Implement Socket API (local loopback only) in xv6

SUBMISSION STEPS:

1. Start in a fresh git repository of xv6 (git clone https://github.com/mit-pdos/xv6-public.git)
2. In your repository folder, make necessary code changes to finish your assignment
3. Commit your changes. (Learn to use **git commit** command. Use **git status** command to check what files you have modified/added, use **git add** command to add the files to a commit.)
4. After successful commit run "git show HEAD > patch_xv6_Socket_<YOUR_ROLL_NO>
5. In step 4 you have created a file (patch_xv6_Socket_<YOUR_ROLL_NO>). Submit it to Moodle.

STEPS TO UNPACK A PATCH TO YOUR REPOSITORY:

1. git apply <patch file name>
2. make and make qemu to check
3. You will apply the above steps on the evaluation day. Create a clone of xv6 in your lab machine and apply the patch,downloaded from Moodle. Make and test before demonstrating to your teacher.
4. You also need #1 and #2 to apply the patch I have given. **THIS SHOULD REDUCE YOUR WORK LOAD QUITE A BIT**

The system calls that should be implemented are as follows:

// Listen to a local port (parameter). Return 0 for success, negative for failure

int listen(int);

// Connect to remote port and host (parameter). Only "localhost" and "127.0.0.1" as host should be supported. Returns the local port number, if connection was successful. Negative value returned indicate failure.

int connect(int, const char* host);

// Send a buffer to remote host. The local port, buffer, size of data are parameters. Return 0 for success. Negative value for failure. Send blocks, if remote host has not yet read earlier data

int send(int, const char*, int);

// Receive data from remote host. The local port, buffer, size of buffer are parameters. Return 0 for success. Negative value for failure. recv blocks, if no data is available.
int recv(int, char*, int);

// Disconnect (and close) the socket. The local port is parameter.
int disconnect(int);

// Implement the following error codes. These should be available for user level programs, as well as in kernal space, as the defined constants.

E_NOTFOUND -1025
E_ACCESS_DENIED -1026
E_WRONG_STATE -1027
E_FAIL -1028
E_INVALID_ARG -1029

Error checking:

Parameter issues should return E_INVALID_ARG
Accessing a socket that is not in the stable should return E_NOTFOUND
Accessing a socket from wrong process should return E_ACCESS_DENIED
Attempts to send or receive, when the socket is not connected, should return E_WRONG_STATE
If no more socket can be opened (limit exceeded), return E_FAIL

(Optional Task) Close the outstanding sockets owned by a process, when process terminates.
(Optional Task) Modify socktest to demonstrate the above feature

Limitations (Future work):

- Send will block the caller process, until the recepients buffer is empty.
- Timeout cannot be specified in the recv() call

NOTE: Keep an eye out in Moodle for some code that I will provide to reduce your work load.

As promised Find the socktest.c file as a separate upload. (No guarantees are provided!)