# Comp 204: Computer Systems and Their Implementation

## Lecture 5: Concurrent Programming and Threads

# Today

- Inter-process communication
  - Sockets
- Introduction to Concurrent Programming
  - Threads

# Connection Procedure

CLIENT

set up socket addr.
create socket

request connection

send/receive data
disconnect socket

SERVER

set up socket addr.
create socket
bind socket addr.
listen for connection requests

accept connection
send/receive data
disconnect socket

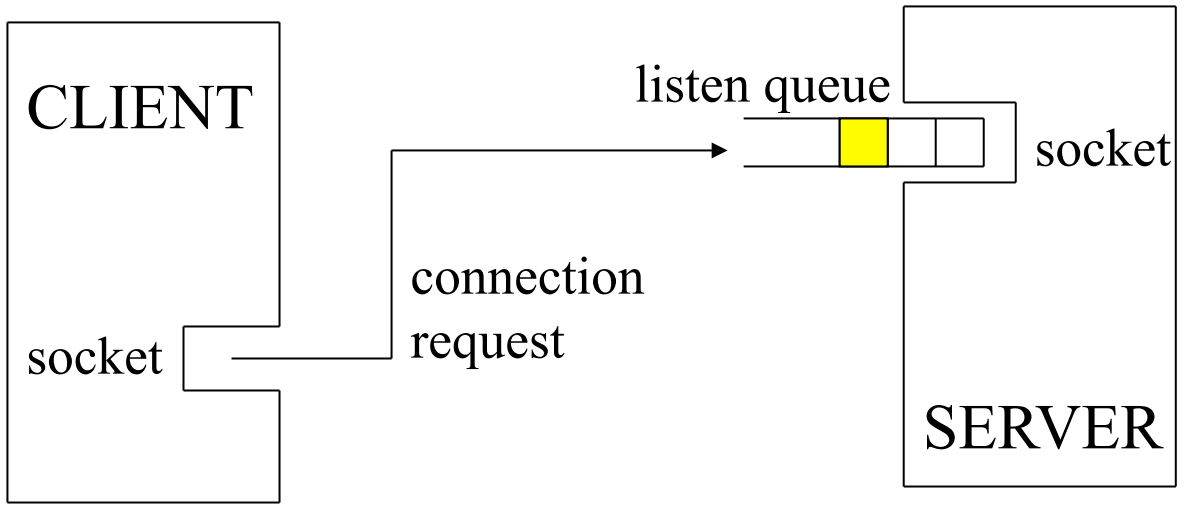
# Procedure

- Set up socket address
  - socket address consists of three parts:
  - (a) the address family
    AF_INET  The Internet address family
    AF_UNIX  The Berkeley UNIX Domain address family.
    - More efficient than AF_INET, but can only be used locally (i.e. on one node)
  - (b) the internet address (identifies the host)
  - (c) the port address (identifies service offered)
    - e.g. http on port 80; ftp on port 21

# Procedure

- Create socket
  - both server and client must create a socket
- Bind socket address
  - to enable clients to connect to its socket, the server must bind an address to its socket
- Listen for connection requests
  - server now listens for incoming connection requests from clients
  - sets up a listen queue, specifying max. no. of requests in queue at any time
- Request connection
  - client can now request a connection to server

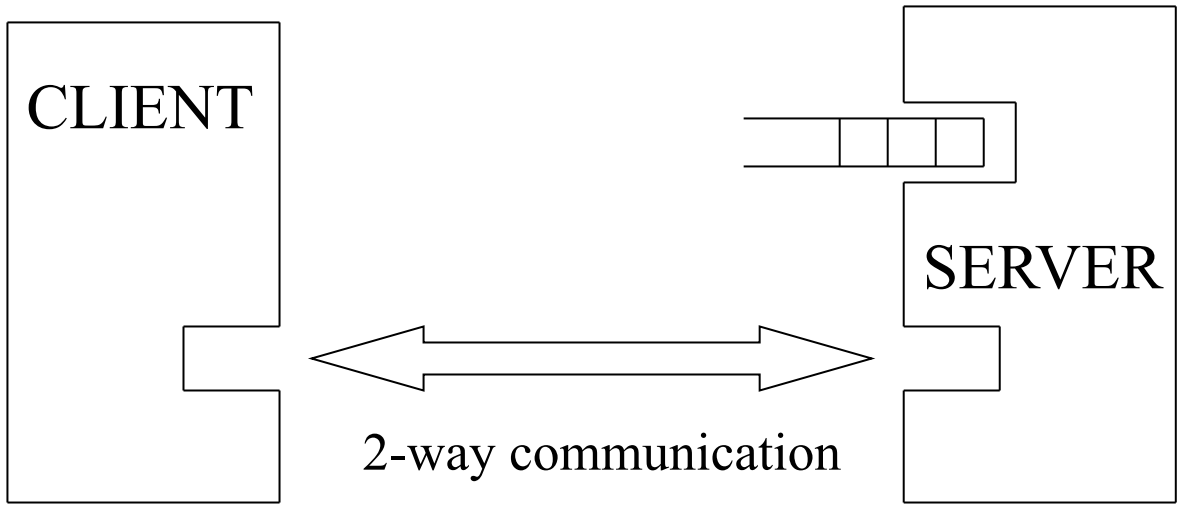
# Procedure


CLIENT
socket
connection
request
listen queue
socket
SERVER

# Procedure

- Accept connection
  - after it has set up its listen queue, the server can accept connection requests
  - when it accepts a request, it creates a new socket which
    - has same properties as original listen socket
    - has same bound address as the original
    - is connected to the client's socket
- Send/Receive data
  - a socket descriptor is treated like a file descriptor
    - read() and write() used to transfer data
    - redirection, etc. can also be performed

# Procedure




- Disconnect socket
  - sockets can be closed with the file close() call

# Exercise

- How can a server cope with a number of clients connected simultaneously, each at different stages of execution?

# Concurrent Programming

- Consider a program that resolves an arithmetic expression
- The steps in the calculation are performed serially: instructions are executed one step at a time
  - every operation within the expression is evaluated in sequence following the order dictated by the programmer (and compiler)
- However, it may be possible to evaluate numerous sub-expressions at the same time in a multiprocessing system

# Concurrent Programming

- Consider formula to find one root of a quadratic:

$$x = (-b + \sqrt{(b^2 - 4ac)}) / 2a$$

- This can be split into several operations:

*Concurrent operations*

1. t1 = -b
2. t2 = b*b
3. t3 = 4*a
4. t4 = 2*a

*Serial operations*

5. t5 = t3*c
6. t5 = t2 - t5
7. t5 = √t5
8. t5 = t1 + t5
9. x = t5/t4

- In Java, parallel execution is achieved with threads

# Exercise

- Identify the parallelism in the following:

1)
```
for i = 1 to 10
        a[i] = a[i] + 1
```

2)
```
for i = 1 to 10
        a[i] = a[i] + a[i - 1]
```

# Question

- In calculating the formula **ut + ½at²** using maximal concurrency, which of the operations might be computed in parallel?

  a) u*t; a/2; t*t
  b) u*t; t+½; a*t
  c) u+a; t*t
  d) u+a; t*t; ½
  e) no parallelism is possible for this formula

**Answer: a**
*u*t; a/2; and t*t – i.e. only those parts of the formula that have no dependencies on other parts of the formula can be run concurrently. Think how the formula could be written in 3-code…*

# Threads

- A thread can be thought of as a lightweight process
- Threads are created *within* a normal (heavyweight) process
- Example 1: a Web browser
  - one thread for retrieving data from Internet
  - another thread displays text and images
- Example 2: a word processor
  - one thread for display
  - one for reading keystrokes
  - one for spell checking

# Thread Benefits

- Four major categories:

- Responsiveness: In a multithreaded interactive application a program may be able to continue executing, even if part of it is blocked
  - e.g. in a web browser: user waiting for image to download, but can still interact with another part of the page

- Resource sharing: Threads share memory and resources of the process they belong to, thus we have several threads all within the same address space

- Economy: Threads are more economical to create and context switch as they share the resources of the processes to which they belong

- Utilisation of multiprocessor architectures: In a multiprocessor architecture, where each thread may be running in parallel on a different processor, the benefits of multithreading can be increased

# Thread Types

- Support for threads may be provided either at the user level, for user threads, or at the kernel level, for kernel threads

- User level: Threads are supported above the kernel and implemented at the user level by a thread library

  – Library provides support for thread creation, scheduling and management, with no support from the kernel

  – User level threads are fast to create and manage

  – But, if kernel is single threaded, one thread performing a blocking system call will cause the entire process to block, even if other threads within the process can run

# Thread Types

- Kernel level: Threads are supported directly by the OS
  - Thread creation, scheduling and management done by the kernel in the kernel space
  - Slow to create and manage
  - But, since the threads are managed by the kernel, if one thread performs a blocking system call, the kernel can schedule another thread within the application to run
  - In a multiprocessor system, kernel can schedule threads on different processors