

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

```
#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 ~ + v
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
$
```



```
QEMU - Press Ctrl+Alt+G to release grab
msgcti      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

```
#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();
    }

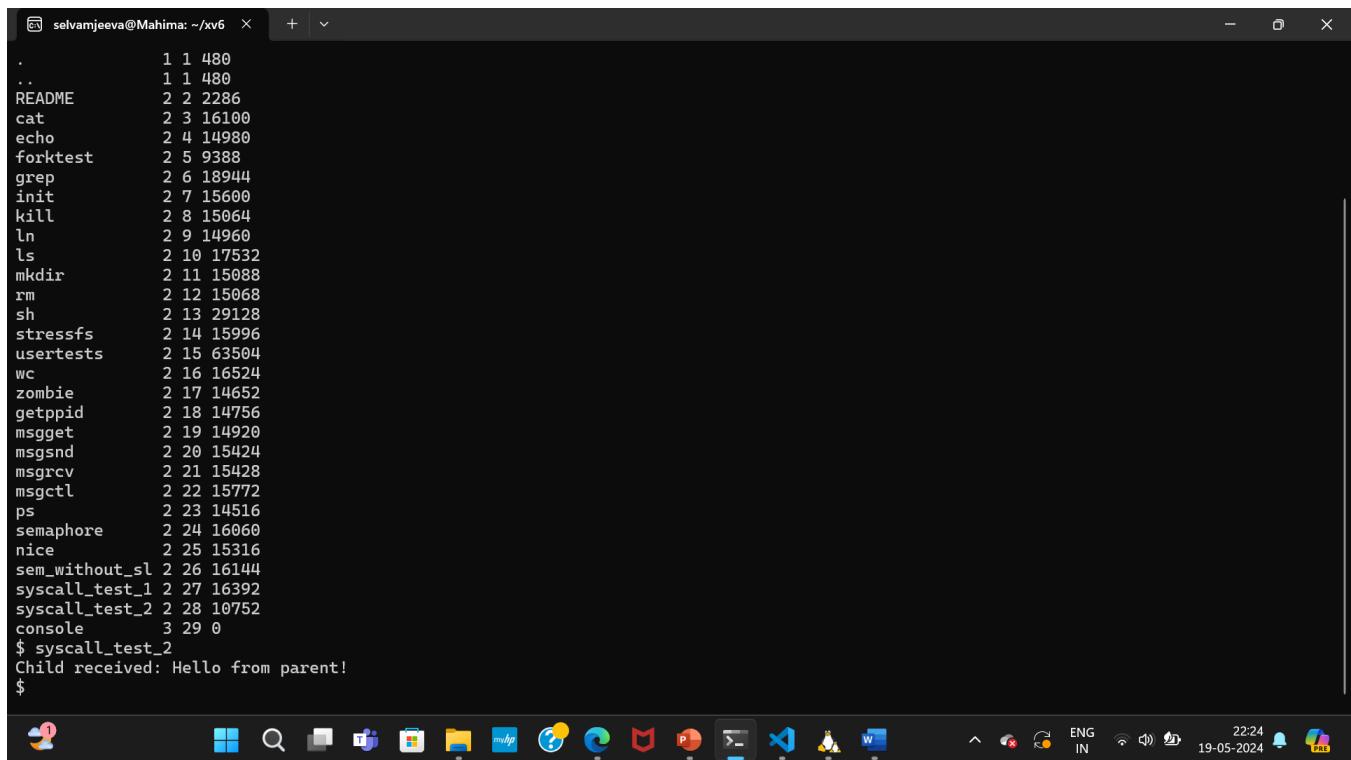
    pid = fork();
    if (pid < 0) {
        printf(1, "Error: fork failed\n");
        exit();
    } else if (pid == 0) {

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

The screenshot shows a terminal window titled "selvamjeeva@Mahima: ~/xv6 ~". The window displays a list of system calls or programs along with their file sizes and modification times. At the bottom of the list, the command "\$ syscall_test_2" is run, followed by the message "Child received: Hello from parent!". The taskbar at the bottom of the screen shows various application icons, and the system tray indicates the date and time as 19-05-2024 22:24.

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

```

```

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

```

#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() :

```
5) #include "types.h"
6) #include "x86.h"
7) #include "defs.h"
8) #include "date.h"
9) #include "param.h"
10) #include "memlayout.h"
11) #include "mmu.h"
12) #include "proc.h"
13) #include "spinlock.h"
14) #include "msg.h"
15) #include "sem.h"
16)
17) #define BUF_SIZE 512
18) #define MSGMAX 10
19) #define MSGQMAX 10
```

```
int sys_mssnd(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 msgsiz, 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 msghbuf) * 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 o

```

```

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 > ls
init: starting sh
$ ls
.
..
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

grep	Z 6 18944
init	Z 7 15600
kill	Z 8 15064
ln	Z 9 14960
ls	Z 10 17532
mkdir	Z 11 15088
rm	Z 12 15068
sh	Z 13 29128
stressfs	Z 14 15996
usertests	Z 15 63504
wc	Z 16 16524
zombie	Z 17 14652
getppid	Z 18 14756
msgget	Z 19 14920
msgsnd	Z 20 15424
msgrcv	Z 21 15428
msgctl	Z 22 15772
ps	Z 23 14516
semaphore	Z 24 16060
nice	Z 25 15316
sem_without_sl	Z 26 16144
syscall_test_1	Z 27 16392
console	Z 28 0
	S -



ENG IN 00:00 19-05-2024

```
selvamjeeva@Mahima: ~/xv6 > $ getppid
Parent PID: 2
$ $ nice
$ 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

0	0=UNUSED
1	0=UNUSED
2	0=UNUSED
3	0=UNUSED
4	0=UNUSED
5	0=UNUSED
6	0=UNUSED
7	0=UNUSED
8	0=UNUSED
9	0=UNUSED
10	0=UNUSED
11	0=UNUSED
12	0=UNUSED
13	0=UNUSED
14	0=UNUSED
15	0=UNUSED
16	0=UNUSED
17	0=UNUSED
18	0=UNUSED
19	0=UNUSED
20	0=UNUSED
21	0=UNUSED
22	0=UNUSED
23	0=UNUSED
24	0=UNUSED
25	0=UNUSED
26	0=UNUSED
27	0=UNUSED
28	0=UNUSED
29	0=UNUSED
30	0=UNUSED
31	0=UNUSED
32	0=UNUSED
33	0=UNUSED
34	0=UNUSED
35	0=UNUSED
36	0=UNUSED
37	0=UNUSED
38	0=UNUSED
39	0=UNUSED
40	0=UNUSED
41	0=UNUSED
42	0=UNUSED
43	0=UNUSED
44	0=UNUSED
45	0=UNUSED
46	0=UNUSED
47	0=UNUSED
48	0=UNUSED
49	0=UNUSED
50	0=UNUSED
51	0=UNUSED
52	0=UNUSED
53	0=UNUSED
54	0=UNUSED
55	0=UNUSED
56	0=UNUSED
57	0=UNUSED
58	0=UNUSED
59	0=UNUSED
60	0=UNUSED
61	0=UNUSED
62	0=UNUSED
63	0=UNUSED
64	0=UNUSED
65	0=UNUSED
66	0=UNUSED
67	0=UNUSED
68	0=UNUSED
69	0=UNUSED
70	0=UNUSED
71	0=UNUSED
72	0=UNUSED
73	0=UNUSED
74	0=UNUSED
75	0=UNUSED
76	0=UNUSED
77	0=UNUSED
78	0=UNUSED
79	0=UNUSED
80	0=UNUSED
81	0=UNUSED
82	0=UNUSED
83	0=UNUSED
84	0=UNUSED
85	0=UNUSED
86	0=UNUSED
87	0=UNUSED
88	0=UNUSED
89	0=UNUSED
90	0=UNUSED
91	0=UNUSED
92	0=UNUSED
93	0=UNUSED
94	0=UNUSED
95	0=UNUSED
96	0=UNUSED
97	0=UNUSED
98	0=UNUSED
99	0=UNUSED
100	0=UNUSED
101	0=UNUSED
102	0=UNUSED
103	0=UNUSED
104	0=UNUSED
105	0=UNUSED
106	0=UNUSED
107	0=UNUSED
108	0=UNUSED
109	0=UNUSED
110	0=UNUSED
111	0=UNUSED
112	0=UNUSED
113	0=UNUSED
114	0=UNUSED
115	0=UNUSED
116	0=UNUSED
117	0=UNUSED
118	0=UNUSED
119	0=UNUSED
120	0=UNUSED
121	0=UNUSED
122	0=UNUSED
123	0=UNUSED
124	0=UNUSED
125	0=UNUSED
126	0=UNUSED
127	0=UNUSED
128	0=UNUSED
129	0=UNUSED
130	0=UNUSED
131	0=UNUSED
132	0=UNUSED
133	0=UNUSED
134	0=UNUSED
135	0=UNUSED
136	0=UNUSED
137	0=UNUSED
138	0=UNUSED
139	0=UNUSED
140	0=UNUSED
141	0=UNUSED
142	0=UNUSED
143	0=UNUSED
144	0=UNUSED
145	0=UNUSED
146	0=UNUSED
147	0=UNUSED
148	0=UNUSED
149	0=UNUSED
150	0=UNUSED
151	0=UNUSED
152	0=UNUSED
153	0=UNUSED
154	0=UNUSED
155	0=UNUSED
156	0=UNUSED
157	0=UNUSED
158	0=UNUSED
159	0=UNUSED
160	0=UNUSED
161	0=UNUSED
162	0=UNUSED
163	0=UNUSED
164	0=UNUSED
165	0=UNUSED
166	0=UNUSED
167	0=UNUSED
168	0=UNUSED
169	0=UNUSED
170	0=UNUSED
171	0=UNUSED
172	0=UNUSED
173	0=UNUSED
174	0=UNUSED
175	0=UNUSED
176	0=UNUSED
177	0=UNUSED
178	0=UNUSED
179	0=UNUSED
180	0=UNUSED
181	0=UNUSED
182	0=UNUSED
183	0=UNUSED
184	0=UNUSED
185	0=UNUSED
186	0=UNUSED
187	0=UNUSED
188	0=UNUSED
189	0=UNUSED
190	0=UNUSED
191	0=UNUSED
192	0=UNUSED
193	0=UNUSED
194	0=UNUSED
195	0=UNUSED
196	0=UNUSED
197	0=UNUSED
198	0=UNUSED
199	0=UNUSED
200	0=UNUSED
201	0=UNUSED
202	0=UNUSED
203	0=UNUSED
204	0=UNUSED
205	0=UNUSED
206	0=UNUSED
207	0=UNUSED
208	0=UNUSED
209	0=UNUSED
210	0=UNUSED
211	0=UNUSED
212	0=UNUSED
213	0=UNUSED
214	0=UNUSED
215	0=UNUSED
216	0=UNUSED
217	0=UNUSED
218	0=UNUSED
219	0=UNUSED
220	0=UNUSED
221	0=UNUSED
222	0=UNUSED
223	0=UNUSED
224	0=UNUSED
225	0=UNUSED
226	0=UNUSED
227	0=UNUSED
228	0=UNUSED
229	0=UNUSED
230	0=UNUSED
231	0=UNUSED
232	0=UNUSED
233	0=UNUSED
234	0=UNUSED
235	0=UNUSED
236	0=UNUSED
237	0=UNUSED
238	0=UNUSED
239	0=UNUSED
240	0=UNUSED
241	0=UNUSED
242	0=UNUSED
243	0=UNUSED
244	0=UNUSED
245	0=UNUSED
246	0=UNUSED
247	0=UNUSED
248	0=UNUSED
249	0=UNUSED
250	0=UNUSED
251	0=UNUSED
252	0=UNUSED
253	0=UNUSED
254	0=UNUSED
255	0=UNUSED
256	0=UNUSED
257	0=UNUSED
258	0=UNUSED
259	0=UNUSED
260	0=UNUSED
261	0=UNUSED
262	0=UNUSED
263	0=UNUSED
264	0=UNUSED
265	0=UNUSED
266	0=UNUSED
267	0=UNUSED
268	0=UNUSED
269	0=UNUSED
270	0=UNUSED
271	0=UNUSED
272	0=UNUSED
273	0=UNUSED
274	0=UNUSED
275	0=UNUSED
276	0=UNUSED
277	0=UNUSED
278	0=UNUSED
279	0=UNUSED
280	0=UNUSED
281	0=UNUSED
282	0=UNUSED
283	0=UNUSED
284	0=UNUSED
285	0=UNUSED
286	0=UNUSED
287	0=UNUSED
288	0=UNUSED
289	0=UNUSED
290	0=UNUSED
291	0=UNUSED
292	0=UNUSED
293	0=UNUSED
294	0=UNUSED
295	0=UNUSED
296	0=UNUSED
297	0=UNUSED
298	0=UNUSED
299	0=UNUSED
300	0=UNUSED
301	0=UNUSED
302	0=UNUSED
303	0=UNUSED
304	0=UNUSED
305	0=UNUSED
306	0=UNUSED
307	0=UNUSED
308	0=UNUSED
309	0=UNUSED
310	0=UNUSED
311	0=UNUSED
312	0=UNUSED
313	0=UNUSED
314	0=UNUSED
315	0=UNUSED
316	0=UNUSED
317	0=UNUSED
318	0=UNUSED
319	0=UNUSED
320	0=UNUSED
321	0=UNUSED
322	0=UNUSED
323	0=UNUSED
324	0=UNUSED
325	0=UNUSED
326	0=UNUSED
327	0=UNUSED
328	0=UNUSED
329	0=UNUSED
330	0=UNUSED
331	0=UNUSED
332	0=UNUSED
333	0=UNUSED
334	0=UNUSED
335	0=UNUSED
336	0=UNUSED
337	0=UNUSED
338	0=UNUSED
339	0=UNUSED
340	0=UNUSED
341	0=UNUSED
342	0=UNUSED
343	0=UNUSED
344	0=UNUSED
345	0=UNUSED
346	0=UNUSED
347	0=UNUSED
348	0=UNUSED
349	0=UNUSED
350	0=UNUSED
351	0=UNUSED
352	0=UNUSED
353	0=UNUSED
354	0=UNUSED
355	0=UNUSED
356	0=UNUSED
357	0=UNUSED
358	0=UNUSED
359	0=UNUSED
360	0=UNUSED
361	0=UNUSED
362	0=UNUSED
363	0=UNUSED
364	0=UNUSED
365	0=UNUSED
366	0=UNUSED
367	0=UNUSED
368	0=UNUSED
369	0=UNUSED
370	0=UNUSED
371	0=UNUSED
372	0=UNUSED
373	0=UNUSED
374	0=UNUSED
375	0=UNUSED
376	0=UNUSED
377	0=UNUSED
378	0=UNUSED
379	0=UNUSED
380	0=UNUSED
381	0=UNUSED
382	0=UNUSED
383	0=UNUSED
384	0=UNUSED
385	0=UNUSED
386	0=UNUSED
387	0=UNUSED
388	0=UNUSED
389	0=UNUSED
390	0=UNUSED
391	0=UNUSED
392	0=UNUSED
393	0=UNUSED
394	0=UNUSED
395	0=UNUSED
396	0=UNUSED
397	0=UNUSED
398	0=UNUSED
399	0=UNUSED
400	0=UNUSED
401	0=UNUSED
402	0=UNUSED
403	0=UNUSED
404	0=UNUSED
405	0=UNUSED
406	0=UNUSED
407	0=UNUSED
408	0=UNUSED
409	0=UNUSED
410	0=UNUSED
411	0=UNUSED
412	0=UNUSED
413	0=UNUSED
414	0=UNUSED
415	0=UNUSED
416	0=UNUSED
417	0=UNUSED
418	0=UNUSED
419	0=UNUSED
420	0=UNUSED
421	0=UNUSED
422	0=UNUSED
423	0=UNUSED
424	0=UNUSED
425	0=UNUSED
426	0=UNUSED
427	0=UNUSED
428	0=UNUSED
429	0=UNUSED
430	0=UNUSED
431	0=UNUSED
432	0=UNUSED
433	0=UNUSED
434	0=UNUSED
435	0=UNUSED
436	0=UNUSED
437	0=UNUSED
438	0=UNUSED
439	0=UNUSED
440	0=UNUSED
441	0=UNUSED
442	0=UNUSED
443	0=UNUSED
444	0=UNUSED
445	0=UNUSED
446	0=UNUSED
447	0=UNUSED
448	0=UNUSED
449	0=UNUSED
450	0=UNUSED
451	0=UNUSED
452	0=UNUSED
453	0=UNUSED
454	0=UNUSED
455	0=UNUSED
456	0=UNUSED
457	0=UNUSED
458	0=UNUSED
459	0=UNUSED
460	0=UNUSED
461	0=UNUSED
462	0=UNUSED
463	0=UNUSED
464	0=UNUSED
465	0=UNUSED
466	0=UNUSED
467	0=UNUSED
468	0=UNUSED
469	0=UNUSED
470	0=UNUSED
471	0=UNUSED
472	0=UNUSED
473	0=UNUSED
474	0=UNUSED
475	0=UNUSED
476	0=UNUSED
477	0=UNUSED
478	0=UNUSED
479	0=UNUSED
480	0=UNUSED
481	0=UNUSED
482	0=UNUSED
483	0=UNUSED
484	0=UNUSED
485	0=UNUSED
486	0=UNUSED
487	0=UNUSED
488	0=UNUSED
489	0=UNUSED
490	0=UNUSED
491	0=UNUSED
492	0=UNUSED
493	0=UNUSED
494	0=UNUSED
495	0=UNUSED
496	0=UNUSED
497	0=UNUSED
498	0=UNUSED
499	0=UNUSED
500	0=UNUSED
501	0=UNUSED
502	0=UNUSED
503	0=UNUSED
504	0=UNUSED
505	0=UNUSED
506	0=UNUSED</

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"
```

```

#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:

```
#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(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(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(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(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) {

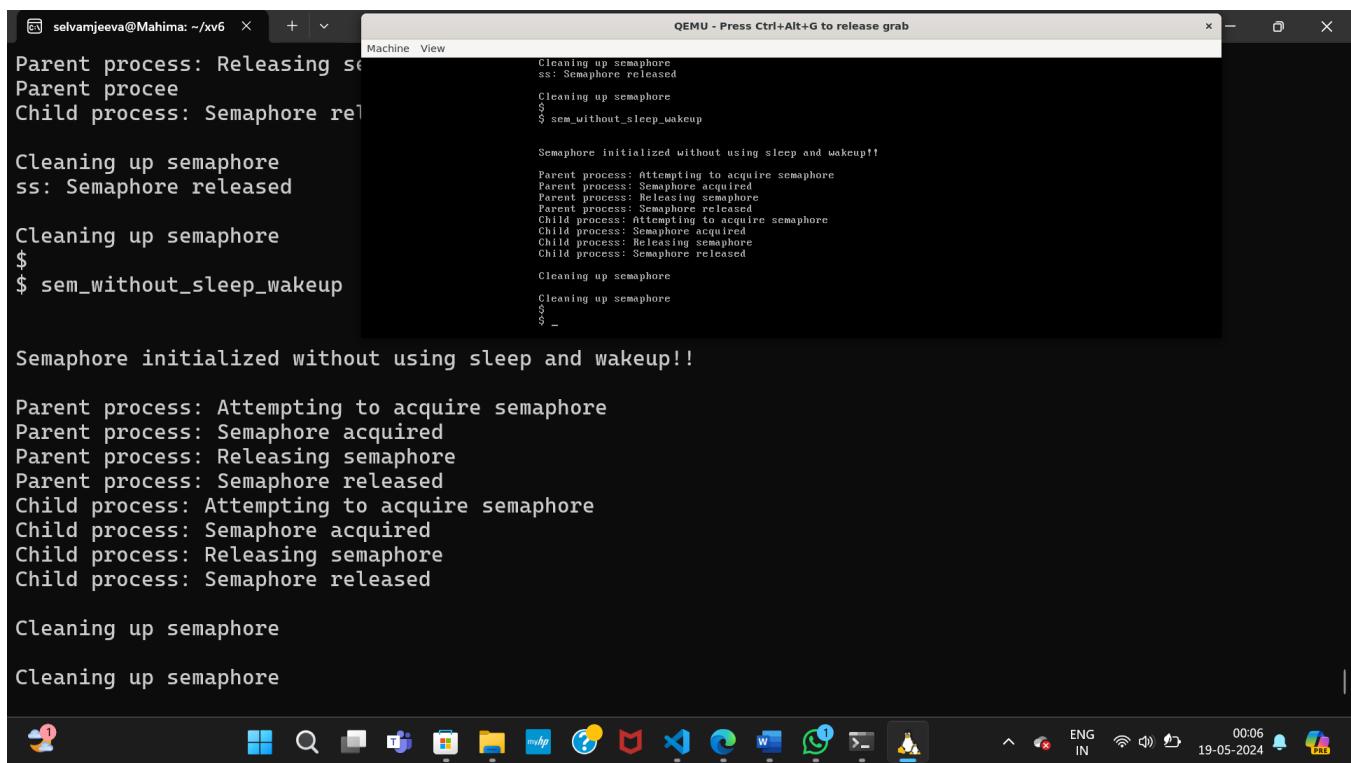
    printf(1, "Child process: Attempting to acquire semaphore\n");
    sem_wait_new(sem_id);
    printf(1, "Child process: Semaphore acquired\n");
    printf(1, "Child process: Releasing semaphore\n");
    sem_signal_new(sem_id);
    printf(1, "Child process: Semaphore released\n");
} else {

    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);

```

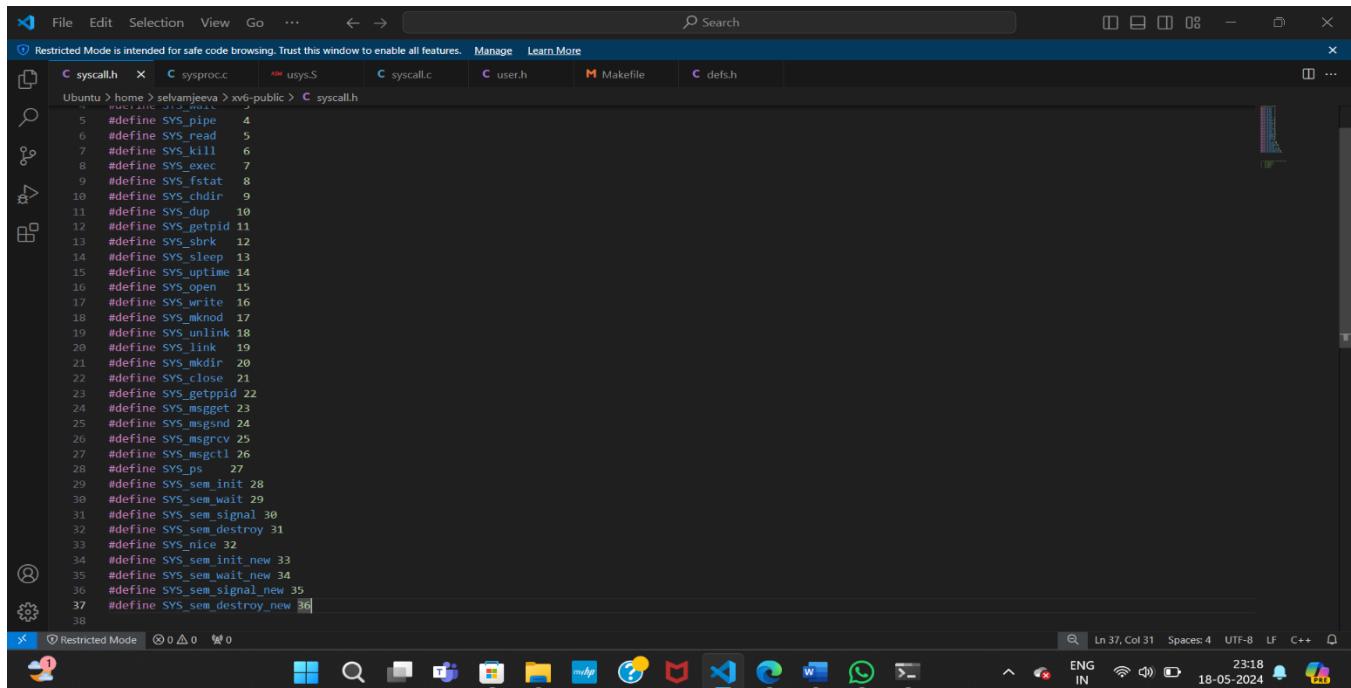
```
    exit();  
}  
  
OUTPUT:
```



```
selvamjeeva@Mahima: ~/xv6 > + < QEMU - Press Ctrl+Alt+G to release grab  
Machine View  
Parent process: Releasing semaphore  
Parent process: Semaphore released  
Child process: Semaphore released  
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
```

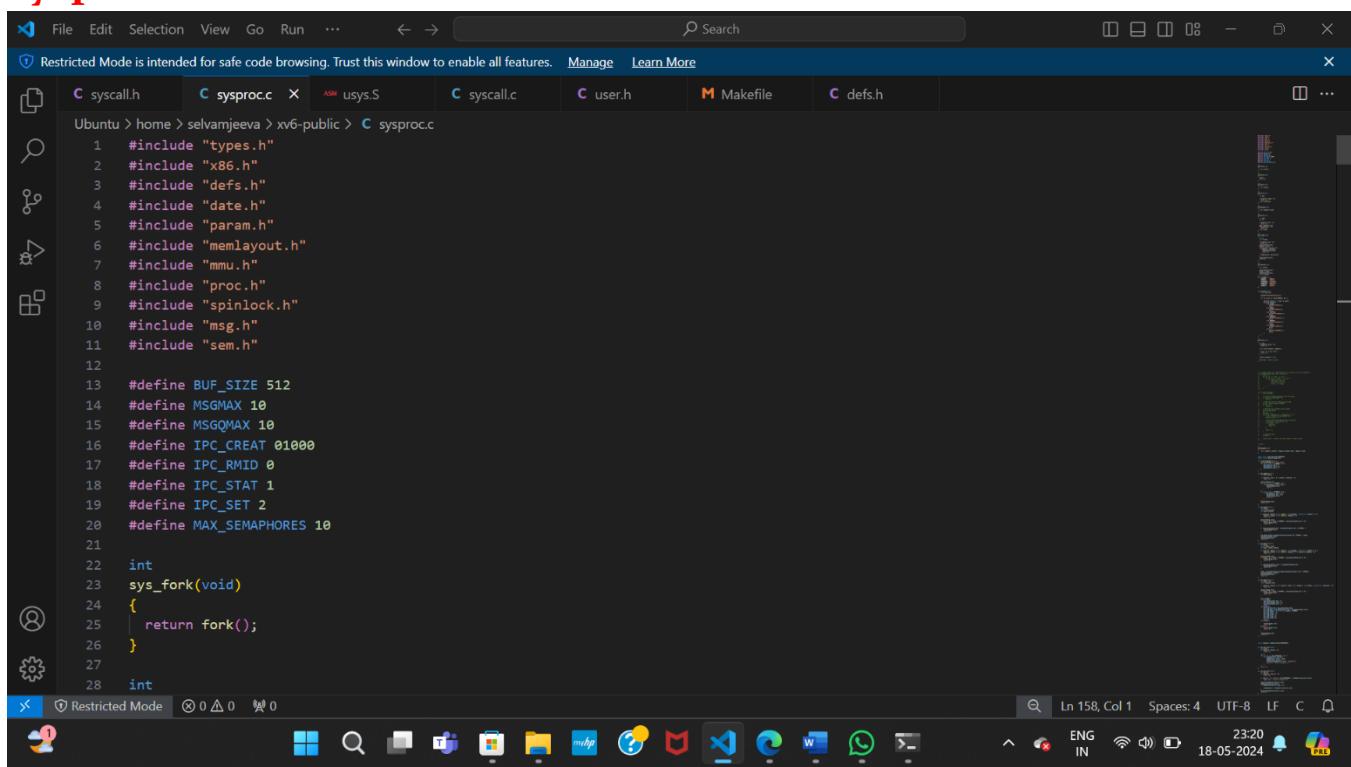
XV6 – MODIFIED FILE IMAGES :

Syscall.h :



```
#define SYS_pipe 4
#define SYS_read 5
#define SYS_kill 6
#define SYS_exec 7
#define SYS_fstat 8
#define SYS_chdir 9
#define SYS_dup 10
#define SYS_getpid 11
#define SYS_sbrk 12
#define SYS_sleep 13
#define SYS_uptime 14
#define SYS_open 15
#define SYS_write 16
#define SYS_mkdir 17
#define SYS_unlink 18
#define SYS_link 19
#define SYS_mkdir 20
#define SYS_close 21
#define SYS_getpid 22
#define SYS_msgget 23
#define SYS_msgrnd 24
#define SYS_msgrcv 25
#define SYS_msctl 26
#define SYS_ps 27
#define SYS_sem_init 28
#define SYS_sem_wait 29
#define SYS_sem_signal 30
#define SYS_sem_destroy 31
#define SYS_nice 32
#define SYS_sem_init_new 33
#define SYS_sem_wait_new 34
#define SYS_sem_signal_new 35
#define SYS_sem_destroy_new 36
```

Sysproc.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 BUF_SIZE 512
#define MSGMAX 10
#define MSGQMAX 10
#define IPC_CREAT 01000
#define IPC_RMID 0
#define IPC_STAT 1
#define IPC_SET 2
#define MAX_SEMAPHORES 10

int
sys_fork(void)
{
    return fork();
}

int
```

sysproc.c

```
Ubuntu > home > selvamjeeva > xv6-public > C sysproc.c

111
112 void sys_ps(void) {
113     struct proc *p;
114
115     cprintf("PID\tName\tState\n");
116
117     for (p = proc; p < &proc[NPROC]; p++) {
118
119         cprintf("%d\t%s\t", p->pid, p->name);
120         switch(p->state) {
121             case UNUSED:
122                 cprintf("UNUSED\n");
123                 break;
124             case EMBRYO:
125                 cprintf("EMBRYO\n");
126                 break;
127             case SLEEPING:
128                 cprintf("SLEEPING\n");
129                 break;
130             case RUNNABLE:
131                 cprintf("RUNNABLE\n");
132                 break;
133             case RUNNING:
134                 cprintf("RUNNING\n");
135                 break;
136             case ZOMBIE:
137                 cprintf("ZOMBIE\n");
138                 break;
139             default:
140                 cprintf("UNKNOWN\n");
141                 break;
142         }
143     }
144 }
```

Restricted Mode

File Edit Selection View Go ... ↶ ↷ Search Manage Learn More

Ln 158, Col 1 Spaces: 4 UTF-8 LF C

ENG IN 23:20 18-05-2024

```
Ubuntu > home > selvamjeeva > xv6-public > C sysproc.c

144 }
145
146 int
147 sys_nice(void)
148 {
149     int inc;
150     if(argint(0, &inc) < 0)
151         return -1;
152
153     struct proc *curproc = myproc();
154
155     if (inc < 0 || inc > 40) {
156         return -1;
157     }
158
159     curproc->quantum += inc;
160
161     return 0; // Return success
162 }
```

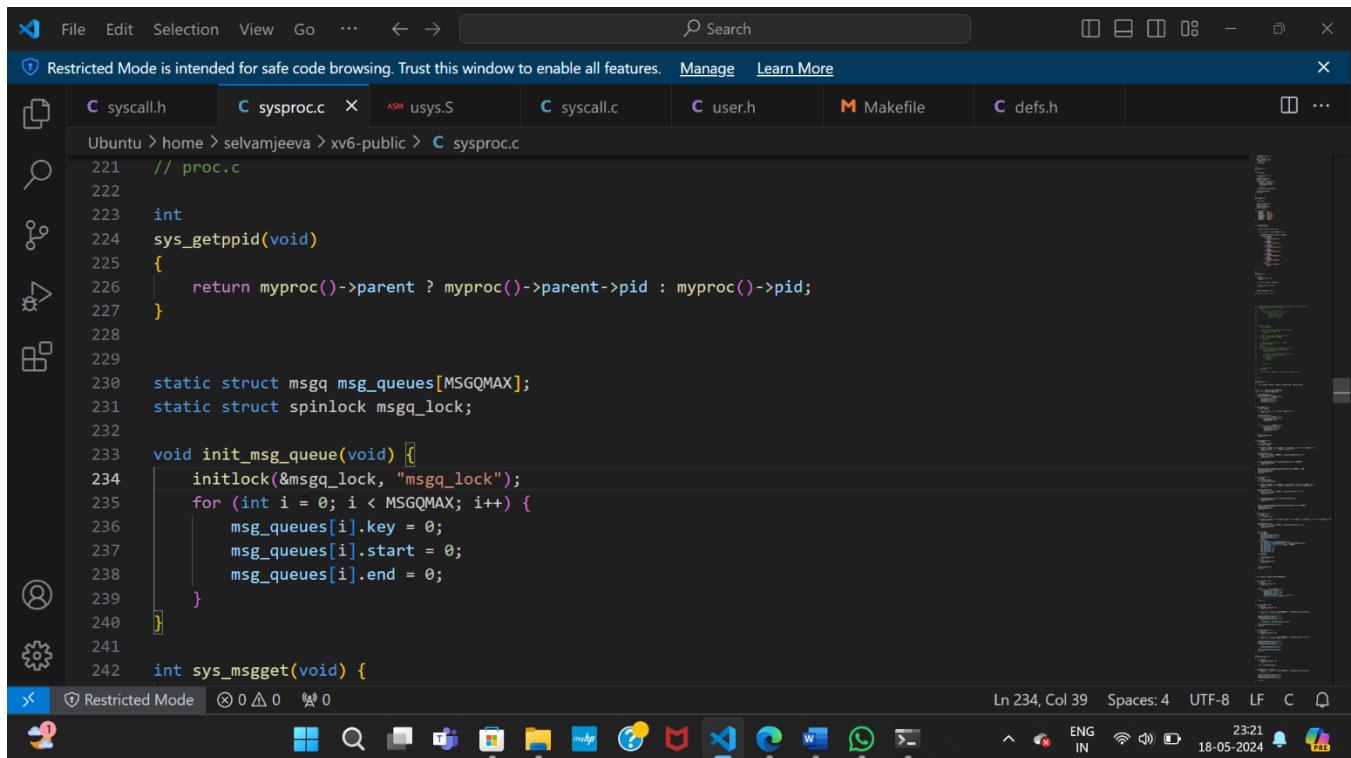
Restricted Mode

File Edit Selection View Go ... ↶ ↷ Search Manage Learn More

Ln 158, Col 1 Spaces: 4 UTF-8 LF C

ENG IN 23:20 18-05-2024

sysproc.c

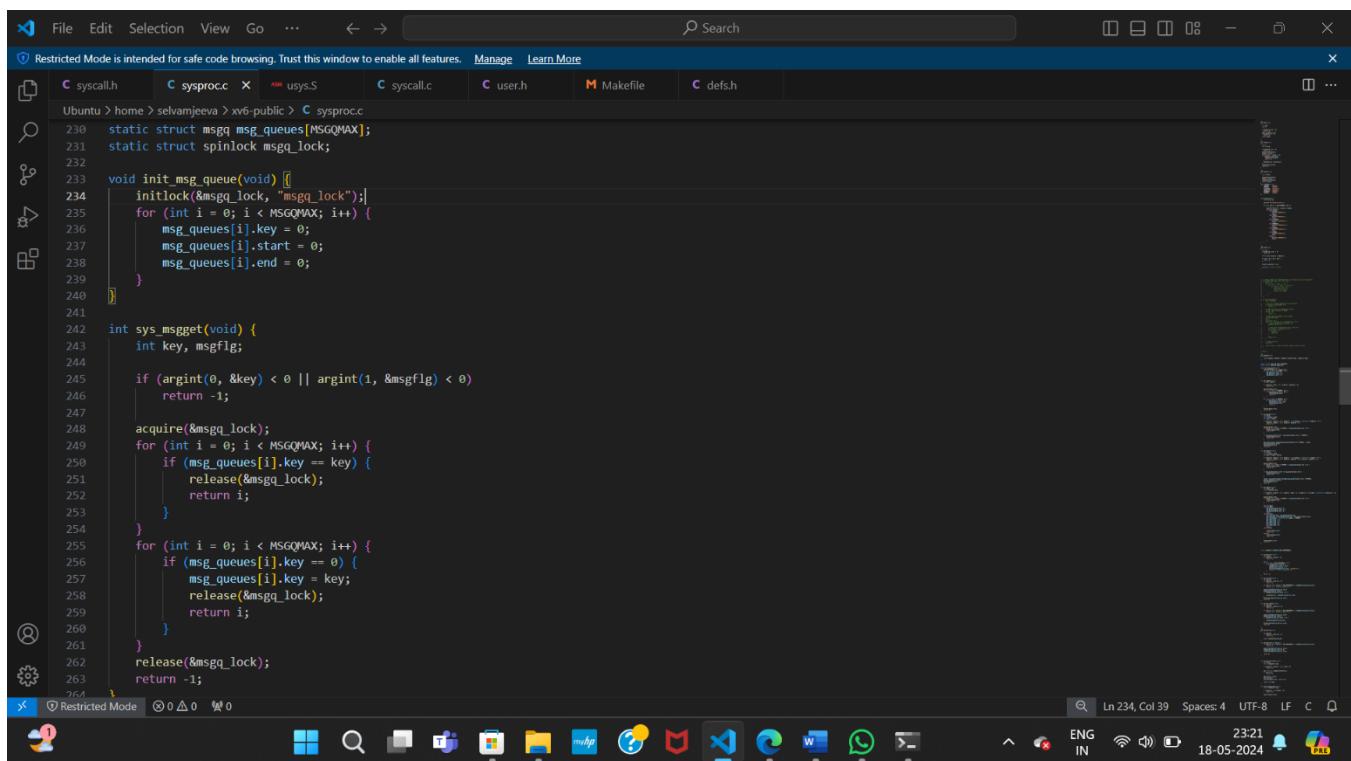


```
Ubuntu > home > selvamjeeva > xv6-public > C sysproc.c
221 // proc.c
222
223 int
224 sys_getppid(void)
225 {
226     return myproc()>parent ? myproc()>parent->pid : myproc()>pid;
227 }
228
229
230 static struct msgq msg_queues[MSGQMAX];
231 static struct spinlock msgq_lock;
232
233 void init_msg_queue(void) {
234     initlock(&msgq_lock, "msgq_lock");
235     for (int i = 0; i < MSGQMAX; i++) {
236         msg_queues[i].key = 0;
237         msg_queues[i].start = 0;
238         msg_queues[i].end = 0;
239     }
240 }
241
242 int sys_msget(void) {

```

Ln 234, Col 39 Spaces: 4 UTF-8 LF C

Restricted Mode 0 0 0 0 ENG IN 23:21 18-05-2024



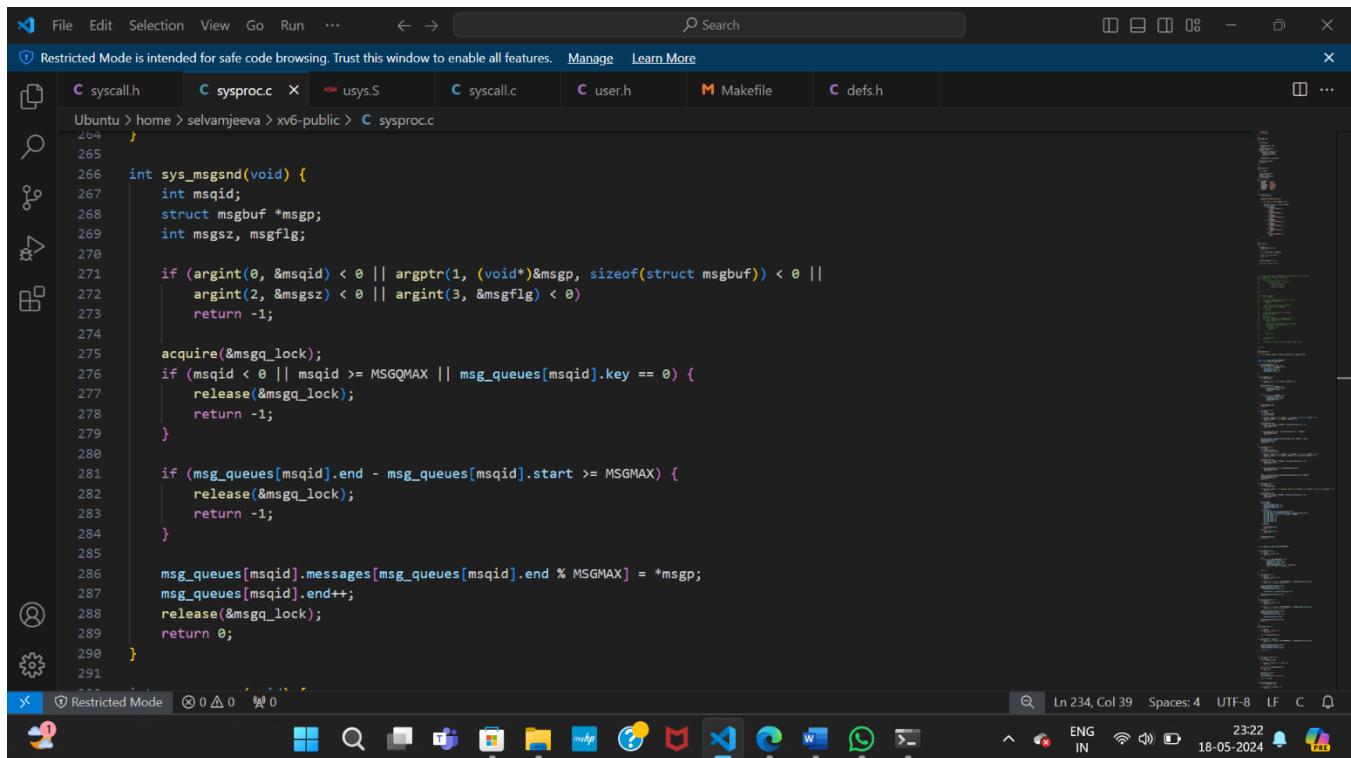
```
Ubuntu > home > selvamjeeva > xv6-public > C sysproc.c
230 static struct msgq msg_queues[MSGQMAX];
231 static struct spinlock msgq_lock;
232
233 void init_msg_queue(void) {
234     initlock(&msgq_lock, "msgq_lock");
235     for (int i = 0; i < MSGQMAX; i++) {
236         msg_queues[i].key = 0;
237         msg_queues[i].start = 0;
238         msg_queues[i].end = 0;
239     }
240 }
241
242 int sys_msget(void) {
243     int key, msgflg;
244
245     if (argint(0, &key) < 0 || argint(1, &msgflg) < 0)
246         return -1;
247
248     acquire(&msgq_lock);
249     for (int i = 0; i < MSGQMAX; i++) {
250         if (msg_queues[i].key == key) {
251             release(&msgq_lock);
252             return i;
253         }
254     }
255     for (int i = 0; i < MSGQMAX; i++) {
256         if (msg_queues[i].key == 0) {
257             msg_queues[i].key = key;
258             release(&msgq_lock);
259             return i;
260         }
261     }
262     release(&msgq_lock);
263     return -1;

```

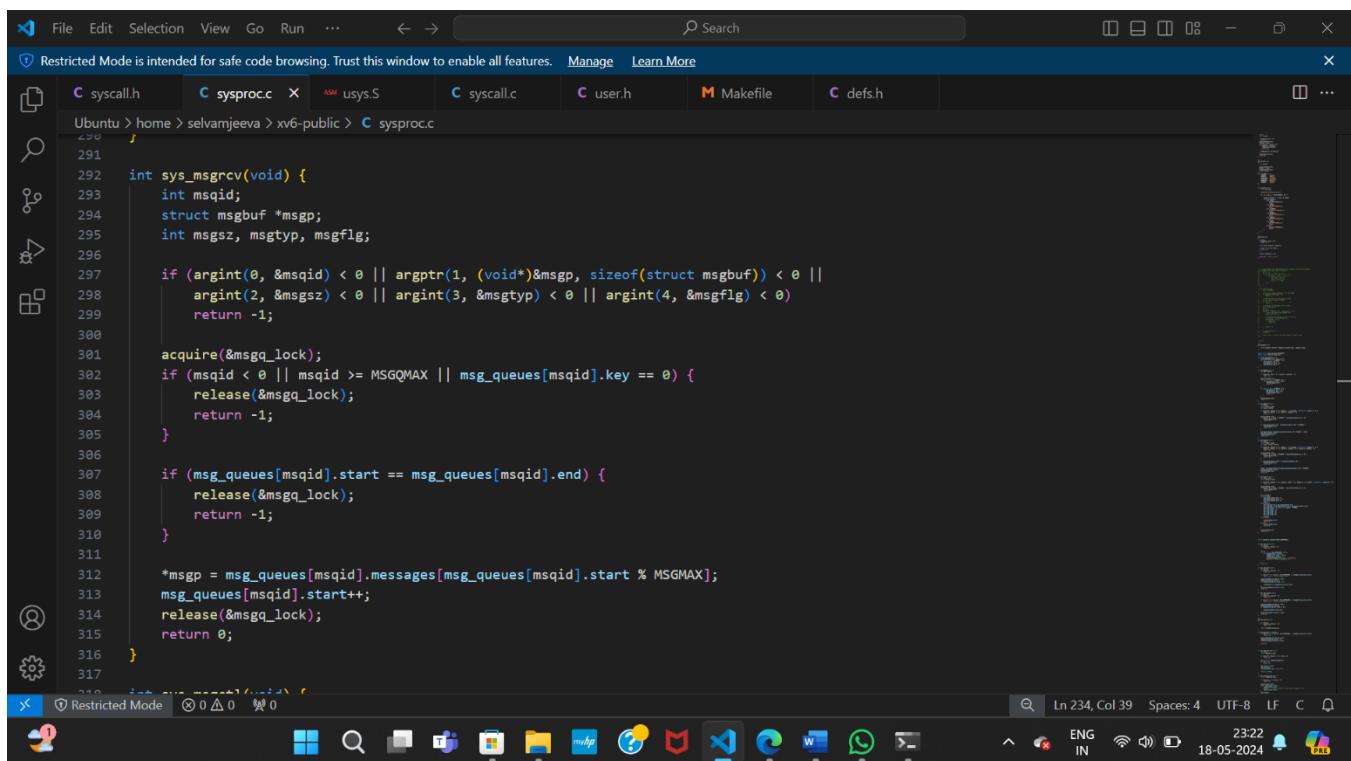
Ln 234, Col 39 Spaces: 4 UTF-8 LF C

Restricted Mode 0 0 0 0 ENG IN 23:21 18-05-2024

sysproc.c

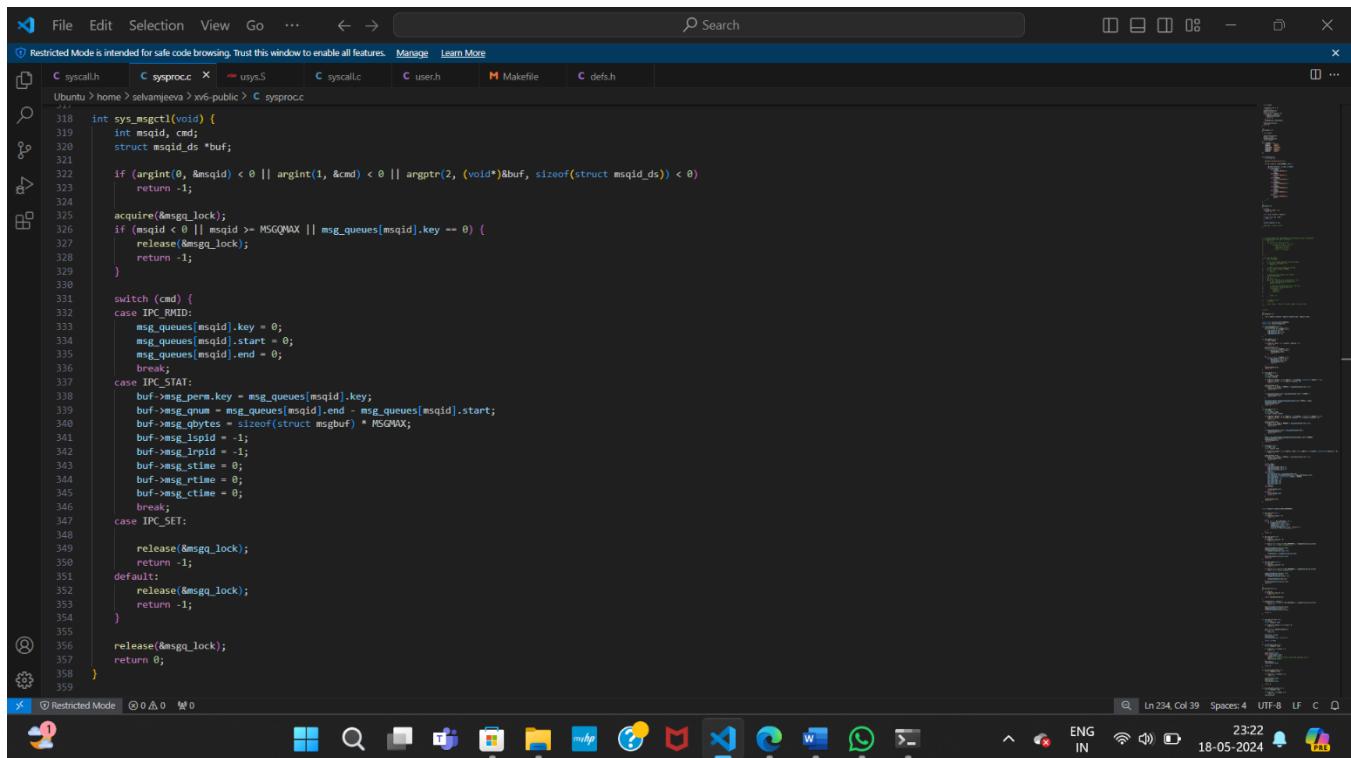


```
264 }
265
266 int sys_msqsnd(void) {
267     int msqid;
268     struct msgbuf *msgp;
269     int msgsz, msgflg;
270
271     if (argint(0, &msqid) < 0 || argptr(1, (void**)&msgp, sizeof(struct msgbuf)) < 0 ||
272         argint(2, &msgsz) < 0 || argint(3, &msgflg) < 0)
273         return -1;
274
275     acquire(&msgq_lock);
276     if (msqid < 0 || msqid >= MSGQMAX || msg_queues[msqid].key == 0) {
277         release(&msgq_lock);
278         return -1;
279     }
280
281     if (msg_queues[msqid].end - msg_queues[msqid].start >= MSGMAX) {
282         release(&msgq_lock);
283         return -1;
284     }
285
286     msg_queues[msqid].messages[msg_queues[msqid].end % MSGMAX] = *msgp;
287     msg_queues[msqid].end++;
288     release(&msgq_lock);
289     return 0;
290 }
```

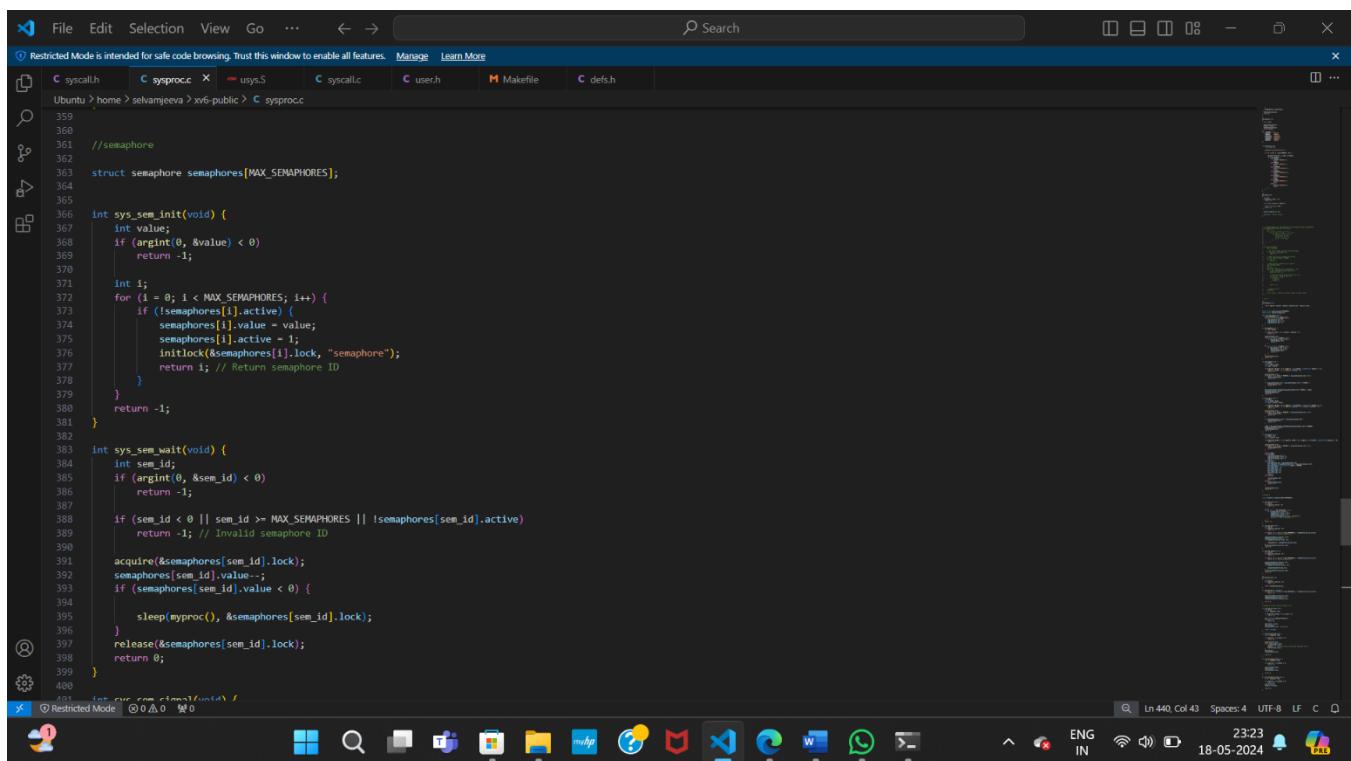


```
291 }
292
293 int sys_msgrcv(void) {
294     int msqid;
295     struct msgbuf *msgp;
296     int msgsz, msgtyp, msgflg;
297
298     if (argint(0, &msqid) < 0 || argptr(1, (void**)&msgp, sizeof(struct msgbuf)) < 0 ||
299         argint(2, &msgsz) < 0 || argint(3, &msgtyp) < 0 || argint(4, &msgflg) < 0)
300         return -1;
301
302     acquire(&msgq_lock);
303     if (msqid < 0 || msqid >= MSGQMAX || msg_queues[msqid].key == 0) {
304         release(&msgq_lock);
305         return -1;
306     }
307
308     if (msg_queues[msqid].start == msg_queues[msqid].end) {
309         release(&msgq_lock);
310         return -1;
311     }
312
313     *msgp = msg_queues[msqid].messages[msg_queues[msqid].start % MSGMAX];
314     msg_queues[msqid].start++;
315     release(&msgq_lock);
316     return 0;
317 }
```

sysproc.c



```
318 int sys_msget(void) {
319     int msqid, cmd;
320     struct msqid_ds *buf;
321
322     if (argint(0, &msqid) < 0 || argint(1, &cmd) < 0 || argptr(2, (void**)&buf, sizeof(struct msqid_ds)) < 0)
323         return -1;
324
325     acquire(&msgq_lock);
326     if (msqid < 0 || msqid >= MSGMAX || msg_queues[msqid].key == 0) {
327         release(&msgq_lock);
328         return -1;
329     }
330
331     switch (cmd) {
332     case IPC_RMID:
333         msg_queues[msqid].key = 0;
334         msg_queues[msqid].start = 0;
335         msg_queues[msqid].end = 0;
336         break;
337     case IPC_STAT:
338         buf->msg_perm.key = msg_queues[msqid].key;
339         buf->msg_qnum = msg_queues[msqid].end - msg_queues[msqid].start;
340         buf->msg_qbytes = sizeof(struct msgbuf) * MSGMAX;
341         buf->msg_lspid = -1;
342         buf->msg_lpid = -1;
343         buf->msg_stime = 0;
344         buf->msg_rtime = 0;
345         buf->msg_ctime = 0;
346         break;
347     case IPC_SET:
348         release(&msgq_lock);
349         return -1;
350     default:
351         release(&msgq_lock);
352         return -1;
353     }
354
355     release(&msgq_lock);
356     return 0;
357 }
358 }
```



```
359
360 //semaphore
361
362 struct semaphore semaphores[MAX_SEMAPHORES];
363
364
365 int sys_sem_init(void) {
366     int value;
367     if (argint(0, &value) < 0)
368         return -1;
369
370     int i;
371     for (i = 0; i < MAX_SEMAPHORES; i++) {
372         if (!semaphores[i].active) {
373             semaphores[i].value = value;
374             semaphores[i].active = 1;
375             initlock(&semaphores[i].lock, "semaphore");
376             return i; // Return semaphore ID
377         }
378     }
379     return -1;
380 }
381
382 int sys_sem_wake(void) {
383     int sem_id;
384     if (argint(0, &sem_id) < 0)
385         return -1;
386
387     if (sem_id < 0 || sem_id >= MAX_SEMAPHORES || !semaphores[sem_id].active)
388         return -1; // Invalid semaphore ID
389
390     acquire(&semaphores[sem_id].lock);
391     semaphores[sem_id].value++;
392     if (semaphores[sem_id].value < 0) {
393
394         sleep(myproc(), &semaphores[sem_id].lock);
395     }
396     release(&semaphores[sem_id].lock);
397     return 0;
398 }
399 }
```

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;
```

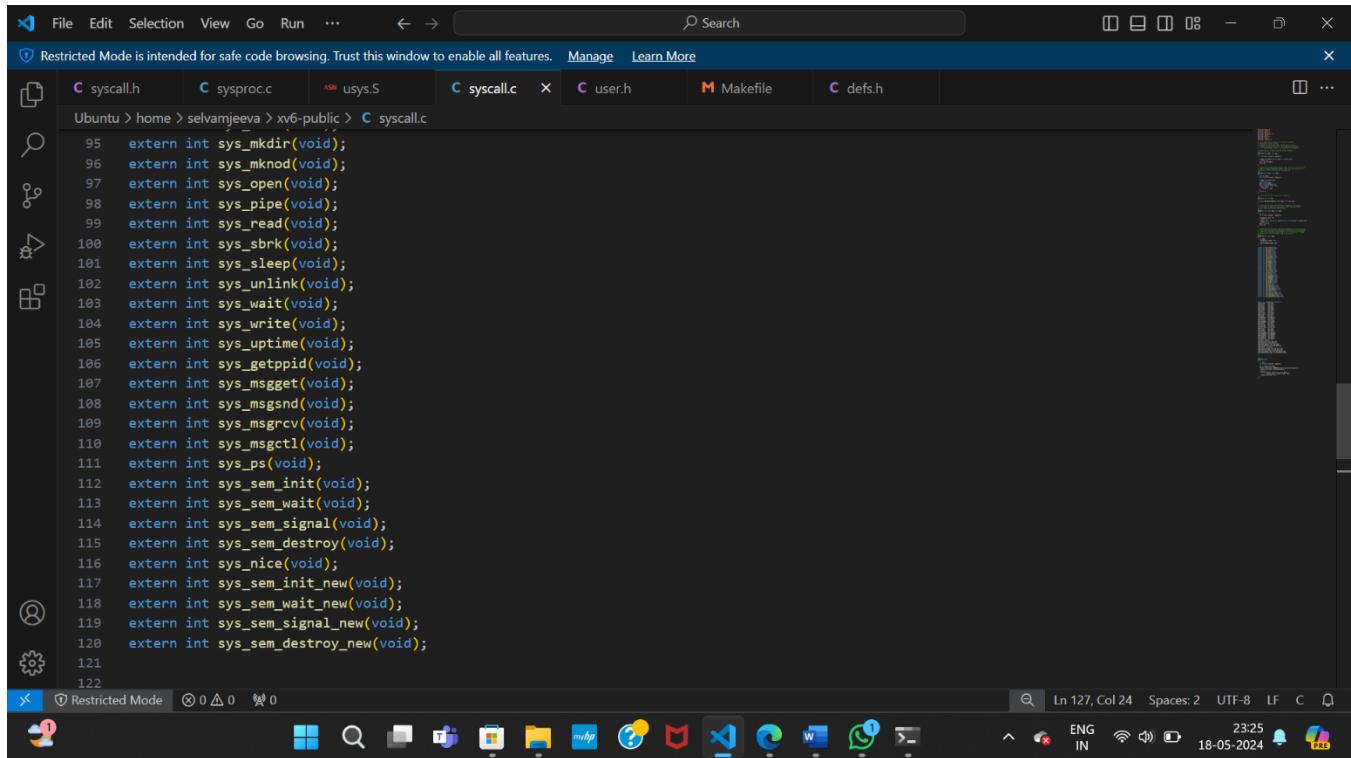
sysproc.c

```
470 int sys_sem_signal_new(void) {
471     struct semaphore *sem;
472
473     if (argint(0, (int*)&sem) < 0)
474         return -1;
475
476     acquire(&sem->lock);
477     sem->value++;
478     release(&sem->lock);
479
480     return 0;
481 }
482
483 int sys_sem_destroy_new(void) {
484     struct semaphore *sem;
485
486     if (argint(0, (int*)&sem) < 0)
487         return -1;
488     sem->active=0;
489     kfree((char*)sem);
490
491     return 0;
492 }
```

Usys.S :

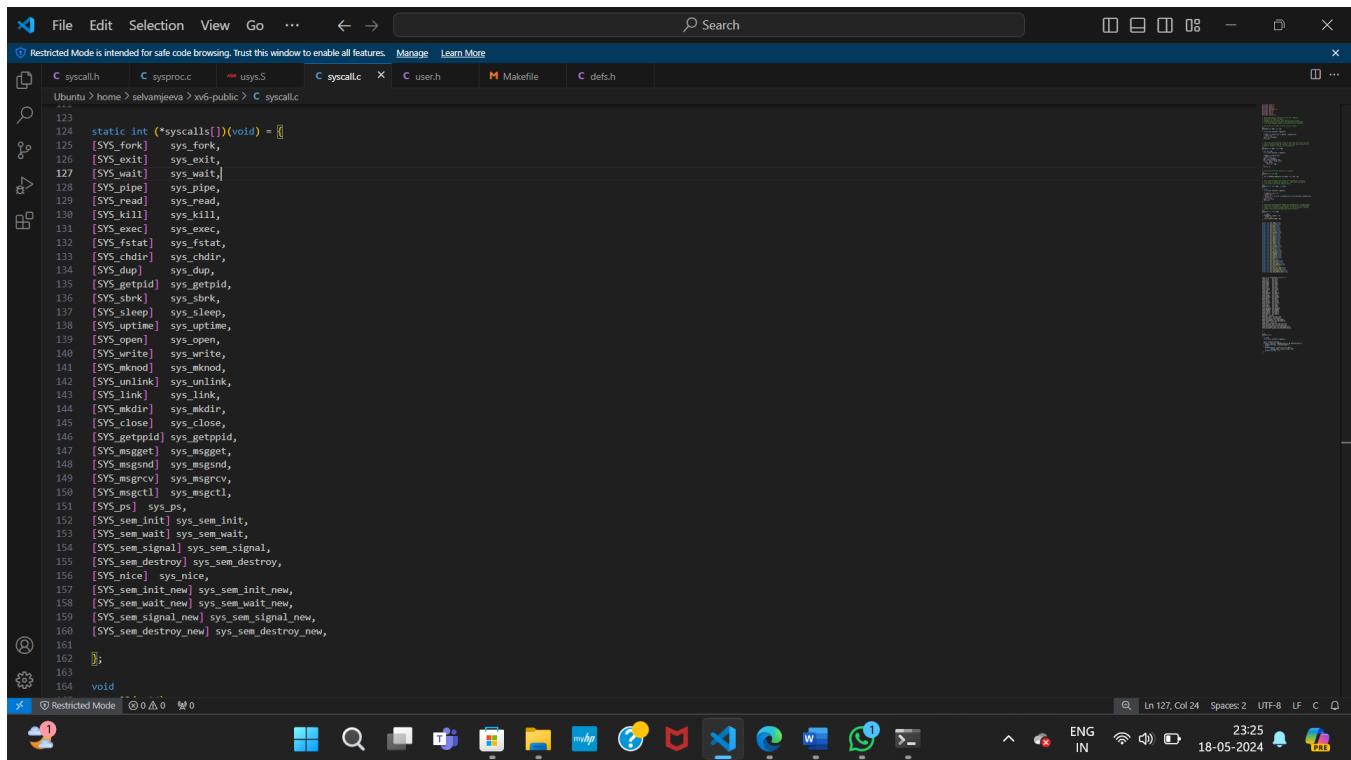
```
21 SYSCALL(mknod)
22 SYSCALL(unlink)
23 SYSCALL(fstat)
24 SYSCALL(link)
25 SYSCALL(mkdir)
26 SYSCALL(chdir)
27 SYSCALL(dup)
28 SYSCALL(getpid)
29 SYSCALL(sbrk)
30 SYSCALL(sleep)
31 SYSCALL(uptime)
32 SYSCALL(getppid)
33 SYSCALL(msqget)
34 SYSCALL(msqsnd)
35 SYSCALL(msgrcv)
36 SYSCALL(msqctl)
37 SYSCALL(ps)
38 SYSCALL(sem_init)
39 SYSCALL(sem_wait)
40 SYSCALL(sem_signal)
41 SYSCALL(sem_destroy)
42 SYSCALL(nice)
43 SYSCALL(sem_init_new)
44 SYSCALL(sem_wait_new)
45 SYSCALL(sem_signal_new)
46 SYSCALL(sem_destroy_new)
47
48
```

Syscall.c :



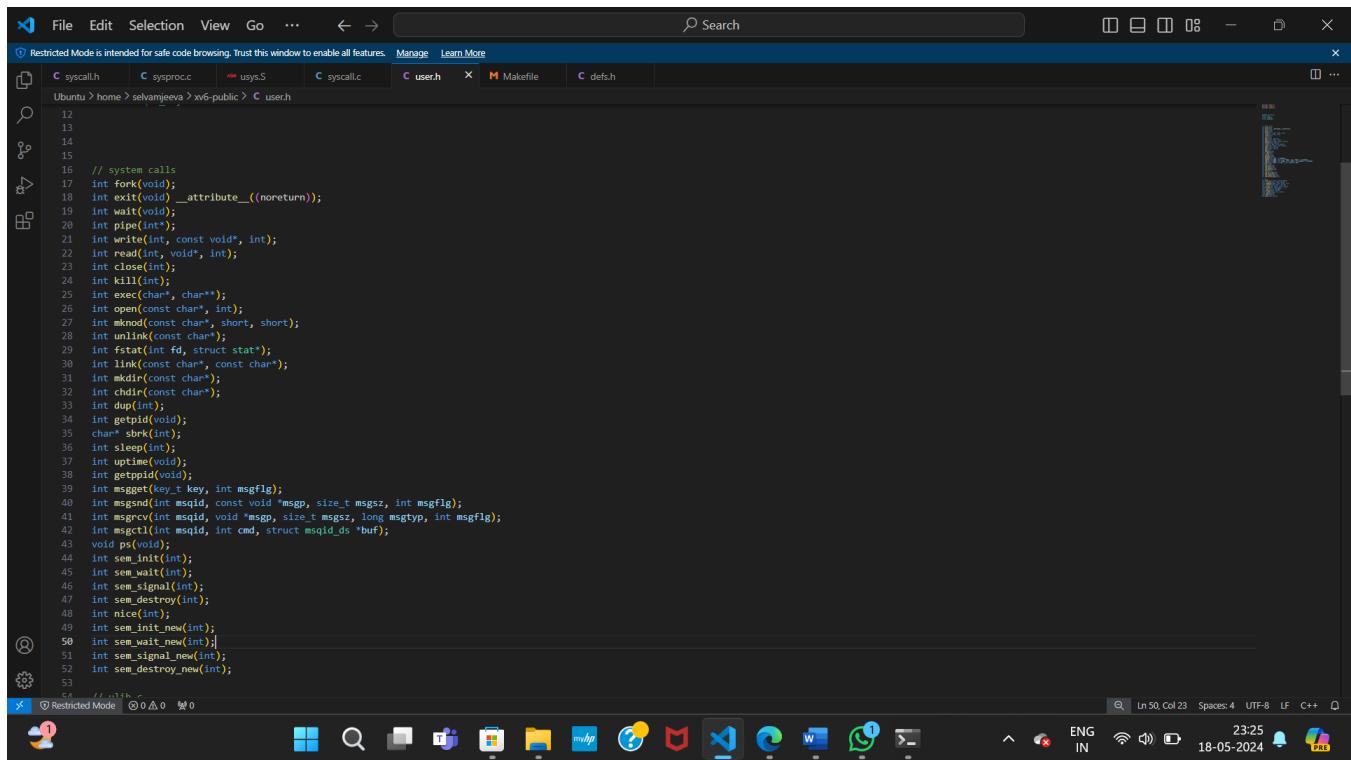
```
95 extern int sys_mkdir(void);
96 extern int sys_mknod(void);
97 extern int sys_open(void);
98 extern int sys_pipe(void);
99 extern int sys_read(void);
100 extern int sys_sbrk(void);
101 extern int sys_sleep(void);
102 extern int sys_unlink(void);
103 extern int sys_wait(void);
104 extern int sys_write(void);
105 extern int sys_uptime(void);
106 extern int sys_getppid(void);
107 extern int sys_msget(void);
108 extern int sys_msgsnd(void);
109 extern int sys_msgrcv(void);
110 extern int sys_msctl(void);
111 extern int sys_ps(void);
112 extern int sys_sem_init(void);
113 extern int sys_sem_wait(void);
114 extern int sys_sem_signal(void);
115 extern int sys_sem_destroy(void);
116 extern int sys_nice(void);
117 extern int sys_sem_init_new(void);
118 extern int sys_sem_wait_new(void);
119 extern int sys_sem_signal_new(void);
120 extern int sys_sem_destroy_new(void);
121
122
```

Syscall.c :



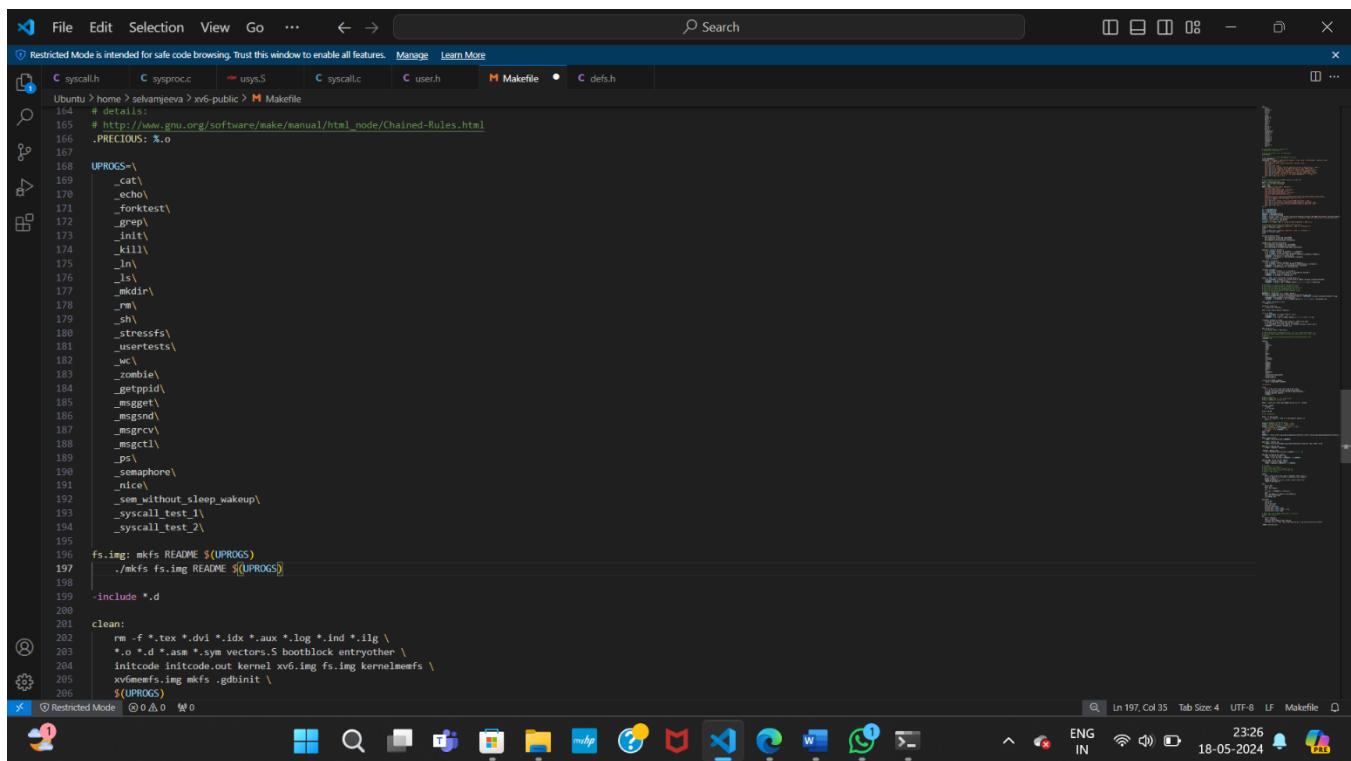
```
123 static int (*syscalls[])(void) = [
124     [SYS_fork] sys_fork,
125     [SYS_exit] sys_exit,
126     [SYS_wait] sys_wait,
127     [SYS_pipe] sys_pipe,
128     [SYS_read] sys_read,
129     [SYS_kill] sys_kill,
130     [SYS_exec] sys_exec,
131     [SYS_fstat] sys_fstat,
132     [SYS_chdir] sys_chdir,
133     [SYS_dup] sys_dup,
134     [SYS_getpid] sys_getpid,
135     [SYS_sbrk] sys_sbrk,
136     [SYS_sleep] sys_sleep,
137     [SYS_uptime] sys_uptime,
138     [SYS_open] sys_open,
139     [SYS_write] sys_write,
140     [SYS_mknod] sys_mknod,
141     [SYS_unlink] sys_unlink,
142     [SYS_link] sys_link,
143     [SYS_mkdir] sys_mkdir,
144     [SYS_close] sys_close,
145     [SYS_getpid] sys_getpid,
146     [SYS_msget] sys_msget,
147     [SYS_msgsnd] sys_msgsnd,
148     [SYS_msgrcv] sys_msgrcv,
149     [SYS_msctl] sys_msctl,
150     [SYS_ps] sys_ps,
151     [SYS_sem_init] sys_sem_init,
152     [SYS_sem_wait] sys_sem_wait,
153     [SYS_sem_signal] sys_sem_signal,
154     [SYS_sem_destroy] sys_sem_destroy,
155     [SYS_nice] sys_nice,
156     [SYS_sem_init_new] sys_sem_init_new,
157     [SYS_sem_wait_new] sys_sem_wait_new,
158     [SYS_sem_signal_new] sys_sem_signal_new,
159     [SYS_sem_destroy_new] sys_sem_destroy_new,
160 ];
161
162 ];
163
164 void
```

User.h :



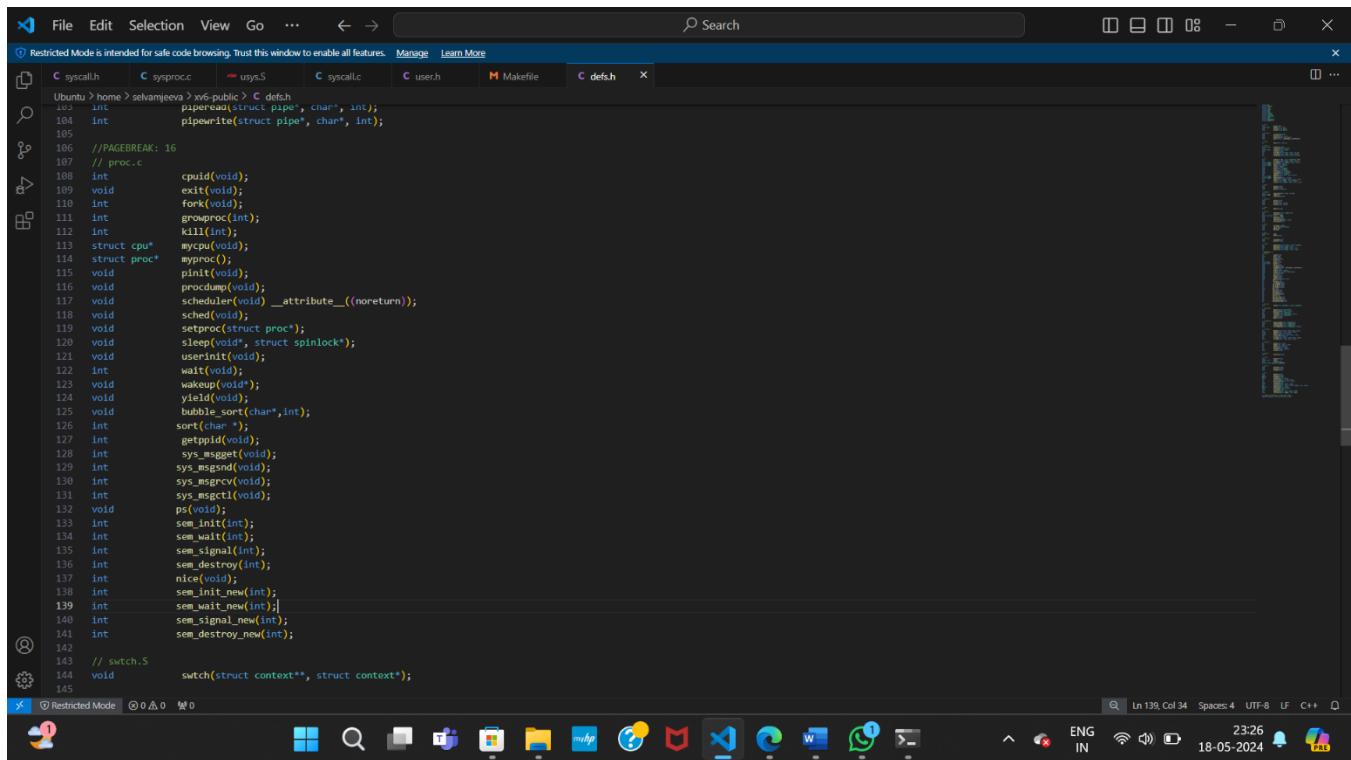
```
12
13
14
15
16 // system calls
17 int fork(void);
18 int exit(void) __attribute__((noreturn));
19 int wait(void);
20 int sleep(int);
21 int write(int, const void*, int);
22 int read(int, void*, int);
23 int close(int);
24 int kill(int);
25 int exec(char*, char**);
26 int open(const char*, int);
27 int mknod(const char*, short, short);
28 int unlink(const char*);
29 int fstat(int fd, struct stat*);
30 int link(const char*, const char*);
31 int mkdir(const char*);
32 int chdir(const char*);
33 int dup(int);
34 int getpid(void);
35 char* sbrk(int);
36 int sleep(int);
37 int uptime(void);
38 int getpid(void);
39 int msgget(key_t key, int msgflg);
40 int msgsnd(int msgid, const void *msgp, size_t msgsz, int msgflg);
41 int msgrcv(int msgid, void *msgp, size_t msgtyp, int msgflg);
42 int msgctl(int msgid, int cmd, struct msqid_ds *buf);
43 void ps(void);
44 int sem_init(int);
45 int sem_wait(int);
46 int sem_signal(int);
47 int sem_destroy(int);
48 int nice(int);
49 int sem_init_new(int);
50 int sem_wait_new(int);
51 int sem_signal_new(int);
52 int sem_destroy_new(int);
53
54 // ... other declarations ...
55
56 Restricted Mode ① 0 △ 0 ⚡ 0
```

Makefile :



```
164 # details:
165 # http://www.gnu.org/software/make/manual/html_node/Chained-Rules.html
166 .PRECIOUS: %.o
167
168 UPROGS=\
169     _cat\
170     _echo\
171     _forktest\
172     _grep\
173     _init\
174     _kill\
175     _ln\
176     _ls\
177     _mkdir\
178     _rm\
179     _sh\
180     _stressfs\
181     _usertests\
182     _wc\
183     _zombie\
184     _getpid\
185     _msgget\
186     _msgsnd\
187     _msgrcv\
188     _msgctl\
189     _ps\
190     _semaphore\
191     _nice\
192     _sem_without_sleep_wakeup\
193     _syscall_test_1\
194     _syscall_test_2\
195
196 fs.img: mkfs README ${UPROGS}
197     ./mkfs fs.img README ${UPROGS}
198
199 -include .d
200
201 clean:
202     rm -f *.tex *.dvi *.idx *.aux *.log *.ind *.ilg \
203         *.o *.d *.asm *.sym vectors.S bootblock entryother \
204         initcode initcode.out kernel xv6.img fs.img kernelmemfs \
205         xv6memfs.img mkfs .gdbinit \
206         ${UPROGS}
207
208 Restricted Mode ① 0 △ 0 ⚡ 0
```

defs.h:



```
Ubuntu > home > selvangeeja > x86-public > C defs.h
103 int         piperead(struct pipe*, char*, int);
104 int         pipewrite(struct pipe*, char*, int);
105
106 //PAGEBREAK: 16
107 // proc.c
108 int         cpuid(void);
109 void        exit(void);
110 int         fork(void);
111 int         growproc(int);
112 int         kill(int);
113 struct cpu* mycpu();
114 struct proc* myproc();
115 void        print(void);
116 void        pswddump(void);
117 void        scheduler(void) __attribute__((noreturn));
118 void        sched(void);
119 void        setproc(struct proc*);
120 void        sleep(void*, struct spinlock*);
121 void        userinit(void);
122 int         wait(void);
123 void        wakeup(void*);
124 void        yield(void);
125 void        bubble_sort(char*,int);
126 int         sort(char *);
127 int         getpid(void);
128 int         sys_msget(void);
129 int         sys_msqnd(void);
130 int         sys_msqry(void);
131 int         sys_msctl(void);
132 void        ps(void);
133 int         sem_init(int);
134 int         sem_wait(int);
135 int         sem_signal(int);
136 int         sem_destroy(int);
137 int         nice(void);
138 int         sem_init_new(int);
139 int         sem_wait_new(int);
140 int         sem_signal_new(int);
141 int         sem_destroy_new(int);
142
143 // swtch.S
144 void        switch(struct context**, struct context*);
```