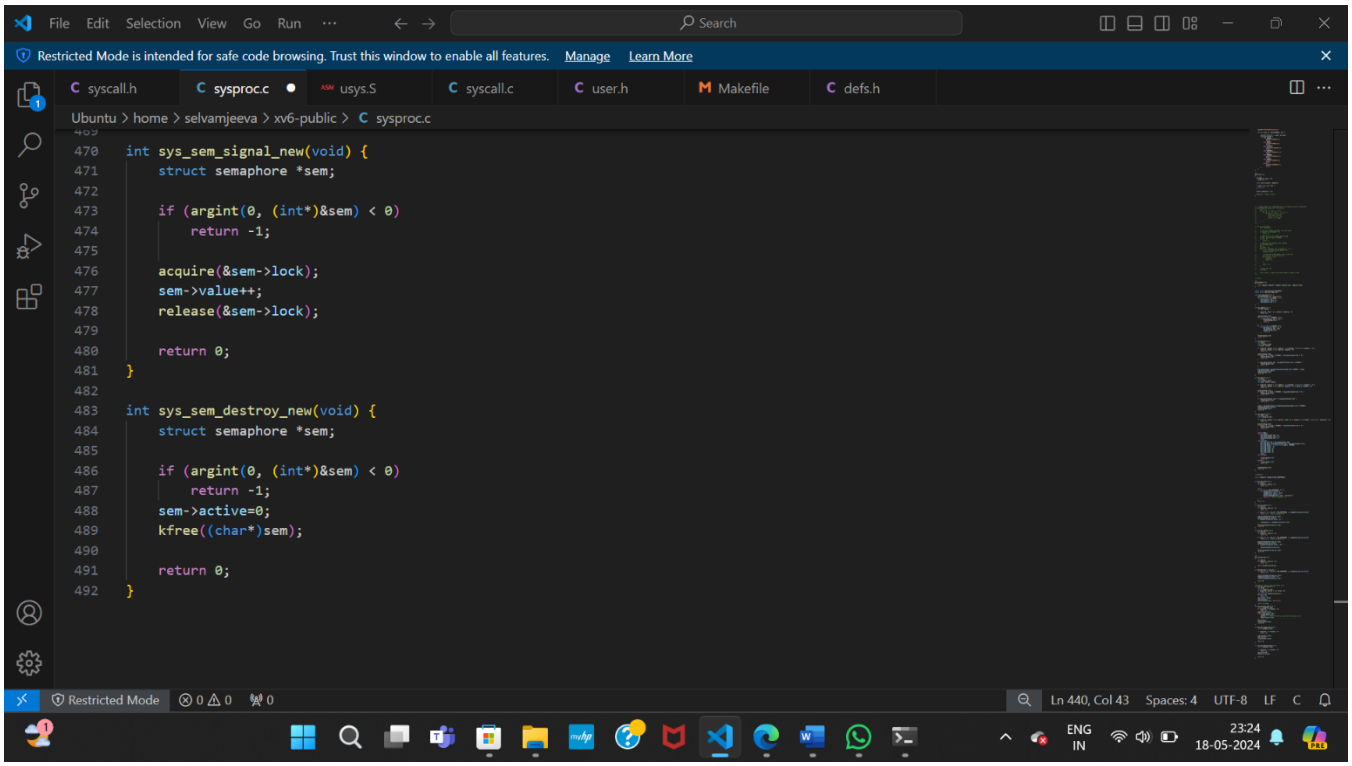


```
399  }
400
401  int sys_sem_signal(void) {
402      int sem_id;
403      if (argint(0, &sem_id) < 0)
404          return -1;
405
406      if (sem_id < 0 || sem_id >= MAX_SEMAPHORES || !semaphores[sem_id].active)
407          return -1; // Invalid semaphore ID
408
409      acquire(&semaphores[sem_id].lock);
410      semaphores[sem_id].value++;
411      if (semaphores[sem_id].value <= 0) {
412
413          wakeup(&semaphores[sem_id]);
414      }
415      release(&semaphores[sem_id].lock);
416      return 0;
417  }
418
419  int
420  sys_sem_destroy(void)
421  {
422      int sem_id;
423      if (argint(0, &sem_id) < 0)
424          return -1;
425
426      return sem_destroy(sem_id);
427  }
428
429  int sem_destroy(int sem_id) {
430      if (sem_id < 0 || sem_id >= MAX_SEMAPHORES || !semaphores[sem_id].active)
431          return -1;
432
433      acquire(&semaphores[sem_id].lock);
434      semaphores[sem_id].active = 0;
435      release(&semaphores[sem_id].lock);
436
437      return 0;
438  }
439
440  //semaphore without sleep and wakeup calls
```

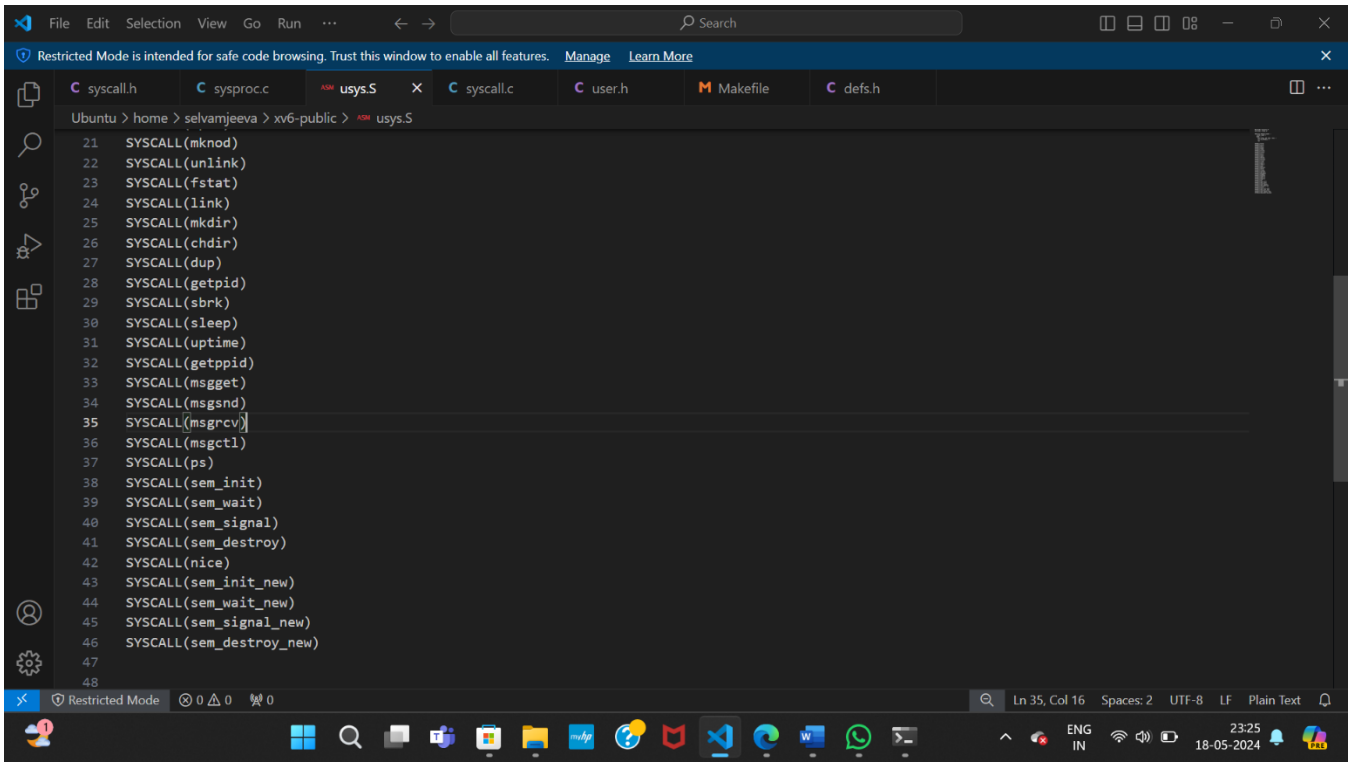


```
440  //semaphore without sleep and wakeup calls
441  int sys_sem_init_new(void) {
442      int value;
443      struct semaphore *sem;
444      if (argint(0, &value) < 0 || value < 0)
445          return -1;
446      sem = (struct semaphore*)kalloc();
447      if (sem == 0)
448          return -1;
449      sem->value = value;
450      sem->active=1;
451      initlock(&sem->lock, "sem_lock");
452
453      return (int)sem;
454  }
455  int sys_sem_wait_new(void) {
456      struct semaphore *sem;
457      if (argint(0, (int*)&sem) < 0)
458          return -1;
459      acquire(&sem->lock);
460      while (sem->value <= 0) {
461          release(&sem->lock);
462          yield();  // Yield the CPU to allow other processes to run
463          acquire(&sem->lock);
464      }
465      sem->value--;
466      release(&sem->lock);
467      return 0;
```

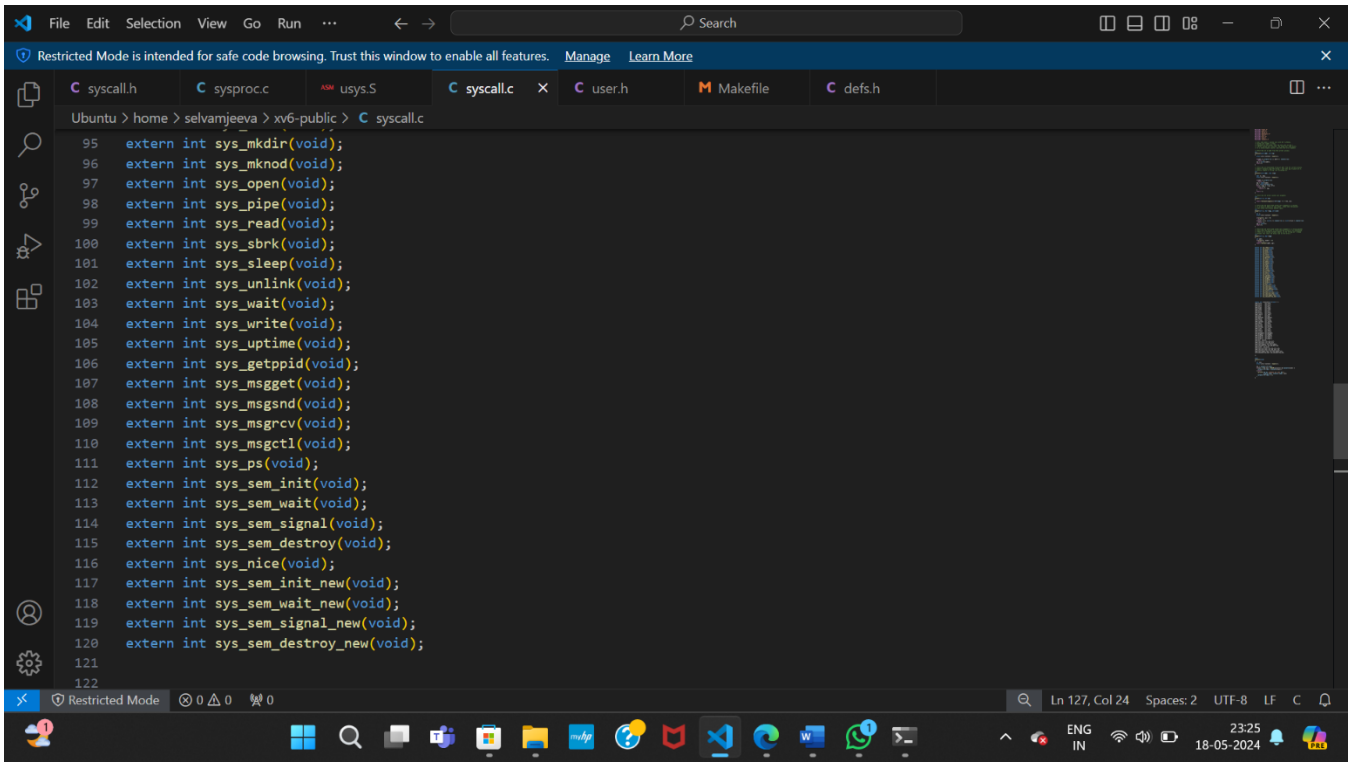
sysproc.c



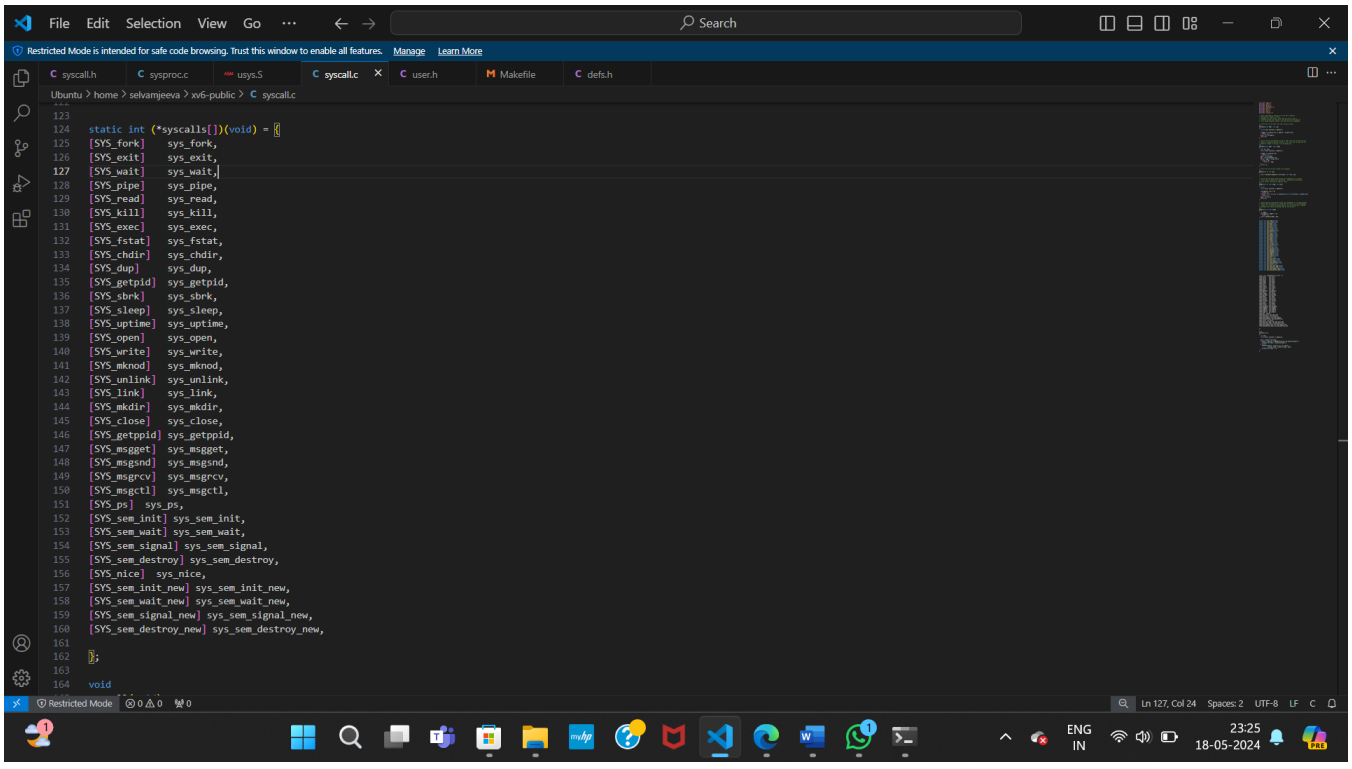
Usys.S :



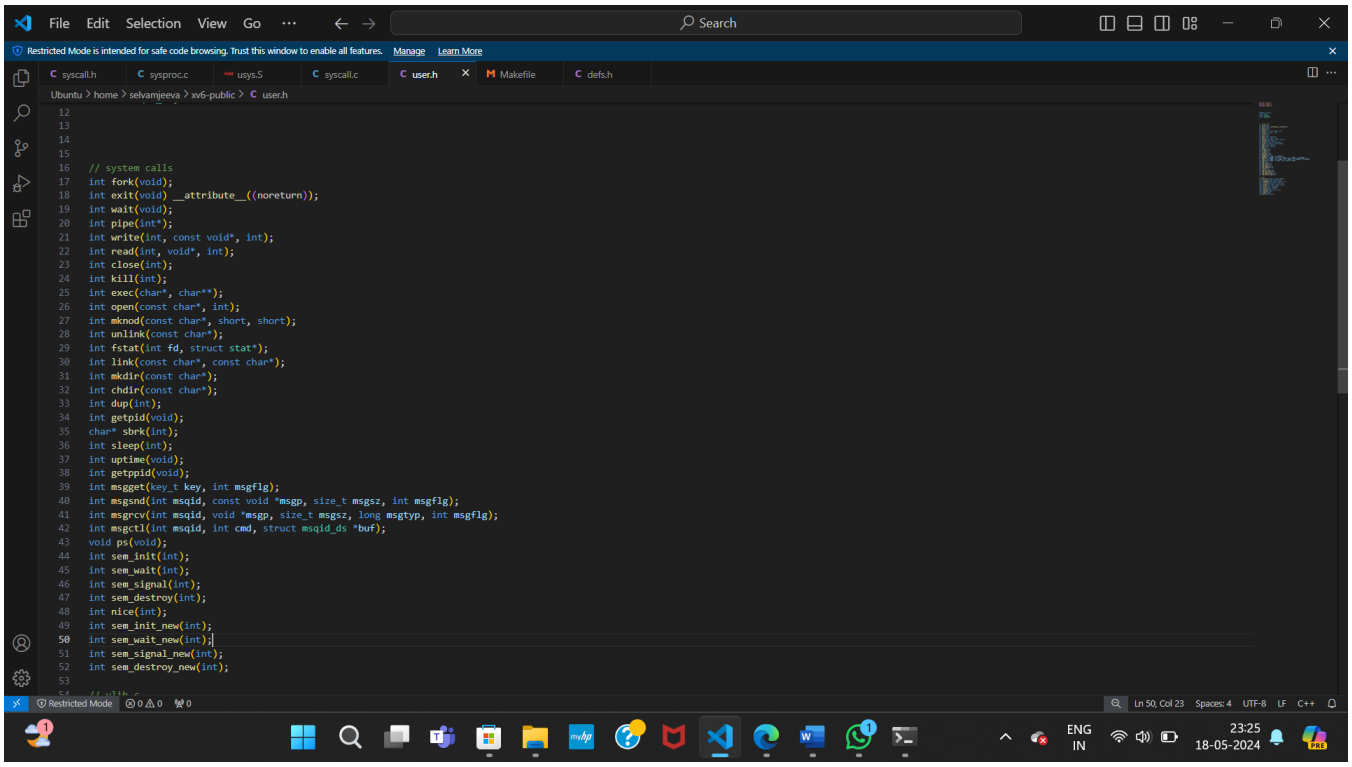
Syscall.c :



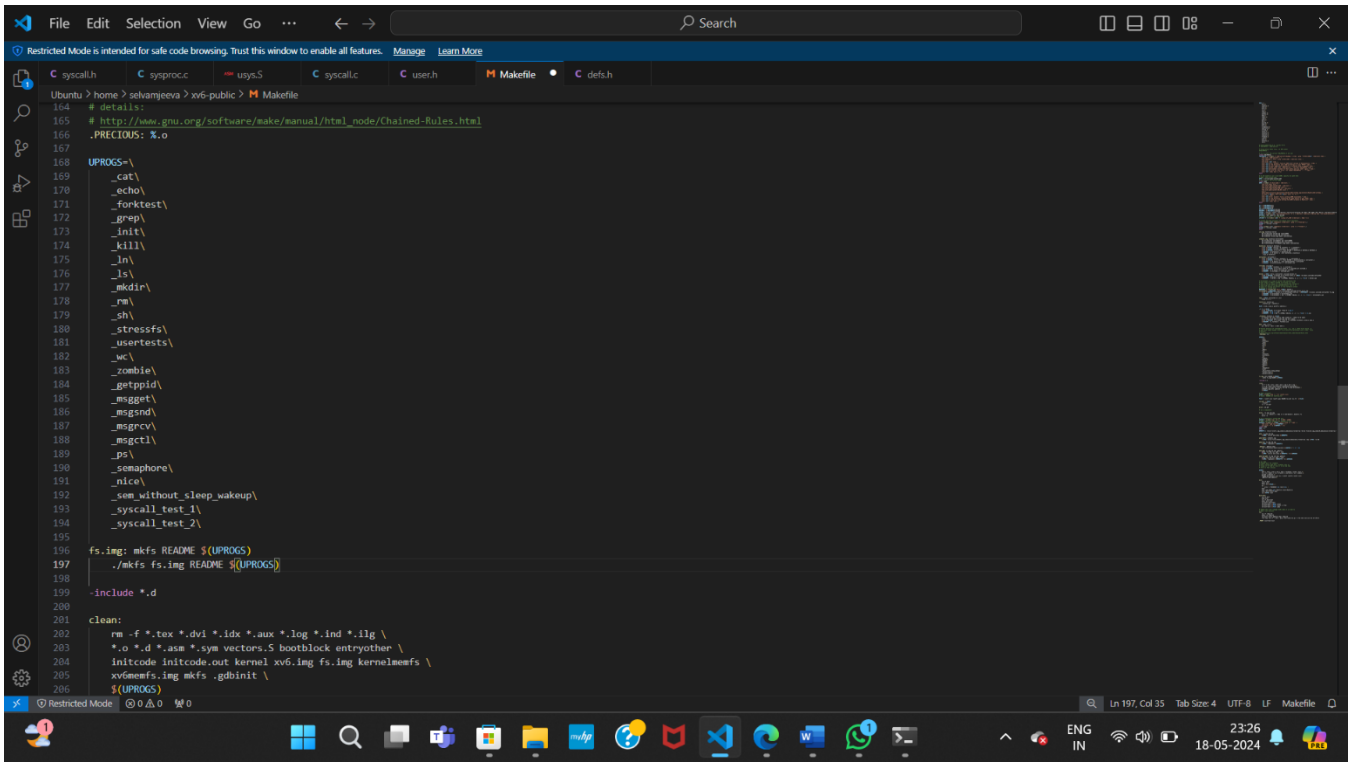
Syscall.c :



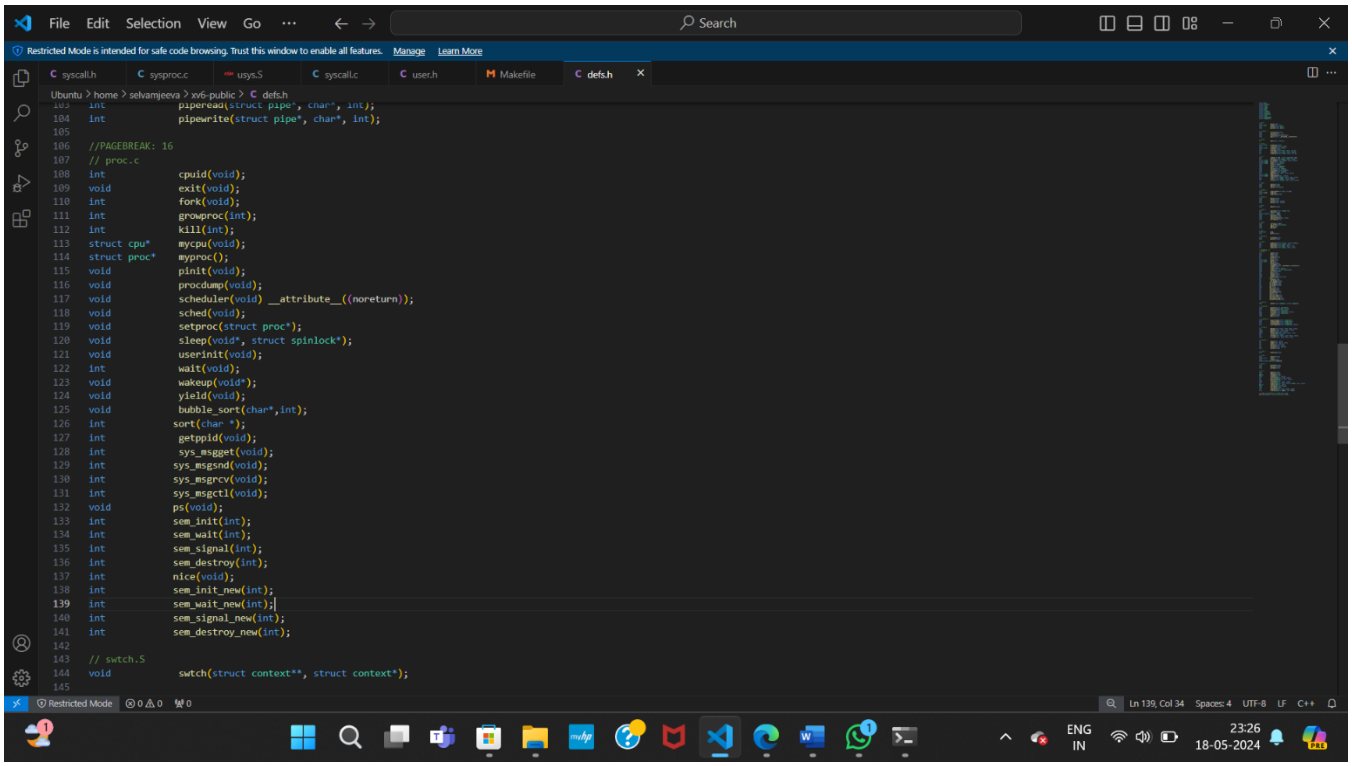
User.h :



Makefile :



defs.h:


File Edit Selection View Go
Search
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
syscall.h
sysproc.c
usys.S
syscall.c
user.h
Makefile
defs.h
Ubuntu > home > selvamjeeva > xv6-public > defs.h
int pipewrite(struct pipe*, char*, int);
//PAGEBREAK: 16
// proc.c
int cpuid(void);
void exit(void);
int fork(void);
int growproc(int);
int kill(int);
struct cpu* mycpu(void);
struct proc* myproc();
void pinit(void);
void procdump(void);
void scheduler(void) __attribute__((noreturn));
void sched(void);
void setproc(struct proc*);
void sleep(void*, struct spinlock*);
void userinit(void);
int wait(void);
void wakeup(void*);
void yield(void);
void bubble_sort(char*,int);
int sort(char *);
int getppid(void);
int sys_msgget(void);
int sys_msgsnd(void);
int sys_msgrcv(void);
int sys_msgctl(void);
void ps(void);
int sem_init(int);
int sem_wait(int);
int sem_signal(int);
int sem_destroy(int);
int nice(void);
int sem_init_new(int);
int sem_wait_new(int);
int sem_signal_new(int);
int sem_destroy_new(int);
// swtch.S
void swtch(struct context**, struct context*);
Restricted Mode
Ln 139, Col 34 Spaces: 4 UTF-8 LF C++
ENG IN
23:26 18-05-2024