XINU Problems

Problem 1: Semaphore

```
#include <xinu.h>

int n = 0;

void producer(sid32 prod,sid32 cons)
{
    while (n <= 5) {
        wait(cons);
        n++;
        signal(prod);
    }
}
void consumer(sid32 prod,sid32 cons)
{
    while (n <= 5) {
        wait(prod);
        printf("%d\n", n);
        signal(cons);
    }
}
```

```
int hello1()
{
    sid32 prod = semcreate(1);
    sid32 cons  = semcreate(0);

    resume(create(producer, 1024, 20,
"producer", 2, prod, cons));
    resume(create(consumer, 1024, 20,
"consumer", 2, prod, cons));

    return 0;
}
```

Terminal Output:

```
xsh $ p1
0
xsh $ 1
2
3
4
5
```

Problem 2: Semaphore

```
#include <xinu.h>
```

```
int n = 5;

void producer(sid32 prod,sid32 cons)
{
    while (n >= 0) {
        wait(cons);
        printf("prod:%d\n",n--);
        signal(prod);
    }
    printf("Done - produced\n");
}
void consumer(sid32 prod,sid32 cons)
{
    while (n >= 0) {
        wait(prod);
        printf("cons:%d\n",n--);
        signal(cons);
    }
    printf("Done - consumed\n");
}

int hello1()
{
    sid32 prod = semcreate(1);
```

```
    sid32 cons  = semcreate(0);

    resume(create(producer, 1024, 20,
"producer", 2, prod, cons));
    resume(create(consumer, 1024, 20,
"consumer", 2, prod, cons));

    return 0;
}
```

Terminal Output:

```
Welcome to Xinu!


xsh $ p1
cons:5
xsh $ prod:4
cons:3
prod:2
cons:1
prod:0
Done - produced
Done - consumed
```

Problem 3: Semaphore

```
#include <xinu.h>

int n = 5;

void producer(sid32 prod,sid32 cons)
{
    while (n >= 0) {
        wait(cons);
        printf("prod:%d\n",n--);
        signal(prod);
    }
    printf("Done - produced\n");
}
void consumer(sid32 prod,sid32 cons)
{
    while (n >= 0) {
        wait(prod);
        printf("cons:%d\n",n--);
        signal(cons);
    }
    printf("Done - consumed\n");
}
```

```
int hello1()
{
    sid32 prod = semcreate(1);
    sid32 cons  = semcreate(0);

    resume(create(consumer, 1024, 20,
"consumer", 2, prod, cons));
    resume(create(producer, 1024, 20,
"producer", 2, prod, cons));

    return 0;
}
```

Terminal Output:

Problem 4: Semaphore
Solution of the error of Problem 3

```
#include <xinu.h>

int n = 5;

void producer(sid32 prod,sid32 cons)
{
    while (n >= 0) {
        wait(cons);
        printf("prod:%d\n",n--);
        signal(prod);
    }
    printf("Done - produced\n");
}
void consumer(sid32 prod,sid32 cons)
{
    while (n >= 0) {
        wait(prod);
        if(n<0){
            signal(cons);
            break;
        }
        printf("cons:%d\n",n--);
```

```
        signal(cons);
    }
    printf("Done - consumed\n");
}

int hello1()
{
    sid32 prod = semcreate(1);
    sid32 cons  = semcreate(0);

    resume(create(consumer, 1024, 20,
"consumer", 2, prod, cons));
    resume(create(producer, 1024, 20,
"producer", 2, prod, cons));

    return 0;
}
```

Terminal Output:

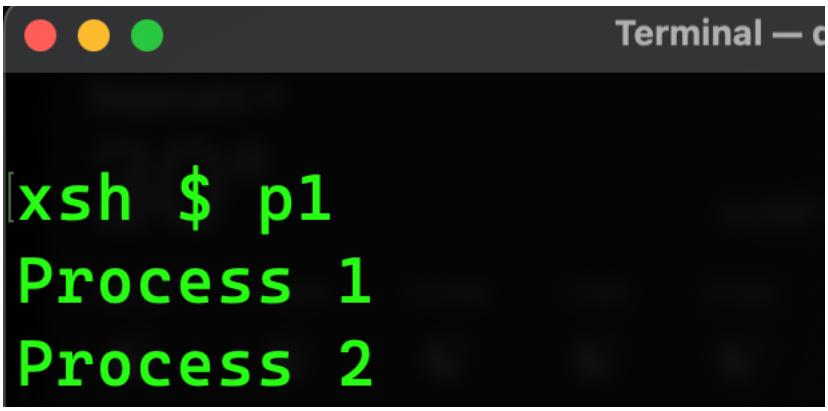
```
xsh $ p1
cons:5
prod:4
xsh $ cons:3
prod:2
cons:1
prod:0
Done - consumed
Done - produced
```

Problem 5: ReadyList

```c
#include <xinu.h>

void proc3()
{
    printf("Process 1\n");
}
void proc4()
{
    printf("Process 2\n");
}
```

```
void PrintReady()
{
    qid16 head = queuehead(readylist);
    qid16 tail = head + 1;

    while (head != tail)
    {
        struct qentry curr = queuetab[head];

printf("id=%d,key=%d,prev=%d,next=%d\n",
head, curr.qkey, curr.qprev, curr.qnext);
        head = curr.qnext;
    }
}
int hello1()
{
    resume(create(proc3, 1024, 20, "proc3",
0));
    resume(create(proc4, 1024, 20, "proc4",
0));

    return 0;
}
```

Terminal Output:



Problem 6: Ready Process Info from Proctab

```c
#include <xinu.h>

void proc3()
{
    printf("Proces 3 - %d\n",getpid());

    while(1){
    }
}
void proc4()
{
    printf("Process 4 - %d\n",getpid());
    while (1)
    {
    }
}
```

```
void PrintReady()
{
    qid16 head = queuehead(readylist);
    qid16 tail = head + 1;

    printf("Head=%d, Tail=%d\n", head, tail);

    while (head != tail)
    {
        struct qentry curr = queuetab[head];
        struct procent process = proctab[head];
        printf("id=%d,prio=%d,stklen=%d,name=%s,
semid=%d,parent=%d\n", head, process.prprio,
process.prstklen, process.prname, process.prsem,
process.prparent);
        head = curr.qnext;
    }
}
int LF()
{
    resume(create(proc3, 1024, 20, "proc3", 0));
    resume(create(proc4, 1024, 20, "proc4", 0));
    resume(create(PrintReady, 1024, 20,
"readyPrinter", 0));

    return 0;
}
```

Terminal Output:

```
xsh $ LF
Proces 3 - 5
Process 4 - 6
Head=300, Tail=301
id=300,prio=0,stklen=0,name=, semid=0,parent=0
id=4,prio=20,stklen=8192,name=LF, semid=-1,parent=3
id=5,prio=20,stklen=1024,name=proc3, semid=-1,parent=4
id=6,prio=20,stklen=1024,name=proc4, semid=-1,parent=4
id=0,prio=0,stklen=8192,name=prnull, semid=0,parent=0
```

Problem 7: WaitList

```c
#include <xinu.h>

void proc3(sid32 sem)
{
    printf("Proces 3 - %d\n", getpid());
    wait(sem);
    while (1)
    {
    }
}
void proc4(sid32 sem)
{
    printf("Process 4 - %d\n", getpid());
    wait(sem);
    while (1)
```

```
    {
    }
}

void printer()
{
    for (int i = 0; i < NPROC; i++)
    {
        struct procent curr = proctab[i];
        if (curr.prstate == PR_WAIT)
        {
            printf("pid-%d\n",i);
        }
    }
}
int LF()
{
    sid32 sem = semcreate(0);

    resume(create(proc3, 1024, 20, "proc3",
1,sem));
    resume(create(proc4, 1024, 20, "proc4",
1,sem));
```

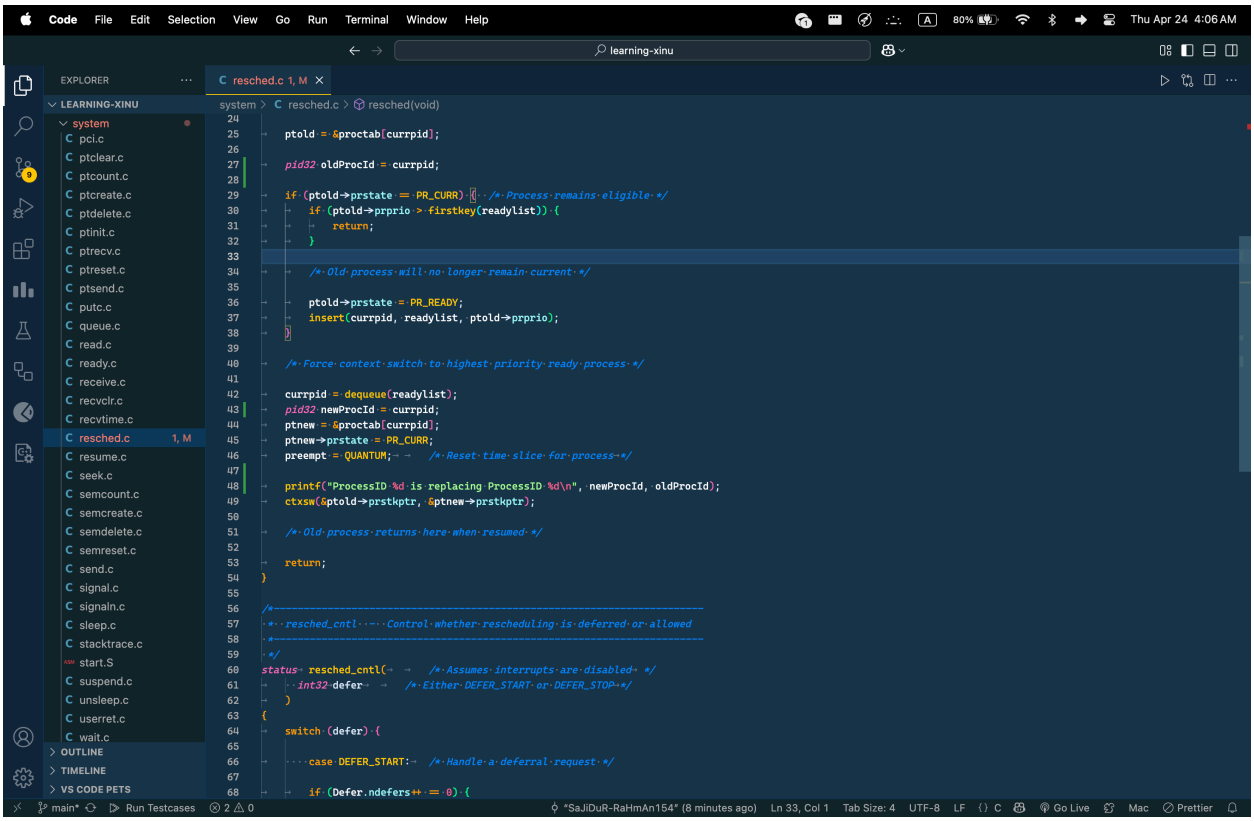
```
    resume(create(printer, 1024, 20,
"readyPrinter", 0));

    return 0;
}
```

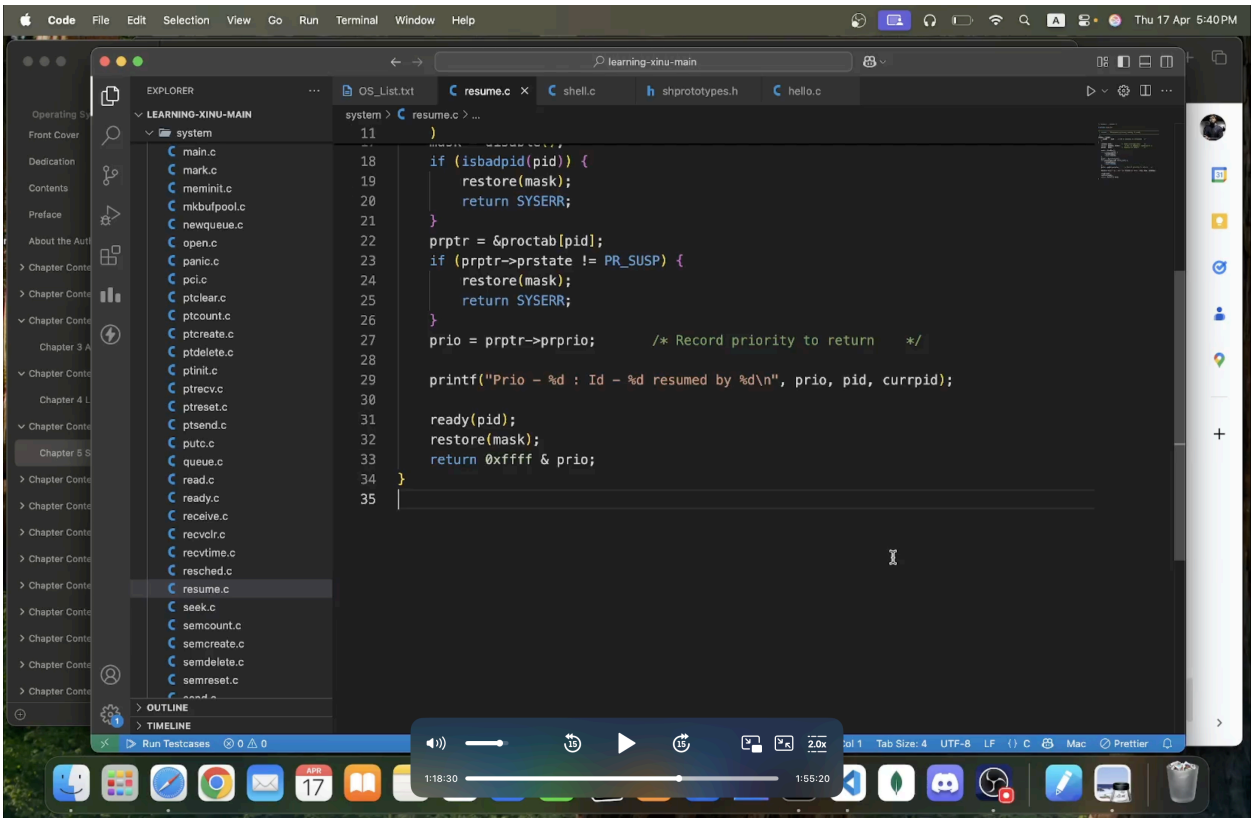
Terminal Output:

```
xsh $ LF
Proces 3 - 6
Process 4 - 7
pid-6
pid-7
```

Problem 8,9: Rescheduling process and Printing the Prio, Resumed process ID, and Resumed By process ID

Terminal Output:


Code File Edit Selection View Go Run Terminal Window Help
Thu Apr 24 4:06 AM
learning-xinu
EXPLORER
resched.c 1, M
LEARNING-XINU
system
pci.c
ptclear.c
ptcount.c
ptcreate.c
ptdelete.c
ptinit.c
ptrecv.c
ptreset.c
ptsend.c
putc.c
queue.c
read.c
ready.c
receive.c
recvclr.c
recvtime.c
resched.c 1, M
resume.c
seek.c
semcount.c
semcreate.c
semdelete.c
semreset.c
send.c
signal.c
signaln.c
sleep.c
stacktrace.c
start.S
suspend.c
unsleep.c
userret.c
wait.c
OUTLINE
TIMELINE
VS CODE PETS
system > resched.c > resched(void)
ptold = &proctab[currpid];
pid32 oldProcId = currpid;
if (ptold->prstate == PR_CURR) { /* Process remains eligible */
if (ptold->prprio > firstkey(readylist)) {
return;
}
/* Old process will no longer remain current */
ptold->prstate = PR_READY;
insert(currpid, readylist, ptold->prprio);
}
/* Force context switch to highest priority ready process */
currpid = dequeue(readylist);
pid32 newProcId = currpid;
ptnew = &proctab[currpid];
ptnew->prstate = PR_CURR;
preempt = QUANTUM; /* Reset time slice for process */
printf("ProcessID %d is replacing ProcessID %d\n", newProcId, oldProcId);
ctxsw(&ptold->prstkptr, &ptnew->prstkptr);
/* Old process returns here when resumed */
return;
}
/*--
* resched_cntl - Control whether rescheduling is deferred or allowed
*--
*/
status resched_cntl(/* Assumes interrupts are disabled */
int32 defer /* Either DEFER_START or DEFER_STOP*/
)
{
switch (defer) {
case DEFER_START: /* Handle a deferral request */
if (Defer.ndefers++ == 0) {
main* Run Testcases 2 0
SaJiDuR-RaHmAn154 (8 minutes ago)
Ln 33, Col 1
Tab Size: 4
UTF-8
LF
C
Go Live
Mac
Prettier


Code File Edit Selection View Go Run Terminal Window Help
Thu 17 Apr 5:40 PM
learning-xinu-main
EXPLORER
OS_List.txt
resume.c
shell.c
shprototypes.h
hello.c
LEARNING-XINU-MAIN
system
main.c
mark.c
meminit.c
mkbufpool.c
newqueue.c
open.c
panic.c
pci.c
ptclear.c
ptcount.c
ptcreate.c
ptdelete.c
ptinit.c
ptrecv.c
ptreset.c
ptsend.c
putc.c
queue.c
read.c
ready.c
receive.c
recvclr.c
recvtime.c
resched.c
resume.c
seek.c
semcount.c
semcreate.c
semdelete.c
semreset.c
OUTLINE
TIMELINE
system > resume.c > ...
)
if (isbadpid(pid)) {
restore(mask);
return SYSERR;
}
prptr = &proctab[pid];
if (prptr->prstate != PR_SUSP) {
restore(mask);
return SYSERR;
}
prio = prptr->prprio; /* Record priority to return */
printf("Prio - %d : Id - %d resumed by %d\n", prio, pid, currpid);
ready(pid);
restore(mask);
return 0xffff & prio;
}
Run Testcases
Tab Size: 4
UTF-8
LF
C
Mac
Prettier
1:18:30
1:55:20

Problem 10: Accessing semtab

```c
#include<xinu.h>

void proc1(){
    printf("Process 1 : %d\n", getpid());
    while (1){
    }

}

void proc2(){
    printf("Process 2 : %d\n", getpid());
    while (1){
    }
}

void Printer(){
    struct sentry curr;

    for(int i = 0; i < NSEM; i++){
        curr = semtab[i];
        if(curr.sstate == S_USED){
            printf("Semid = %d, count = %d\n", i, curr.scount);
        }
    }
}
int hello1(){

    sid32 sem1 = semcreate(0);
    sid32 sem2 = semcreate(1);
    sid32 sem3 = semcreate(2);
    sid32 sem4 = semcreate(3);

    printf("Newly created semaphore ids - %d, %d, %d, %d\n", sem1, sem2, sem3, sem4);

    resume(create(proc1, 1024, 20, "proc1", 0));
    resume(create(proc2, 1024, 20, "proc2", 0));
    resume(create(Printer, 1024, 20, "Printer", 0));
    return 0;
}
```

Terminal Output:

```
xsh $ p1
Newly created semaphore ids - 10, 11, 12, 13
Process 1 : 7
Process 2 : 8
Semid = 0, count = 0
Semid = 1, count = 64
Semid = 2, count = 1
Semid = 3, count = 1
Semid = 4, count = 1
Semid = 5, count = 1
Semid = 6, count = 1
Semid = 7, count = 1
Semid = 8, count = 1
Semid = 9, count = 1
Semid = 10, count = 0
Semid = 11, count = 1
Semid = 12, count = 2
Semid = 13, count = 3
```

Problem 11: semdelete

```c
#include<xinu.h>

void proc1(){
    printf("Process 1 : %d\n", getpid());
    while (1){
    }

}

void proc2(){
    printf("Process 2 : %d\n", getpid());
    while (1){
    }
}

void Printer(){
    struct sentry curr;
```

```
    for(int i = 0; i < NSEM; i++){
        curr = semtab[i];
        if(curr.sstate == S_USED){
            printf("Semid = %d, count = %d\n", i, curr.scount);
        }
    }
}

int hello1(){

    sid32 sem1 = semcreate(0);
    sid32 sem2 = semcreate(1);
    sid32 sem3 = semcreate(2);
    sid32 sem4 = semcreate(3);

    printf("Newly created semaphore ids - %d, %d, %d, %d\n", sem1, sem2, sem3, sem4);

    resume(create(proc1, 1024, 20, "proc1", 0));
    resume(create(proc2, 1024, 20, "proc2", 0));
    resume(create(Printer, 1024, 20, "Printer", 0));

    semdelete(sem3);
    semdelete(sem4);

    printf("After deleting sem3 and sem4\n");
    resume(create(Printer, 1024, 20, "Printer", 0));

    return 0;
}
```

Terminal Output:

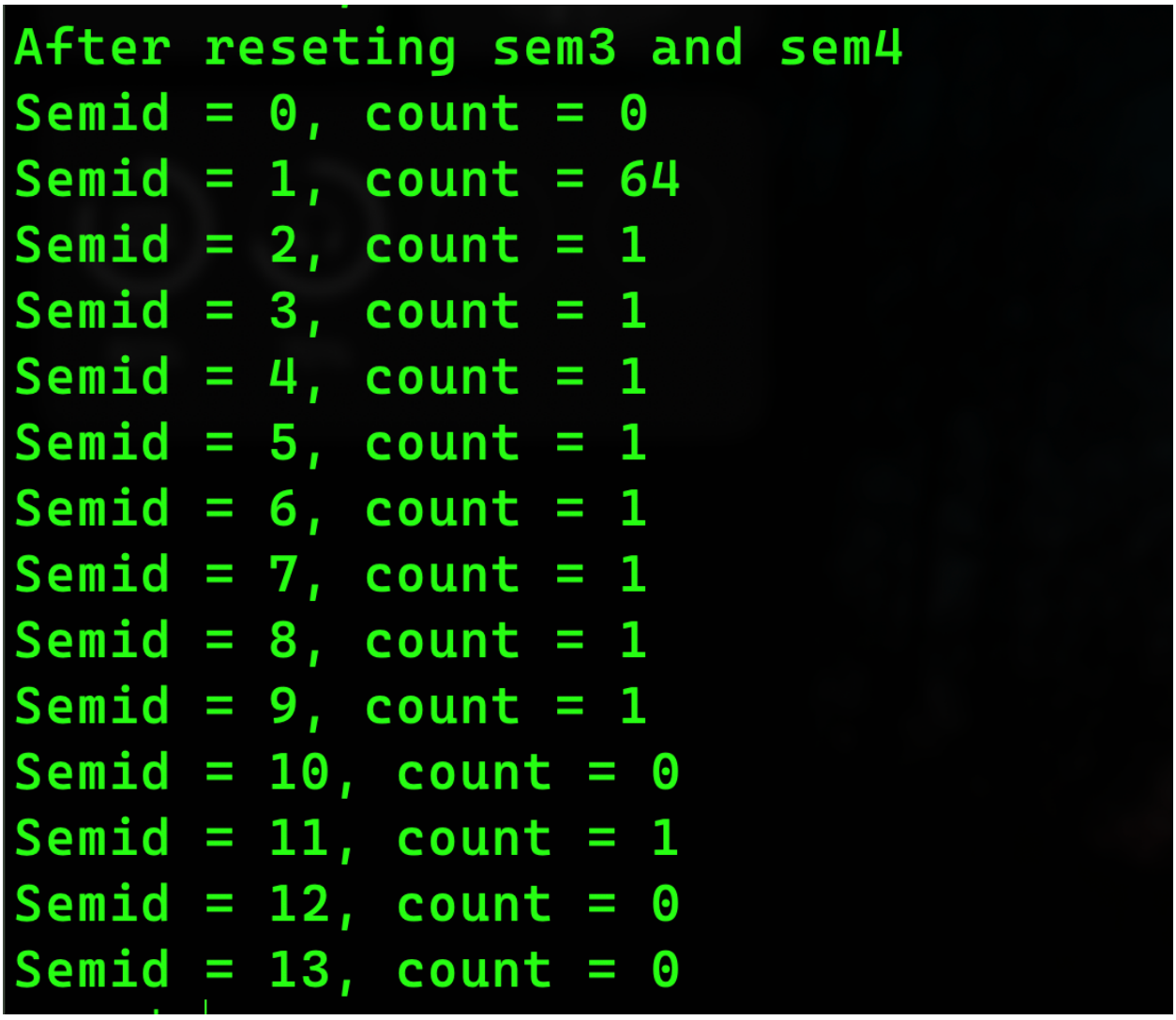
```
After deleting sem3 and sem4
Semid = 0, count = 0
Semid = 1, count = 64
Semid = 2, count = 1
Semid = 3, count = 1
Semid = 4, count = 1
Semid = 5, count = 1
Semid = 6, count = 1
Semid = 7, count = 1
Semid = 8, count = 1
Semid = 9, count = 1
Semid = 10, count = 0
Semid = 11, count = 1
```

Problem 12: semreset

```c
#include <xinu.h>

void proc1()
{
    printf("Process 1 : %d\n", getpid());
    while (1)
    {
    }
}

void proc2()
{
    printf("Process 2 : %d\n", getpid());
    while (1)
    {
    }
}

void Printer()
{
    struct sentry curr;

    for (int i = 0; i < NSEM; i++)
    {
        curr = semtab[i];
        if (curr.sstate == S_USED)
        {
            printf("Semid = %d, count = %d\n", i, curr.scount);
        }
    }
}

int hello1()
{

    sid32 sem1 = semcreate(0);
    sid32 sem2 = semcreate(1);
    sid32 sem3 = semcreate(2);
    sid32 sem4 = semcreate(3);

    printf("Newly created semaphore ids - %d, %d, %d, %d\n", sem1, sem2, sem3, sem4);

    resume(create(proc1, 1024, 20, "proc1", 0));
    resume(create(proc2, 1024, 20, "proc2", 0));
```

```
    resume(create(Printer, 1024, 20, "Printer", 0));

    semreset(sem3, 0);
    semreset(sem4, 0);

    printf("After reseting sem3 and sem4\n");
    resume(create(Printer, 1024, 20, "Printer", 0));

    return 0;
}
```

Terminal Output:



System Programming

Problem 1: Parent Creating N child processes

```c
#include <stdio.h>        // Include standard input/output functions (printf, etc.)
#include <stdlib.h>       // Include standard library functions (exit, EXIT_SUCCESS, etc.)
#include <unistd.h>       // Include UNIX standard functions (fork, etc.)
#include <sys/types.h>    // Include system type definitions (pid_t, etc.)
#include <sys/wait.h>     // Include wait functions for process synchronization

/* Global variable for number of children */
const int numChildren = 4; // Set the desired number of child processes to create

int main() {
    int j;                // Loop counter variable for tracking child number
    pid_t childPid;       // Variable to store the process ID returned by fork()

    setbuf(stdout, NULL); // Make stdout unbuffered to ensure immediate output display

    for (j = 0; j < numChildren; j++) {  // Loop to create numChildren child processes
        switch (childPid = fork()) {     // Fork a new process and store the return value in childPid
```

```c
        case -1:                        // If fork() returns -
1, an error occurred
            printf("Error: fork failed\n");
            return EXIT_FAILURE;        // Exit the program
with failure status
        case 0:                         // If fork() returns
0, we are in the child process
            printf("%d child,pid=%d\n", j,getpid());     //
Print which child number this is
            exit(EXIT_SUCCESS);         // Child process exits
successfully
        default:                        // If fork() returns a
positive value, we are in the parent process
            printf("%d parent,pid=%d\n", j,getpid());    //
Print that parent created child number j
            //wait will hold the value returned by child
process while terminating
            wait(NULL);                 // Parent waits for
the child to terminate (preventing zombie processes)
            break;                      // Continue to the
next iteration to create another child
        }
    }

    return EXIT_SUCCESS;                // Parent process
exits successfully after creating all children
}
```

Terminal Output:

```
0 parent,pid=17900
0 child,pid=17901
1 parent,pid=17900
1 child,pid=17902
2 parent,pid=17900
2 child,pid=17903
3 parent,pid=17900
3 child,pid=17904
```

Problem 2: Parent creating child and that child creating another child (2*n)

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

const int numChildren = 4;

int main()
{
    int j;
    pid_t childPid;

    setbuf(stdout, NULL);

    for (j = 0; j < numChildren; j++)
    {
        switch (childPid = fork())
```

```
        {
        case -1:
            printf("Error: fork failed\n");
            return EXIT_FAILURE;
        case 0:
            printf("%d child,pid=%d\n", j, getpid());
            pid_t grandChildPid = fork();
            if (grandChildPid == 0)
            {
                printf("%d grandchild,pid=%d\n", j, getpid());
                exit(EXIT_SUCCESS);
            }
            else
            {
                exit(EXIT_SUCCESS);
            }

        default:
            // printf("%d parent,pid=%d\n", j, getpid());
            wait(NULL);
            break;
        }
    }

    return EXIT_SUCCESS;
}
```

Terminal Output:

```
0 child,pid=19250
0 grandchild,pid=19251
1 child,pid=19252
1 grandchild,pid=19253
2 child,pid=19254
2 grandchild,pid=19255
3 child,pid=19256
3 grandchild,pid=19257
```

Problem 3: Parent Exiting Last

Wait(NULL) comment kore dile parent er kaj sesh holei exit korbe

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

const int numChildren = 4;

int main() {
    int j;
    pid_t childPid;

    setbuf(stdout, NULL);
```

```
    for (j = 0; j < numChildren; j++) {
        switch (childPid = fork()) {
        case -1:
            printf("Error: fork failed\n");
            return EXIT_FAILURE;
        case 0:
            printf("%d child,pid=%d\n", j, getpid());
            exit(EXIT_SUCCESS);
        default:
            printf("%d parent,pid=%d\n", j, getpid());
            wait(NULL);
            break;
        }
    }
    printf("Parent exiting last, pid = %d\n", getpid());

    return EXIT_SUCCESS;
}
```

Terminal Output:

```
0 parent,pid=20481
0 child,pid=20482
1 parent,pid=20481
1 child,pid=20483
2 parent,pid=20481
2 child,pid=20484
3 parent,pid=20481
3 child,pid=20485
Parent exiting last, pid = 20481
```