Computer Networks Lab

Lab 6:

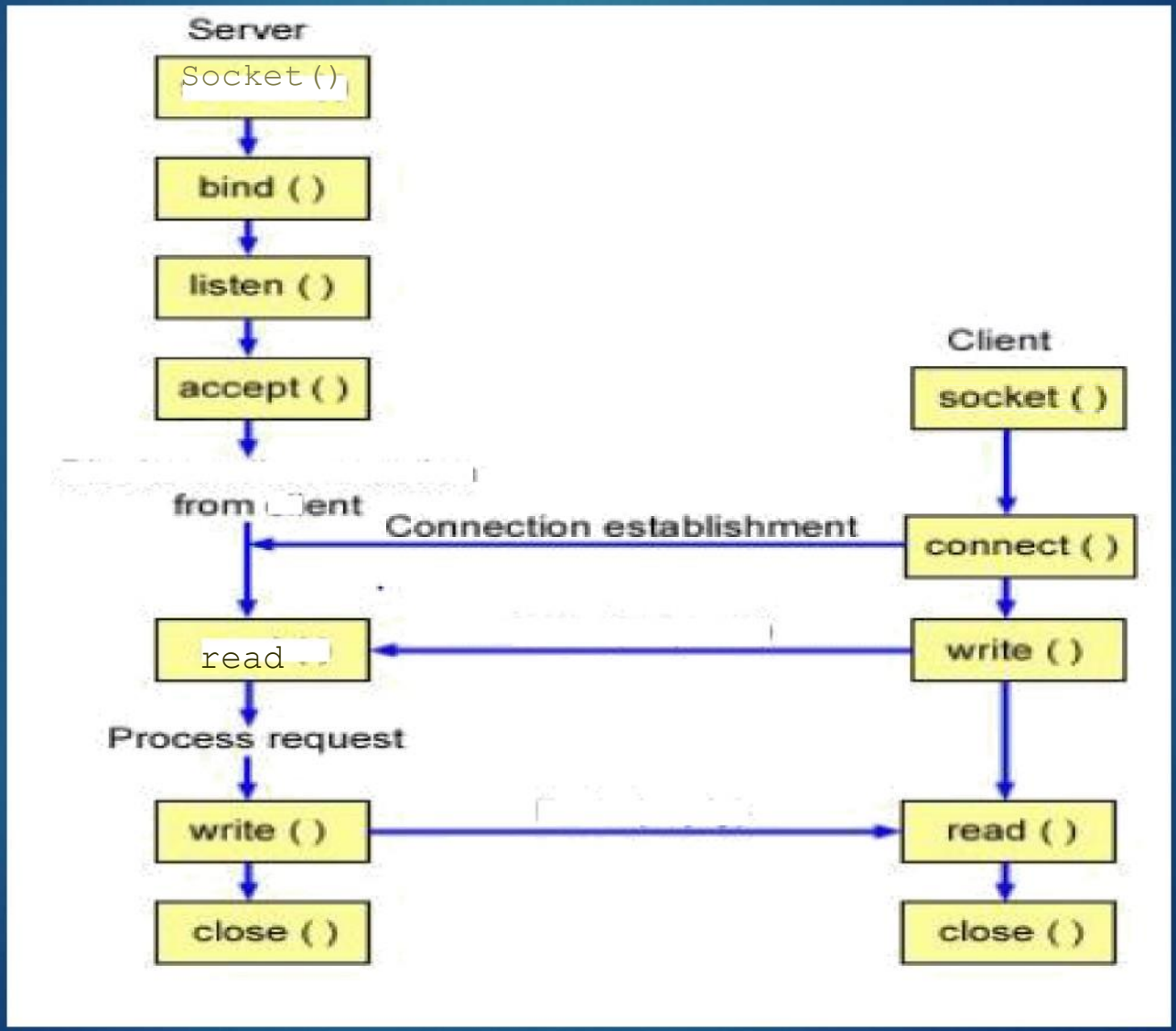
Socket Programming:

(TCP) Concurrent Servers

Outline

1. Iterative servers
2. Fork()
3. Concurrent servers
4. Handling Child Servers

TCP Client Server Interaction



Iterative Server

- Server is iteratively waiting for clients. Iterative servers are fairly simple and are suitable for transactions that do not last long.
- When a client calls connect function the server gets connected with it and provides it services.
- During this period no more client can connect with the server because it is tied up with a single client.

Iterative servers

create a socket- Socket()
bind to the desired port- Bind()

put the socket in passive mode - listen()
while (1)
{
 Accept the next connection-Accept()

 while (client requests)
 {
 read the request from the client - read()/recv()
 process request
 send a reply to client- write()/send()
 }

 close the client socket- close()
}

Fork() System Call

When the server receives and *accepts* the client's connection, it *forks* a copy of itself called child and lets the child handle the client

The prototype of "fork" call is:

pid_t fork(void);

where the definition of "pid_t" is given in **<sys/types>** include file, and the include file **<unistd.h>** contain the declaration of "fork" system call.

When successful, "fork" returns the process ID of the child process in the parent process, and it returns a "0" in the child process.

Fork() System Call

By checking the return value from "fork", a process can easily determine if it is a parent or child process.

A process can generate multiple child processes, but each child process has only one parent.

What happens when a fork system call is made?

✓ When a fork system call is made, the operating system generates a copy of the parent process which becomes the child process.

✓ The OS will pass to the child process most of the parent's system information (e.g. open file descriptors, environment information etc.)

Fork() System Call

Some information, however, is unique to the child process.

Process ID

Different parentprocess

ID Process times

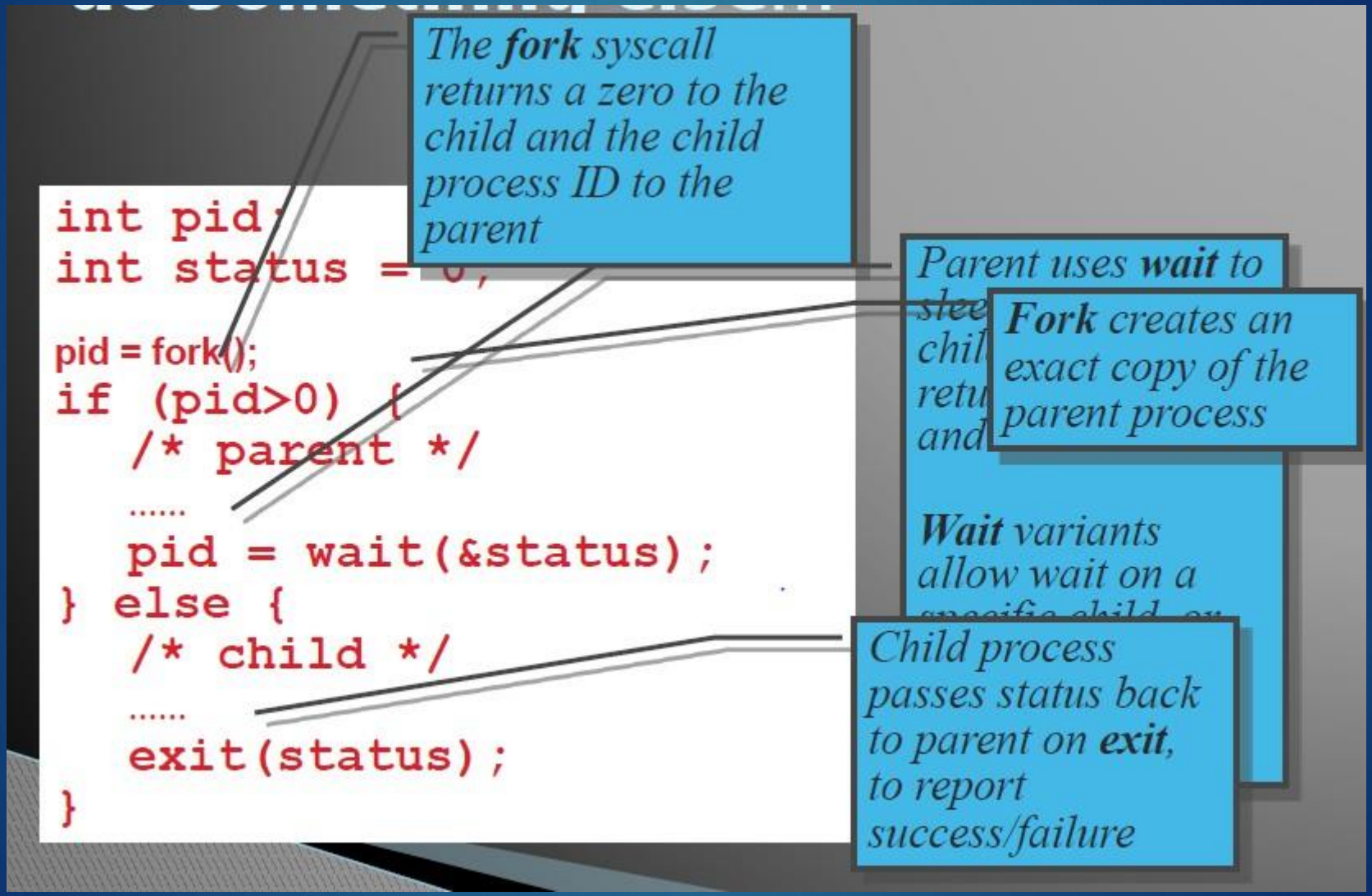
Own copy of file descriptors

Resource utilization (initialized to zero)

Fork()

```
pid_t new_pid;
new_pid = fork();
switch(new_pid) {
case -1 : /* Error */
break;
case 0 : /* I am child */
break;
default : /* I am parent */
break;
}
```

But we want the child process to do something else…



Concurrent servers

Concurrent servers can handle multiple clients at same time.

Concurrent servers fork a child to handle each client.

After the establishment of connection sever calls fork and creates child to serve the client.

Then parent closes the connected socket.

Concurrent servers

```
bind to the desired port
put the socket in listen mode (passive)
while (1)
{
 Accept the next client connection
 fork() // create a new process to handle the request if
 (child) // i.e if return value of the previous fork is zero {
        this part of the code will only be executed by the child
        process communicate with the client socket process client
        request
        close client socket
        exit // you can safely terminate the child process-> parent is still
listening for new clients
 }
 else // i,e. for the parent process
 {
        close client socket as parent will not handle it
 }
}
```

Thank You