Network Programming Assignment 2, Design Document: Q1

Student Name	**Student ID**
Anirudh Buvanesh	2016B4A70614P
Ashish Kumar	2016B4A70636P

The problem requires us to design a preforking server, where the server create a process pool to handle incoming connections

1. Data structures

a. **child_state_info[][2]:** For each child created by the parent 2 values are stored in this array. The first entry is either *IDLE* or *SERVING* depending on whether the child is handling connections or not. The second entry is the number of connections being handled.
b. **child_pid_map[]:** Stores pid of the child which is later helpful for identifying the socket through which the parent needs to send the reply.
c. **child_unix_fd[][2]**: Stores the unix sockets (*created via socketpair*) for parent child communication.
d. **Struct ipc_message:** The structure used during IPC. Stores pid of the sender, message type (could be *CONNECTION_ESTABLISHED, CONNECTION_CLOSED, RECYCLE_CONNECTION*).

2. Implementation details

In order to ensure that the number of idle processes are in the specified range there are 2 functions *max_idle_handler* and *min_idle_handler*. The parent starts off by making a call f *min_idle_handler* which takes care of spawning processes and then *max_idle_handler* which kills off the excess processes by sending a SIGTERM signal to them.

Every incoming request and completion of request (*identified by EOF on the child side*) is notified to the parent with message with the type set as either *CONNECTION_ESTABLISHED* or *CONNECTION_CLOSED*. On receiving the message the parent updates the above structures, which would be later used for printing process pool updates/

When the request limit of a child process is reached it stops listening to more connections by clearing *listen_fd* from the interest list in select call and once it has handled all it's connections it sends a message to the parent with type *RECYCLE_CONNECTION*. When the parent receives this message it sends a SIGTERM signal to the process.

Testing was done using **httperf** tool

Files submitted

prefork_server.c - source code (*implementation logic over here*)
prefork_sevrer.h - header file that contains header files for the corresponding source code
makefile - generates the executable prefork_server.o
Instructions.txt - provides the instructions for executing program