Operating Systems 2 - CS3523:

Lab Assignment 2: Hands-on with xv6 OS

Task-1.1: Write user-level sleep program for xv6

CODE -
sleep-CS22BTECH11061.c

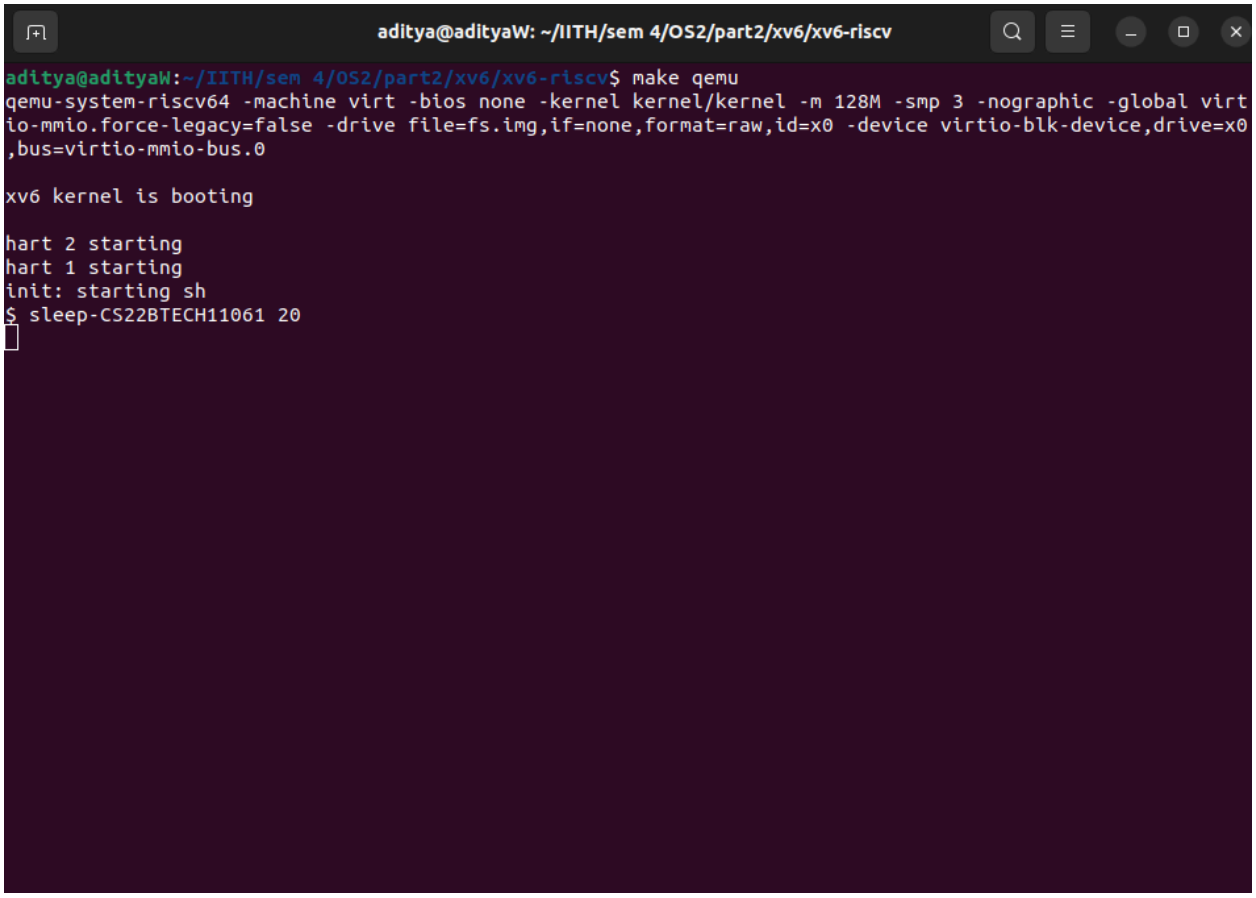
```c
#include "kernel/types.h"
#include "kernel/stat.h"
#include "user/user.h"

int
main(int argc, char *argv[])
{
   //checking correct number of arguments
   if(argc != 2){
       fprintf(2, "Usage: sleep <ticks>\n");
       exit(1);
   }

   //getting the number of ticks
   int ticks = atoi(argv[1]);
   if(ticks < 0){
       fprintf(2, "sleep: invalid number of ticks\n");
       exit(1);
   }

   //sleeping for the given number of ticks
   sleep(ticks*10);

   exit(0);
}
```

Methodology -

1. We Use System Calls to Implement sleep command. For this we have system libraries available.
2. Usage: sleep <ticks>. It takes one argument for number of ticks

3. After extracting number of ticks, We call sleep system call.

Output -



Task-1.2: Write pingpong for IPC between two processes for xv6

CODE -

pingpong-CS22BTECH11061.c

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


int main() {
   int p[2]; // Pipe file descriptors
   //p[0] to read and p[1] to write
```

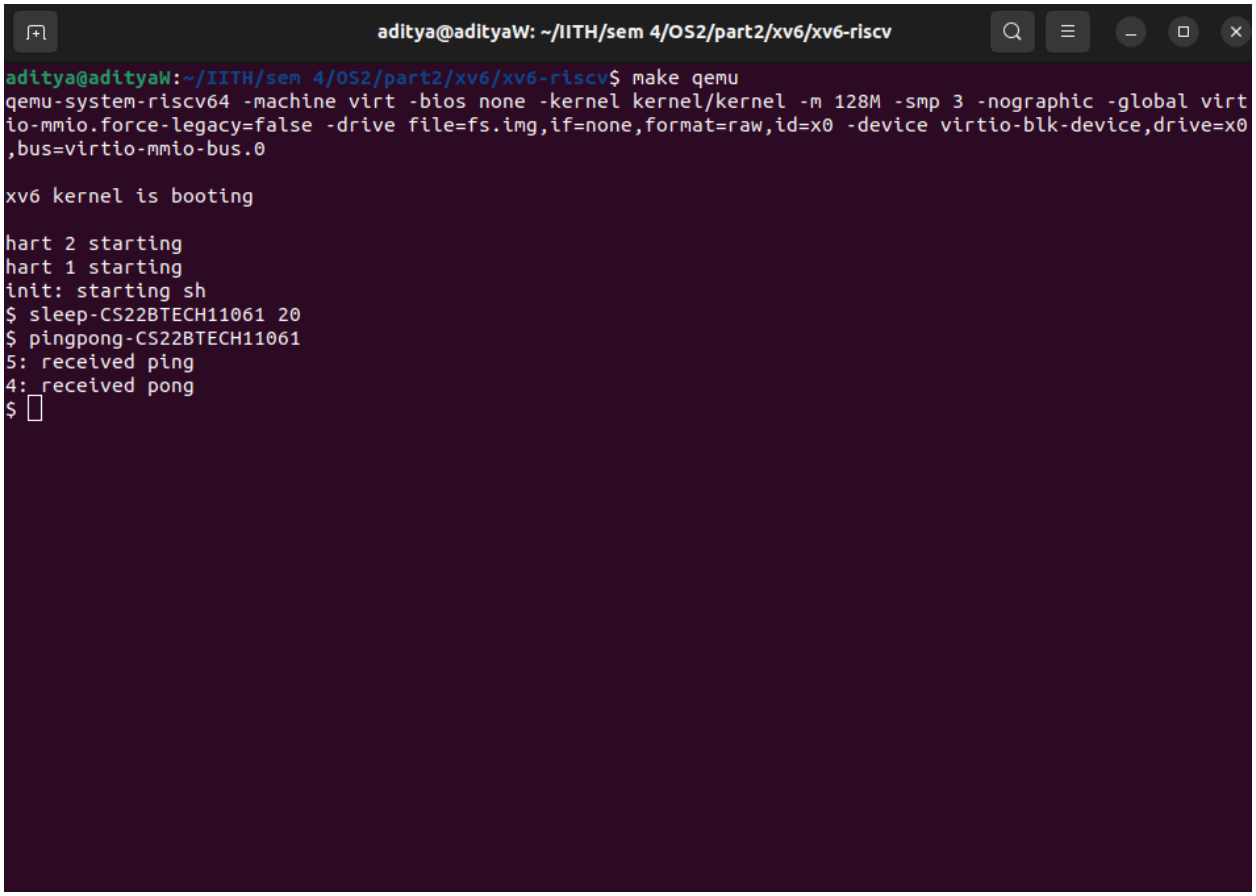
```
// Create pipes
if (pipe(p) < 0) {
    fprintf(2, "pipe failed\n");
    exit(1);
}

int pid = fork();

if (pid < 0) {
    fprintf(2, "fork failed\n");
    exit(1);
}

if (pid == 0) {
    // Child process
    char msg[1];

    // Read byte from parent
    if (read(p[0], msg, 1) != 1) {
        fprintf(2, "read error\n");
        exit(1);
    }
    close(p[0]); // Close read end
    printf("%d: received ping\n", getpid());

    // Write same byte to parent
    if (write(p[1], msg, 1) != 1) {
        fprintf(2, "child write error\n");
        exit(1);
    }
    close(p[1]); // Close write

    exit(0);
} else {
    // Parent process
    char msg[1] = "a"; // Byte to send

    // Write byte to child
    if (write(p[1], msg, 1) != 1) {
```

```
            fprintf(2, "parent write error\n");
            exit(1);
        }
        wait(0); // Wait for child process to exit

        // Read byte from child
        if (read(p[0], msg, 1) != 1) {
            fprintf(2, "parent read error\n");
            exit(1);
        }
        printf("%d: received pong\n", getpid());

        close(p[0]); // Close read end
        //write close is delayed so that we can still read
        close(p[1]); // Close write end
        exit(0);
    }

    exit(0);
}
```

Methodology -

1. We Use System Calls to Implement pipes and fork. For this we have system libraries available.
2. We first create a pipe and fork the current process.
3. The single byte message we use here is the char ‘a’.
4. First parent sends the byte to child and after reading it child prints ‘received ping’.
5. After child is done, it sends the byte message back to the parent and exits. Parent waits for the child to exit before reading. Once byte is received, parent prints ‘received pong’.

Output -



Executing -

1. Add sleep-CS22BTECH11061.c and pingpong-CS22BTECH11061.c to user folder.
2. Add the program to UPROGS in Makefile. In this way -

   $U/_sleep-CS22BTECH11061\

   $U/_pingpong-CS22BTECH11061\

3. Run the Commands.