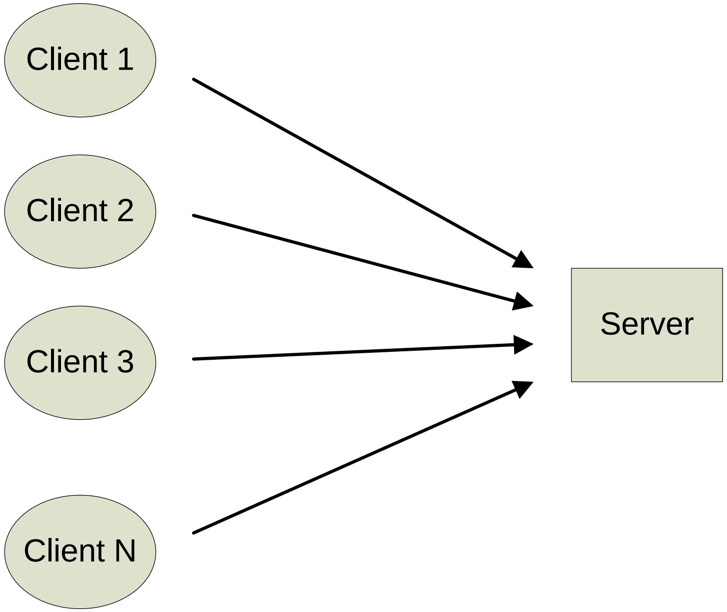
Sockets Programming

http://ict.siit.tu.ac.th/~steven/its332/

Server Handling Multiple Connections

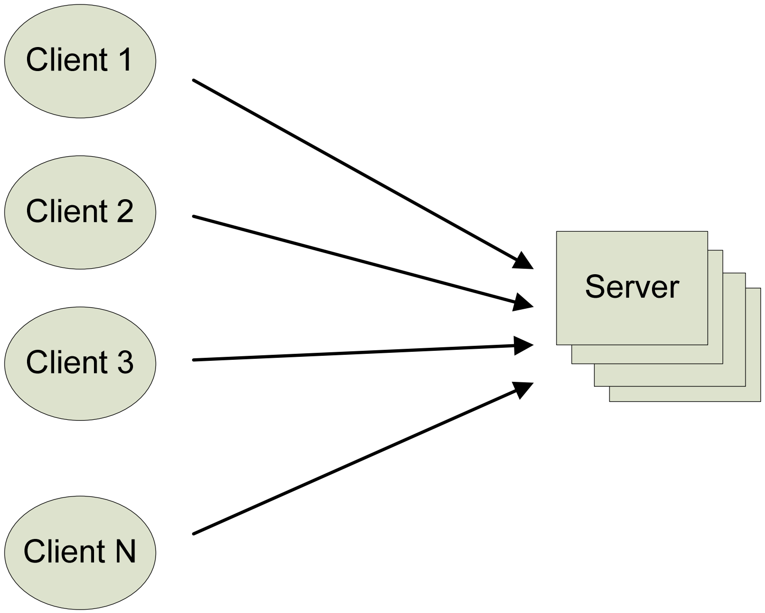
- A server often receives connection requests from multiple clients (and even multiple requests from the one client)



- If we have just one server running, then would have to wait for server to finish processing data transfer from Client 1 before can process data transfer for Client 2
  - Not practical, because most servers want to process data from clients in “parallel”

Multiple Copies of the Server Program

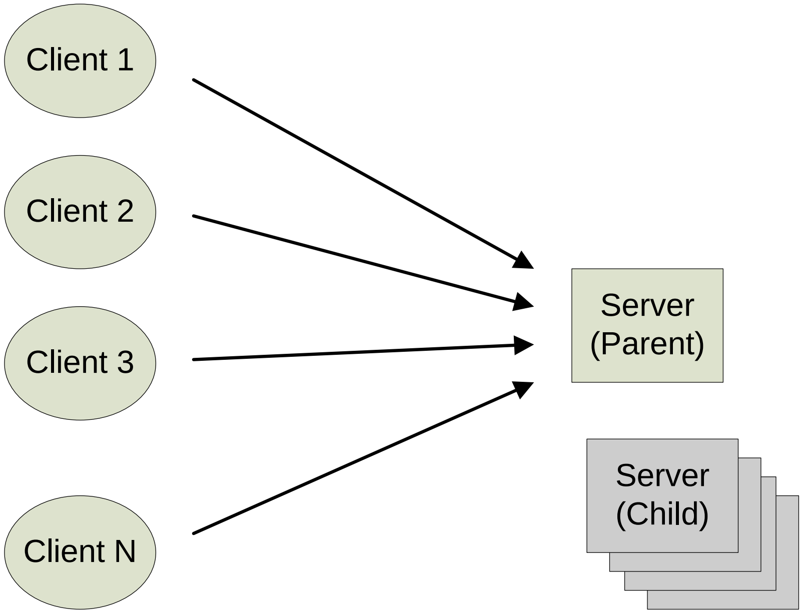
- The user starts multiple copies of the server program



- How many?
  - Not enough: clients will try to connection, but connections will be refused
  - Too many: Very inefficient (use memory, CPU) if no requests from clients

Multiple Dynamic Copies of Server Process

- The user starts a single Server program (called the Parent process)
- The Parent automatically starts new copies of the Server process whenever a Client request is received (called a Child process)
  - When the Client finishes the connection, the Child process ends



Example of Parent/Child Processes

```
// User starts Server process
S = socket (…);
…
while (1) {
   newS=accept(S,&cli,&clilen);
   pid = fork();
   if (pid < 0)
        error("ERROR on fork");
   if (pid == 0)  {
        close(S);
        dostuff(newS);
        exit(0);}
   else
        close(newS);
}
```

fork() creates an exact copy of this process, including current values of variables

fork() returns 0 for the newly created Child process

fork() returns the current process ID (not 0) for the Parent process

Example of Parent/Child Processes

```
// User starts Server process
S = socket (…);
…
while (1) {
   newS=accept(S,&cli,&clilen);
   pid = fork();
   if (pid < 0)
        error("ERROR on fork");
   if (pid == 0)  {
        close(S);
        dostuff(newS);
        exit(0);}
   else
        close(newS);
}
```

When accept() is called, it blocks until a TCP connection setup is complete

If TCP connection is successful, accept() creates a new socket, and returns its identifier (newS)

Child Process
- Close the old socket (S)
- Process the request using new socket (newS)
- Exit (stop the Child process)

Example of Parent/Child Processes

```
// User starts Server process
S = socket (…);
…
while (1) {
   newS=accept(S,&cli,&clilen);
   pid = fork();
   if (pid < 0)
        error("ERROR on fork");
   if (pid == 0)  {
        close(S);
        dostuff(newS);
        exit(0);}
   else
        close(newS);
}
```

When accept() is called, it blocks until a TCP connection setup is complete

If TCP connection is successful, accept() creates a new socket, and returns its identifier (newS)

Parent Process
- Close the new socket (newS)
- Repeat the while(1) loop (e.g. wait for new TCP connection)

Implementation Details

- Our example uses fork() to create Child processes
  - Parent server handles connection setup
  - Child servers handles data transfer
    - Children are created when a new connection request is accepted
    - Children are destroyed when data transfer is complete

- fork() uses a separate process for children

- There are other implementation techniques (threads) that can be more efficient (but often complex) in some cases