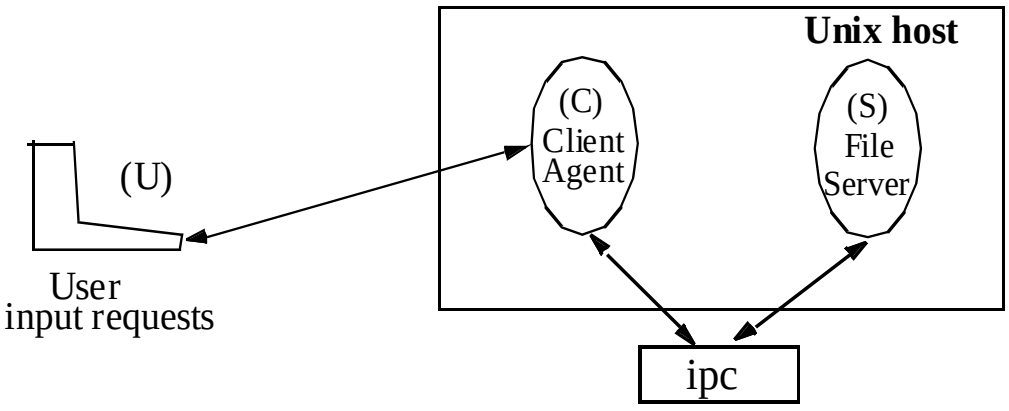
Santa Clara University

Department of Computer Engineering
(COEN 236)

Project-2

Project Overview:

In this project, we would like to re-implement project1 (File Server) using the different Unix IPC mechanism covered in the class. The client agent process reads input commands from the user (U), pass the operation required to the file server (S) through our IPC mechanism, gets the result back and pass it back to the user (U). The interactions between the user, client, and the file server are synchronous, i.e., the user has to wait to receive the result of an operation before requesting another operation, etc.



Functional Requirements:

This project requirement is exactly similar to **project-1** with the exception of the IPC mechanism used. In **project-1** we used a shared disk file as the communication facility between (**C**) and (**S**) and signals as the synchronization to regulate accessing the shared disk file. In this project, we have multiple implementations using different IPC mechanisms as discussed below:

1. Use a message abstraction on top of **pipes** between (C) and (S). The message format is:

```
#define         MAXBUF          100         /* any reasonable constant *.

typedef     struct {
                int         mesg_len;           /* msg length in bytes */
                int         mesg_type;          /* msg type */
                char        mesg_data[MAXBUF];  /* buffer address */
} Mesg;
```

P.S. You may redefine MAXBUF to ensure it is always larger than the file you will be reading, i.e., no need to deal with multiple packets when reading a file.

2. Use the same message abstraction on top of a **FIFO** between (**C**) and (**S**).
3. Use a **System V Message Queue** between (**C**) and (**S**).
4. Use a **Posix Message Queue** between (**C**) and (**S**).

Programming Hints:

1. Structure your code to minimize reimplementation, i.e., maximize code reuse, of every module for the different IPC mechanisms.
2. Your agent process (C) should be structured into four segments:
    - *Initialization segment*: equivalent to project-1 with the relevant IPC mechanism used.
    - *Command input segment*: Command input segment: read a line command <Opcode filename> where the supported Opcodes include (EXIT, READ, DELETE) from the standard input, check legal syntax, and branch to execute legitimate commands.
    - *Command execution segment*: if Non Exit command, forward the file operation to the file server and waits for the result. If Exit do cleanup, including the server, and terminates.

- *Output segment*: check for success operation and write the result on the standard output, and asks for the next command.

3. Include one/two pages description of your project implementation. Remember to have good comments in your code. Also, include a hard copy of user interaction with your file server application. Finally, write/type clearly on the Front page the absolute UNIX pathname for your application to the grader to try (making sure RWX permissions for others).

4. Bring to class two complete copy sets of your project (one for the grader and the second for myself).