Lab Assignment 2: Hands-on with xv6 OS

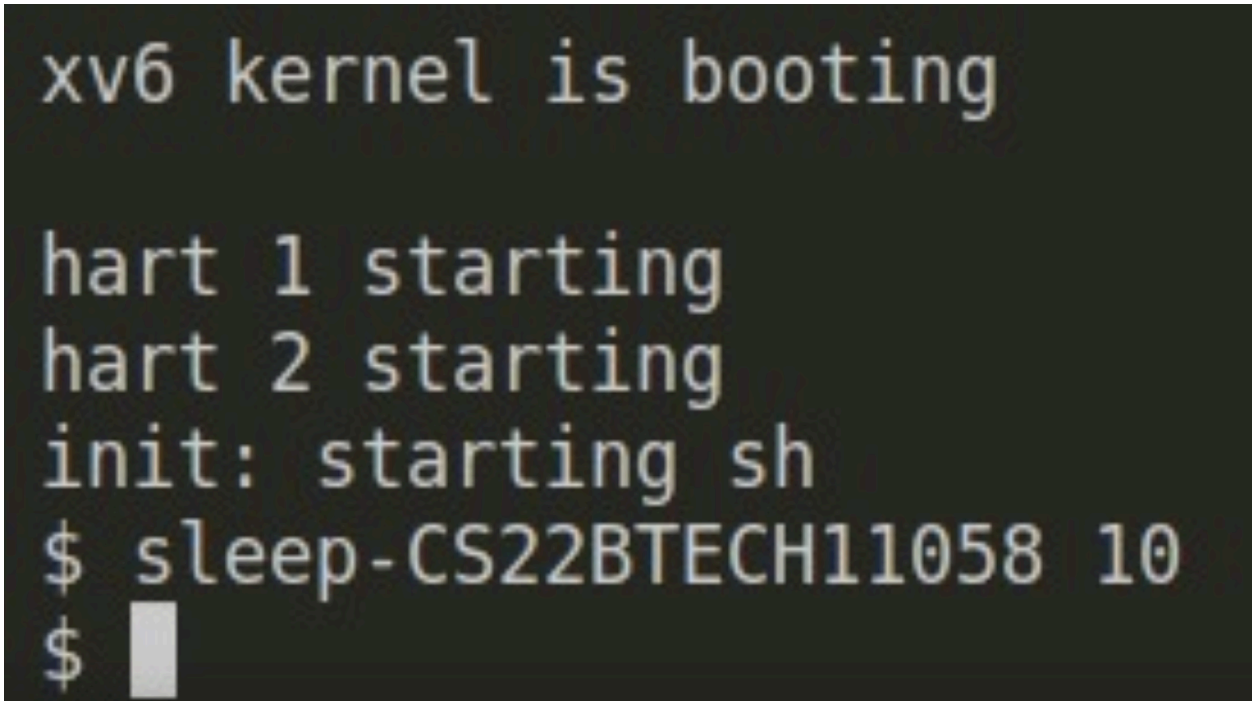
Name:- Syed Abrar
Roll Number:- CS22BTECH11058

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

Code:-

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

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

   int ticks = atoi(argv[1]);
    ticks = ticks * 10;

   if(ticks <= 0){
       fprintf(2, "Invalid number of ticks: %d\n", ticks);
       exit(1);
   }

   sleep(ticks);

   exit(0);
}
```

Output:-



Solution Methodology:-

The program checks if the correct number of arguments are provided. If not, it prints a usage message and exits with an error code.

Ticks Calculation: It multiplies the provided number of ticks (specified in the command line argument) by 10.

Error Handling: If the calculated number of ticks is less than or equal to 0 after multiplication, it prints an error message and exits with an error code.

It calls the sleep() function with the calculated number of ticks.

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

Code:-

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

int main(int argc, char *argv[]) {
   int pipe_fds[2];
   char buffer[10];

   pipe(pipe_fds);

   int child_pid = fork();

   if (child_pid < 0) {
       printf("Error: Failed to create child process\n");
       exit(1);
   }
   else if (child_pid == 0) {
       read(pipe_fds[0], buffer, sizeof(buffer));
       printf("[%d - Child] Received message: %s\n", getpid(), buffer);
       write(pipe_fds[1], "pong", 5);
       close(pipe_fds[0]);
       close(pipe_fds[1]);
       exit(0);
```

```
    }
    else {
        write(pipe_fds[1], "ping", 5);
        wait(0);
        read(pipe_fds[0], buffer, sizeof(buffer));
        printf("[%d - Parent] Received message: %s\n", getpid(), buffer);
        close(pipe_fds[0]);
        close(pipe_fds[1]);
    }

    exit(0);
}
```

Output:-

```
init: starting sh
$ pingpong
[4 - Child] Received message: ping
[3 - Parent] Received message: pong
$ pingpong
[6 - Child] Received message: ping
[5 - Parent] Received message: pong
$ pingpong
[8 - Child] Received message: ping
[7 - Parent] Received message: pong
$ pingpong
[10 - Child] Received message: ping
[9 - Parent] Received message: pong
```

Solution Methodology:-

The program creates a pipe using the pipe() function, which returns two file descriptors: one for reading (pipe_fds[0]) and one for writing (pipe_fds[1]).

It forks a child process to for communication between the parent and child.

In the child process, it reads from the read end of the pipe (pipe_fds[0]), prints the received message, writes a response message ("pong") to the write end of the pipe (pipe_fds[1]), and then closes both ends of the pipe before exiting.

In the parent process, it writes a message ("ping") to the write end of the pipe, waits for the child process to finish, reads the response message from the read end of the pipe, prints the received message, and then closes both ends of the pipe.

Output shows the exchange of messages ping and pong for inter process communication