

# Multithreaded Servers

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

- 1 Server for Multiple Clients  
avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

- 2 Waiting for Data from 3 Clients  
running a simple multithreaded server  
code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

- 3 n Handler Threads for m Client Requests  
managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

- 4 Pleasingly Parallel Computations

MCS 275 Lecture 32  
Programming Tools and File Management  
Jan Verschelde, 2 April 2010

# Multithreaded Servers

## Server for Multiple Clients

avoid to block clients with waiting

using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server

code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically

code to manage handler threads

## Pleasingly Parallel Computations

- 1 **Server for Multiple Clients**  
avoid to block clients with waiting  
using sockets and threads
- 2 **Waiting for Data from 3 Clients**  
running a simple multithreaded server  
code for client and server
- 3 **n Handler Threads for m Client Requests**  
managing threads dynamically  
code to manage handler threads
- 4 **Pleasingly Parallel Computations**

2 Apr 2010

Server for  
Multiple  
Clientsavoid to block clients  
with waitingusing sockets and  
threadsWaiting for  
Data from 3  
Clientsrunning a simple  
multithreaded servercode for client and  
servern Handler  
Threads for m  
Client  
Requestsmanaging threads  
dynamicallycode to manage  
handler threadsPleasingly  
Parallel  
Computations

# Server for Multiple Clients

avoid to block clients with waiting

## Client/Server computing:

- 1 server provides hardware or software service,
- 2 clients make requests to server.

Typically, clients and servers run on different computers.

## Operations in server side scripting:

- 1 define addresses, port numbers, sockets,
- 2 wait to accept client requests,
- 3 service the requests of clients.

For multiple clients, server could either

- 1 wait till every one is connected before serving,
- 2 or accept connections and serve one at a time.

In either protocol, clients are blocked waiting.

# Server for Multiple Clients

avoid to block clients with waiting

## Client/Server computing:

- 1 server provides hardware or software service,
- 2 clients make requests to server.

Typically, clients and servers run on different computers.

## Operations in server side scripting:

- 1 define addresses, port numbers, sockets,
- 2 wait to accept client requests,
- 3 service the requests of clients.

For multiple clients, server could either

- 1 wait till every one is connected before serving,
- 2 or accept connections and serve one at a time.

In either protocol, clients are blocked waiting.

# Server for Multiple Clients

avoid to block clients with waiting

Client/Server computing:

- 1 server provides hardware or software service,
- 2 clients make requests to server.

Typically, clients and servers run on different computers.

Operations in server side scripting:

- 1 define addresses, port numbers, sockets,
- 2 wait to accept client requests,
- 3 service the requests of clients.

For multiple clients, server could either

- 1 wait till every one is connected before serving,
- 2 or accept connections and serve one at a time.

In either protocol, clients are blocked waiting.

# Server for Multiple Clients

avoid to block clients with waiting

## Server for Multiple Clients

avoid to block clients with waiting

using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server

code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically

code to manage handler threads

## Pleasingly Parallel Computations

Client/Server computing:

- 1 server provides hardware or software service,
- 2 clients make requests to server.

Typically, clients and servers run on different computers.

Operations in server side scripting:

- 1 define addresses, port numbers, sockets,
- 2 wait to accept client requests,
- 3 service the requests of clients.

For multiple clients, server could either

- 1 wait till every one is connected before serving,
- 2 or accept connections and serve one at a time.

In either protocol, clients are blocked waiting.

# Server for Multiple Clients

avoid to block clients with waiting

## Server for Multiple Clients

avoid to block clients with waiting

using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server

code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically

code to manage handler threads

## Pleasingly Parallel Computations

Client/Server computing:

- 1 server provides hardware or software service,
- 2 clients make requests to server.

Typically, clients and servers run on different computers.

Operations in server side scripting:

- 1 define addresses, port numbers, sockets,
- 2 wait to accept client requests,
- 3 service the requests of clients.

For multiple clients, server could either

- 1 wait till every one is connected before serving,
- 2 or accept connections and serve one at a time.

In either protocol, clients are blocked waiting.

# Server for Multiple Clients

avoid to block clients with waiting

## Server for Multiple Clients

avoid to block clients with waiting

using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server

code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically

code to manage handler threads

## Pleasingly Parallel Computations

Client/Server computing:

- 1 server provides hardware or software service,
- 2 clients make requests to server.

Typically, clients and servers run on different computers.

Operations in server side scripting:

- 1 define addresses, port numbers, sockets,
- 2 wait to accept client requests,
- 3 service the requests of clients.

For multiple clients, server could either

- 1 wait till every one is connected before serving,
- 2 or accept connections and serve one at a time.

In either protocol, clients are blocked waiting.

# Server for Multiple Clients

avoid to block clients with waiting

Client/Server computing:

- 1 server provides hardware or software service,
- 2 clients make requests to server.

Typically, clients and servers run on different computers.

Operations in server side scripting:

- 1 define addresses, port numbers, sockets,
- 2 wait to accept client requests,
- 3 service the requests of clients.

For multiple clients, server could either

- 1 wait till every one is connected before serving,
- 2 or accept connections and serve one at a time.

In either protocol, clients are blocked waiting.

# Server for Multiple Clients

avoid to block clients with waiting

## Server for Multiple Clients

avoid to block clients with waiting

using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server

code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically

code to manage handler threads

## Pleasingly Parallel Computations

Client/Server computing:

- 1 server provides hardware or software service,
- 2 clients make requests to server.

Typically, clients and servers run on different computers.

Operations in server side scripting:

- 1 define addresses, port numbers, sockets,
- 2 wait to accept client requests,
- 3 service the requests of clients.

For multiple clients, server could either

- 1 wait till every one is connected before serving,
- 2 or accept connections and serve one at a time.

In either protocol, clients are blocked waiting.

# Server for Multiple Clients

avoid to block clients with waiting

## Server for Multiple Clients

avoid to block clients with waiting

using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server

code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically

code to manage handler threads

## Pleasingly Parallel Computations

Client/Server computing:

- 1 server provides hardware or software service,
- 2 clients make requests to server.

Typically, clients and servers run on different computers.

Operations in server side scripting:

- 1 define addresses, port numbers, sockets,
- 2 wait to accept client requests,
- 3 service the requests of clients.

For multiple clients, server could either

- 1 wait till every one is connected before serving,
- 2 or accept connections and serve one at a time.

In either protocol, clients are blocked waiting.

2 Apr 2010

Server for  
Multiple  
Clientsavoid to block clients  
with waitingusing sockets and  
threadsWaiting for  
Data from 3  
Clientsrunning a simple  
multithreaded servercode for client and  
servern Handler  
Threads for m  
Client  
Requestsmanaging threads  
dynamicallycode to manage  
handler threadsPleasingly  
Parallel  
Computations

# A Print Server

Imagine a printer shared between multiple users connected in a computer network.

Handling a request to print could include:

- 1 accepting path name of the file,
- 2 verifying whether the file exists,
- 3 placing request in the printer queue.

Each of these three stages involves network communication with possible delays during which other request could be handled.

A multithreaded print server has multiple handlers running simultaneously.

Server for  
Multiple  
Clientsavoid to block clients  
with waitingusing sockets and  
threadsWaiting for  
Data from 3  
Clientsrunning a simple  
multithreaded servercode for client and  
servern Handler  
Threads for m  
Client  
Requestsmanaging threads  
dynamicallycode to manage  
handler threadsPleasingly  
Parallel  
Computations

# A Print Server

Imagine a printer shared between multiple users connected in a computer network.

Handling a request to print could include:

- 1 accepting path name of the file,
- 2 verifying whether the file exists,
- 3 placing request in the printer queue.

Each of these three stages involves network communication with possible delays during which other request could be handled.

A multithreaded print server has multiple handlers running simultaneously.

Server for  
Multiple  
Clients

avoid to block clients  
with waiting

using sockets and  
threads

Waiting for  
Data from 3  
Clients

running a simple  
multithreaded server

code for client and  
server

n Handler  
Threads for m  
Client  
Requests

managing threads  
dynamically

code to manage  
handler threads

Pleasingly  
Parallel  
Computations

# A Print Server

Imagine a printer shared between multiple users connected in a computer network.

Handling a request to print could include:

- 1 accepting path name of the file,
- 2 verifying whether the file exists,
- 3 placing request in the printer queue.

Each of these three stages involves network communication with possible delays during which other request could be handled.

A multithreaded print server has multiple handlers running simultaneously.

# A Print Server

## Server for Multiple Clients

avoid to block clients with waiting

using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server

code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically

code to manage handler threads

## Pleasingly Parallel Computations

Imagine a printer shared between multiple users connected in a computer network.

Handling a request to print could include:

- 1 accepting path name of the file,
- 2 verifying whether the file exists,
- 3 placing request in the printer queue.

Each of these three stages involves network communication with possible delays during which other request could be handled.

A multithreaded print server has multiple handlers running simultaneously.

# A Print Server

## Server for Multiple Clients

avoid to block clients with waiting

using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server

code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically

code to manage handler threads

## Pleasingly Parallel Computations

Imagine a printer shared between multiple users connected in a computer network.

Handling a request to print could include:

- 1 accepting path name of the file,
- 2 verifying whether the file exists,
- 3 placing request in the printer queue.

Each of these three stages involves network communication with possible delays during which other request could be handled.

A multithreaded print server has multiple handlers running simultaneously.

# A Print Server

## Server for Multiple Clients

avoid to block clients with waiting

using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server

code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically

code to manage handler threads

## Pleasingly Parallel Computations

Imagine a printer shared between multiple users connected in a computer network.

Handling a request to print could include:

- 1 accepting path name of the file,
- 2 verifying whether the file exists,
- 3 placing request in the printer queue.

Each of these three stages involves network communication with possible delays during which other request could be handled.

A multithreaded print server has multiple handlers running simultaneously.

Server for  
Multiple  
Clientsavoid to block clients  
with waitingusing sockets and  
threadsWaiting for  
Data from 3  
Clientsrunning a simple  
multithreaded servercode for client and  
servern Handler  
Threads for m  
Client  
Requestsmanaging threads  
dynamicallycode to manage  
handler threadsPleasingly  
Parallel  
Computations

# Waiting for Client Information

interactive dialogues

Imagine an automated bank teller:

- 1 ask for account number,
- 2 client must provide password,
- 3 prompt for menu selection, etc...

Each of these questions may lead to extra delays during which other clients could be serviced.

Running multiple threads on server will utilize the time server is idle waiting for one client for the benefit of the request of other clients.

Server for  
Multiple  
Clientsavoid to block clients  
with waitingusing sockets and  
threadsWaiting for  
Data from 3  
Clientsrunning a simple  
multithreaded servercode for client and  
servern Handler  
Threads for m  
Client  
Requestsmanaging threads  
dynamicallycode to manage  
handler threadsPleasingly  
Parallel  
Computations

# Waiting for Client Information

interactive dialogues

Imagine an automated bank teller:

- 1 ask for account number,
- 2 client must provide password,
- 3 prompt for menu selection, etc...

Each of these questions may lead to extra delays during which other clients could be serviced.

Running multiple threads on server will utilize the time server is idle waiting for one client for the benefit of the request of other clients.

# Waiting for Client Information

interactive dialogues

Imagine an automated bank teller:

- 1 ask for account number,
- 2 client must provide password,
- 3 prompt for menu selection, etc...

Each of these questions may lead to extra delays during which other clients could be serviced.

Running multiple threads on server will utilize the time server is idle waiting for one client for the benefit of the request of other clients.

# Waiting for Client Information

interactive dialogues

Imagine an automated bank teller:

- 1 ask for account number,
- 2 client must provide password,
- 3 prompt for menu selection, etc...

Each of these questions may lead to extra delays during which other clients could be serviced.

Running multiple threads on server will utilize the time server is idle waiting for one client for the benefit of the request of other clients.

# Waiting for Client Information

interactive dialogues

## Server for Multiple Clients

avoid to block clients with waiting

using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server

code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically

code to manage handler threads

## Pleasingly Parallel Computations

Imagine an automated bank teller:

- 1 ask for account number,
- 2 client must provide password,
- 3 prompt for menu selection, etc...

Each of these questions may lead to extra delays during which other clients could be serviced.

Running multiple threads on server will utilize the time server is idle waiting for one client for the benefit of the request of other clients.

# Multithreaded Servers

## Server for Multiple Clients

avoid to block clients with waiting

using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server

code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically

code to manage handler threads

## Pleasingly Parallel Computations

### 1 Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

### 2 Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

### 3 n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

### 4 Pleasingly Parallel Computations

# Using Sockets and Threads

object oriented design

## Server for Multiple Clients

avoid to block clients with waiting

using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

The main program will

- 1 define address, port numbers, server socket,
- 2 launch the handler threads of the server.

Typically we will have as many handler threads as the number of connections the server listens to.

Inheriting from the class `Thread`,

- 1 the constructor of the handler will store a reference to the socket server as data attribute,
- 2 the `run` contains the code to accept connections and handle requests.

Locks are needed to manage shared resources safely.

# Using Sockets and Threads

object oriented design

## Server for Multiple Clients

avoid to block clients with waiting

using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

The main program will

- 1 define address, port numbers, server socket,
- 2 launch the handler threads of the server.

Typically we will have as many handler threads as the number of connections the server listens to.

Inheriting from the class `Thread`,

- 1 the constructor of the handler will store a reference to the socket server as data attribute,
- 2 the `run` contains the code to accept connections and handle requests.

Locks are needed to manage shared resources safely.

# Using Sockets and Threads

object oriented design

Server for  
Multiple  
Clients

avoid to block clients  
with waiting

using sockets and  
threads

Waiting for  
Data from 3  
Clients

running a simple  
multithreaded server  
code for client and  
server

n Handler  
Threads for m  
Client  
Requests

managing threads  
dynamically  
code to manage  
handler threads

Pleasingly  
Parallel  
Computations

The main program will

- 1 define address, port numbers, server socket,
- 2 launch the handler threads of the server.

Typically we will have as many handler threads as the number of connections the server listens to.

Inheriting from the class `Thread`,

- 1 the constructor of the handler will store a reference to the socket server as data attribute,
- 2 the `run` contains the code to accept connections and handle requests.

Locks are needed to manage shared resources safely.

# Using Sockets and Threads

object oriented design

## Server for Multiple Clients

avoid to block clients with waiting

using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

The main program will

- 1 define address, port numbers, server socket,
- 2 launch the handler threads of the server.

Typically we will have as many handler threads as the number of connections the server listens to.

Inheriting from the class `Thread`,

- 1 the constructor of the handler will store a reference to the socket server as data attribute,
- 2 the `run` contains the code to accept connections and handle requests.

Locks are needed to manage shared resources safely.

# Using Sockets and Threads

object oriented design

Server for  
Multiple  
Clients

avoid to block clients  
with waiting

using sockets and  
threads

Waiting for  
Data from 3  
Clients

running a simple  
multithreaded server  
code for client and  
server

n Handler  
Threads for m  
Client  
Requests

managing threads  
dynamically  
code to manage  
handler threads

Pleasingly  
Parallel  
Computations

The main program will

- 1 define address, port numbers, server socket,
- 2 launch the handler threads of the server.

Typically we will have as many handler threads as the number of connections the server listens to.

Inheriting from the class `Thread`,

- 1 the constructor of the handler will store a reference to the socket server as data attribute,
- 2 the `run` contains the code to accept connections and handle requests.

Locks are needed to manage shared resources safely.

# Using Sockets and Threads

object oriented design

## Server for Multiple Clients

avoid to block clients with waiting

using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

The main program will

- 1 define address, port numbers, server socket,
- 2 launch the handler threads of the server.

Typically we will have as many handler threads as the number of connections the server listens to.

Inheriting from the class `Thread`,

- 1 the constructor of the handler will store a reference to the socket server as data attribute,
- 2 the `run` contains the code to accept connections and handle requests.

Locks are needed to manage shared resources safely.

# Multithreaded Servers

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

- 1 Server for Multiple Clients  
avoid to block clients with waiting  
using sockets and threads
- 2 **Waiting for Data from 3 Clients**  
running a simple multithreaded server  
code for client and server
- 3 n Handler Threads for m Client Requests  
managing threads dynamically  
code to manage handler threads
- 4 Pleasingly Parallel Computations

# Waiting for Data from 3 Clients

a simple multithreaded server

Suppose 3 clients send a message to a server.

Application: collect a vote from three people.

Clients behave as follows:

- 1 may connect at any time with the server,
- 2 getting message typed in takes time.

Multithreaded server listens to 3 clients:

- 1 three threads can handle requests,
- 2 each thread simply receives message,
- 3 server closes after three threads are done.

# Waiting for Data from 3 Clients

a simple multithreaded server

Suppose 3 clients send a message to a server.  
Application: collect a vote from three people.

Clients behave as follows:

- 1 may connect at any time with the server,
- 2 getting message typed in takes time.

Multithreaded server listens to 3 clients:

- 1 three threads can handle requests,
- 2 each thread simply receives message,
- 3 server closes after three threads are done.

# Waiting for Data from 3 Clients

a simple multithreaded server

Suppose 3 clients send a message to a server.  
Application: collect a vote from three people.

Clients behave as follows:

- 1 may connect at any time with the server,
- 2 getting message typed in takes time.

Multithreaded server listens to 3 clients:

- 1 three threads can handle requests,
- 2 each thread simply receives message,
- 3 server closes after three threads are done.

# Waiting for Data from 3 Clients

a simple multithreaded server

Suppose 3 clients send a message to a server.  
Application: collect a vote from three people.

Clients behave as follows:

- 1 may connect at any time with the server,
- 2 getting message typed in takes time.

Multithreaded server listens to 3 clients:

- 1 three threads can handle requests,
- 2 each thread simply receives message,
- 3 server closes after three threads are done.

# Waiting for Data from 3 Clients

a simple multithreaded server

Suppose 3 clients send a message to a server.  
Application: collect a vote from three people.

Clients behave as follows:

- 1 may connect at any time with the server,
- 2 getting message typed in takes time.

Multithreaded server listens to 3 clients:

- 1 three threads can handle requests,
- 2 each thread simply receives message,
- 3 server closes after three threads are done.

# Waiting for Data from 3 Clients

a simple multithreaded server

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

Suppose 3 clients send a message to a server.  
Application: collect a vote from three people.

Clients behave as follows:

- 1 may connect at any time with the server,
- 2 getting message typed in takes time.

Multithreaded server listens to 3 clients:

- 1 three threads can handle requests,
- 2 each thread simply receives message,
- 3 server closes after three threads are done.

# Waiting for Data from 3 Clients

a simple multithreaded server

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

Suppose 3 clients send a message to a server.  
Application: collect a vote from three people.

Clients behave as follows:

- 1 may connect at any time with the server,
- 2 getting message typed in takes time.

Multithreaded server listens to 3 clients:

- 1 three threads can handle requests,
- 2 each thread simply receives message,
- 3 server closes after three threads are done.

2 Apr 2010

# Running Multithreaded server

accepting a message from three clients

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

```
$ python mtserver.py
give #clients : 3
server is ready for 3 clients
server starts 3 threads
0 accepted request from ('127.0.0.1', 49153)
0 waits for data
1 accepted request from ('127.0.0.1', 49154)
1 waits for data
1 received this is B
0 received this is A
2 accepted request from ('127.0.0.1', 49155)
2 waits for data
2 received this is C
$
```

2 Apr 2010

# Running Multithreaded server

accepting a message from three clients

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

```
$ python mtserver.py
give #clients : 3
server is ready for 3 clients
server starts 3 threads
0 accepted request from ('127.0.0.1', 49153)
0 waits for data
1 accepted request from ('127.0.0.1', 49154)
1 waits for data
1 received this is B
0 received this is A
2 accepted request from ('127.0.0.1', 49155)
2 waits for data
2 received this is C
$
```

2 Apr 2010

# Running Multithreaded server

accepting a message from three clients

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

```
$ python mtserver.py
give #clients : 3
server is ready for 3 clients
server starts 3 threads
0 accepted request from ('127.0.0.1', 49153)
0 waits for data
1 accepted request from ('127.0.0.1', 49154)
1 waits for data
1 received this is B
0 received this is A
2 accepted request from ('127.0.0.1', 49155)
2 waits for data
2 received this is C
$
```

2 Apr 2010

# Running Multithreaded server

accepting a message from three clients

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

```
$ python mtserver.py
give #clients : 3
server is ready for 3 clients
server starts 3 threads
0 accepted request from ('127.0.0.1', 49153)
0 waits for data
1 accepted request from ('127.0.0.1', 49154)
1 waits for data
1 received this is B
0 received this is A
2 accepted request from ('127.0.0.1', 49155)
2 waits for data
2 received this is C
$
```

2 Apr 2010

# Running Multithreaded server

accepting a message from three clients

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

```
$ python mtserver.py
give #clients : 3
server is ready for 3 clients
server starts 3 threads
0 accepted request from ('127.0.0.1', 49153)
0 waits for data
1 accepted request from ('127.0.0.1', 49154)
1 waits for data
1 received this is B
0 received this is A
2 accepted request from ('127.0.0.1', 49155)
2 waits for data
2 received this is C
$
```

2 Apr 2010

# Client Sessions

run simultaneously in separate terminal windows

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

```
$ python mtclient.py  
client is connected  
Give message : this is A  
$
```

```
$ python mtclient.py  
client is connected  
Give message : this is B  
$
```

```
$ python mtclient.py  
client is connected  
Give message : this is C
```

2 Apr 2010

# Client Sessions

run simultaneously in separate terminal windows

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

```
$ python mtclient.py  
client is connected  
Give message : this is A  
$
```

```
$ python mtclient.py  
client is connected  
Give message : this is B  
$
```

```
$ python mtclient.py  
client is connected  
Give message : this is C
```

2 Apr 2010

# Client Sessions

run simultaneously in separate terminal windows

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

```
$ python mtclient.py  
client is connected  
Give message : this is A  
$
```

```
$ python mtclient.py  
client is connected  
Give message : this is B  
$
```

```
$ python mtclient.py  
client is connected  
Give message : this is C
```

# Multithreaded Servers

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server  
**code for client and server**

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

- 1 Server for Multiple Clients  
avoid to block clients with waiting  
using sockets and threads
- 2 **Waiting for Data from 3 Clients**  
running a simple multithreaded server  
**code for client and server**
- 3 n Handler Threads for m Client Requests  
managing threads dynamically  
code to manage handler threads
- 4 Pleasingly Parallel Computations

# Code for Client

sending a user given message to server

```
from socket import *

hostname = 'localhost' # on same host
number = 11267         # same port number
buffer = 80           # size of the buffer

server_address = (hostname, number)
client = socket(AF_INET, SOCK_STREAM)
client.connect(server_address)

print 'client is connected'
data = raw_input('Give message : ')
client.send(data)

client.close()
```

# Function `main()` in Server

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server

code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

```
def main():
    """
    Prompts for number of connections,
    starts the server and handler threads.
    """
    n = input('give #clients : ')
    b = 80; server = connect(n,b)
    print 'server is ready for %d clients' % n
    T = []
    for i in range(0,n):
        T.append(Handler(str(i),server,b))
    print 'server starts %d threads' % n
    for t in T: t.start()
    server.close()
```

# Function `main()` in Server

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server

code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

```
def main():
    """
    Prompts for number of connections,
    starts the server and handler threads.
    """
    n = input('give #clients : ')
    b = 80; server = connect(n,b)
    print 'server is ready for %d clients' % n
    T = []
    for i in range(0,n):
        T.append(Handler(str(i),server,b))
    print 'server starts %d threads' % n
    for t in T: t.start()
    server.close()
```

# Function `main()` in Server

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server

code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

```
def main():
    """
    Prompts for number of connections,
    starts the server and handler threads.
    """
    n = input('give #clients : ')
    b = 80; server = connect(n,b)
    print 'server is ready for %d clients' % n
    T = []
    for i in range(0,n):
        T.append(Handler(str(i),server,b))
    print 'server starts %d threads' % n
    for t in T: t.start()
    server.close()
```

# The Function `connect()`

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server

## code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

```
def connect(n):
    """
    Connects a server to listen to n clients,
    buffer size is b. Returns the socket.
    """
    hostname = '' # to use any address
    number = 11267 # number for the port
    buffer = b # size of the buffer
    server_address = (hostname, number)
    server = socket(AF_INET, SOCK_STREAM)
    server.bind(server_address)
    server.listen(n)
    return server
```

# The Function `connect()`

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server

code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

```
def connect(n):
    """
    Connects a server to listen to n clients,
    buffer size is b. Returns the socket.
    """
    hostname = '' # to use any address
    number = 11267 # number for the port
    buffer = b # size of the buffer
    server_address = (hostname, number)
    server = socket(AF_INET, SOCK_STREAM)
    server.bind(server_address)
    server.listen(n)
    return server
```

# The Function `connect()`

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server

code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

```
def connect(n):
    """
    Connects a server to listen to n clients,
    buffer size is b. Returns the socket.
    """
    hostname = '' # to use any address
    number = 11267 # number for the port
    buffer = b # size of the buffer
    server_address = (hostname, number)
    server = socket(AF_INET, SOCK_STREAM)
    server.bind(server_address)
    server.listen(n)
    return server
```

# The Handler Class

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server

## code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

```
class Handler(Thread):
    """
    Defines handler threads.
    """
    def __init__(self,n,s,b):
        """
        Name of handler is n, server
        socket is s, buffer size is b.
        """
    def run(self):
        """
        Handler accepts connection,
        prints message received from client
        """
```

# The Handler Class

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server

## code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

```
class Handler(Thread):
    """
    Defines handler threads.
    """
    def __init__(self,n,s,b):
        """
        Name of handler is n, server
        socket is s, buffer size is b.
        """
    def run(self):
        """
        Handler accepts connection,
        prints message received from client
        """
```

# Constructor of Handler Thread

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

The object data attributes for each handler are server socket and buffer size.

```
def __init__(self,n,s,b):  
    """  
    Name of handler is n, server  
    socket is s, buffer size is b.  
    """  
    Thread.__init__(self,name=n)  
    self.sv = s  
    self.bf = b
```

# Constructor of Handler Thread

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

The object data attributes for each handler are server socket and buffer size.

```
def __init__(self,n,s,b):  
    """  
    Name of handler is n, server  
    socket is s, buffer size is b.  
    """  
    Thread.__init__(self,name=n)  
    self.sv = s  
    self.bf = b
```

# Code for the Handler Thread

## Server for Multiple Clients

avoid to block clients  
with waiting  
using sockets and  
threads

## Waiting for Data from 3 Clients

running a simple  
multithreaded server

## code for client and server

## n Handler Threads for m Client Requests

managing threads  
dynamically  
code to manage  
handler threads

## Pleasingly Parallel Computations

```
def run(self):
    """
    Handler accepts connection,
    prints message received from client.
    """
    n = self.getName()
    server = self.sv
    buffer = self.bf
    client, client_address = server.accept()
    print n + ' accepted request from ', \
          client_address
    print n + ' waits for data'
    data = client.recv(buffer)
    print n + ' received ', data
```

## Code for the Handler Thread

Server for  
Multiple  
Clients

avoid to block clients  
with waiting  
using sockets and  
threads

Waiting for  
Data from 3  
Clients

running a simple  
multithreaded server

code for client and  
servern Handler  
Threads for m  
Client  
Requests

managing threads  
dynamically  
code to manage  
handler threads

Pleasingly  
Parallel  
Computations

```
def run(self):
    """
    Handler accepts connection,
    prints message received from client.
    """
    n = self.getName()
    server = self.sv
    buffer = self.bf
    client, client_address = server.accept()
    print n + ' accepted request from ', \
          client_address
    print n + ' waits for data'
    data = client.recv(buffer)
    print n + ' received ', data
```

## Code for the Handler Thread

Server for  
Multiple  
Clients

avoid to block clients  
with waiting  
using sockets and  
threads

Waiting for  
Data from 3  
Clients

running a simple  
multithreaded server

code for client and  
server

n Handler  
Threads for m  
Client  
Requests

managing threads  
dynamically  
code to manage  
handler threads

Pleasingly  
Parallel  
Computations

```
def run(self):
    """
    Handler accepts connection,
    prints message received from client.
    """
    n = self.getName()
    server = self.sv
    buffer = self.bf
    client, client_address = server.accept()
    print n + ' accepted request from ', \
          client_address
    print n + ' waits for data'
    data = client.recv(buffer)
    print n + ' received ', data
```

# Multithreaded Servers

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

- 1 Server for Multiple Clients  
avoid to block clients with waiting  
using sockets and threads
- 2 Waiting for Data from 3 Clients  
running a simple multithreaded server  
code for client and server
- 3 n Handler Threads for m Client Requests  
managing threads dynamically  
code to manage handler threads
- 4 Pleasingly Parallel Computations

# $n$ Handlers for $m$ Requests

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

## $n$ Handler Threads for $m$ Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

Consider the following situation:

- 1  $n$  threads are available to handle requests,
- 2 a total number of  $m$  requests will be handled,
- 3  $m$  is always larger than  $n$ .

Example application: calling center with 8 operators is paid to handle 100 calls per day.

Some operators handle few calls that take long time.

Other operators handle many short calls.

# $n$ Handlers for $m$ Requests

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

## $n$ Handler Threads for $m$ Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

Consider the following situation:

- 1  $n$  threads are available to handle requests,
- 2 a total number of  $m$  requests will be handled,
- 3  $m$  is always larger than  $n$ .

Example application: calling center with 8 operators is paid to handle 100 calls per day.

Some operators handle few calls that take long time.

Other operators handle many short calls.

# $n$ Handlers for $m$ Requests

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

## $n$ Handler Threads for $m$ Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

Consider the following situation:

- 1  $n$  threads are available to handle requests,
- 2 a total number of  $m$  requests will be handled,
- 3  $m$  is always larger than  $n$ .

Example application: calling center with 8 operators is paid to handle 100 calls per day.

Some operators handle few calls that take long time.

Other operators handle many short calls.

# $n$ Handlers for $m$ Requests

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

## $n$ Handler Threads for $m$ Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

Consider the following situation:

- 1  $n$  threads are available to handle requests,
- 2 a total number of  $m$  requests will be handled,
- 3  $m$  is always larger than  $n$ .

Example application: calling center with 8 operators is paid to handle 100 calls per day.

Some operators handle few calls that take long time.

Other operators handle many short calls.

# Management Handlers Threads

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

Previous multithreaded server, after starting threads:

- 1 request of client accepted by one handler,
- 2 request is handled by a thread,
- 3 after handling request, the thread dies.

Simple protocol, but server must wait till all threads have received and handled their request.

To handle  $m$  requests by  $n$  server threads:

- 1 server checks status of thread  $t$ : `t.isAlive()`,
- 2 if thread dead: increase counter of requests handled,
- 3 start a new thread if number of requests still to be handled is larger than number of live threads.

# Management Handlers Threads

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

Previous multithreaded server, after starting threads:

- 1 request of client accepted by one handler,
- 2 request is handled by a thread,
- 3 after handling request, the thread dies.

Simple protocol, but server must wait till all threads have received and handled their request.

To handle  $m$  requests by  $n$  server threads:

- 1 server checks status of thread  $t$ : `t.isAlive()`,
- 2 if thread dead: increase counter of requests handled,
- 3 start a new thread if number of requests still to be handled is larger than number of live threads.

# Management Handlers Threads

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

Previous multithreaded server, after starting threads:

- 1 request of client accepted by one handler,
- 2 request is handled by a thread,
- 3 after handling request, the thread dies.

Simple protocol, but server must wait till all threads have received and handled their request.

To handle  $m$  requests by  $n$  server threads:

- 1 server checks status of thread  $t$ : `t.isAlive()`,
- 2 if thread dead: increase counter of requests handled,
- 3 start a new thread if number of requests still to be handled is larger than number of live threads.

# Management Handlers Threads

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

Previous multithreaded server, after starting threads:

- 1 request of client accepted by one handler,
- 2 request is handled by a thread,
- 3 after handling request, the thread dies.

Simple protocol, but server must wait till all threads have received and handled their request.

To handle  $m$  requests by  $n$  server threads:

- 1 server checks status of thread  $t$ : `t.isAlive()`,
- 2 if thread dead: increase counter of requests handled,
- 3 start a new thread if number of requests still to be handled is larger than number of live threads.

# Management Handlers Threads

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

Previous multithreaded server, after starting threads:

- 1 request of client accepted by one handler,
- 2 request is handled by a thread,
- 3 after handling request, the thread dies.

Simple protocol, but server must wait till all threads have received and handled their request.

To handle  $m$  requests by  $n$  server threads:

- 1 server checks status of thread  $t$ : `t.isAlive()`,
- 2 if thread dead: increase counter of requests handled,
- 3 start a new thread if number of requests still to be handled is larger than number of live threads.

# Management Handlers Threads

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

Previous multithreaded server, after starting threads:

- 1 request of client accepted by one handler,
- 2 request is handled by a thread,
- 3 after handling request, the thread dies.

Simple protocol, but server must wait till all threads have received and handled their request.

To handle  $m$  requests by  $n$  server threads:

- 1 server checks status of thread  $t$ : `t.isAlive()`,
- 2 if thread dead: increase counter of requests handled,
- 3 start a new thread if number of requests still to be handled is larger than number of live threads.

2 Apr 2010

# Running the Server

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

```
$ python mtnserver.py
give #clients : 3
give #requests : 7
server is ready for 3 clients
server starts 3 threads
0 is alive, cnt = 0
1 is alive, cnt = 0
2 is alive, cnt = 0
0 accepted request from ('127.0.0.1', 49234)
0 waits for data
0 is alive, cnt = 0
... to be continued ...
```

# Running the Server

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

```
$ python mtnserver.py
give #clients : 3
give #requests : 7
server is ready for 3 clients
server starts 3 threads
0 is alive, cnt = 0
1 is alive, cnt = 0
2 is alive, cnt = 0
0 accepted request from ('127.0.0.1', 49234)
0 waits for data
0 is alive, cnt = 0
... to be continued ...
```

# Running the Server continued

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

```

1 is alive, cnt = 0
2 is alive, cnt = 0
0 received 1
1 accepted request from ('127.0.0.1', 49235)
1 waits for data
0 handled request 1
restarting 0
1 is alive, cnt = 1
2 is alive, cnt = 1
1 received 2
0 is alive, cnt = 1
1 handled request 2
restarting 1
2 is alive, cnt = 2
. . . .

```

2 Apr 2010

# Running the Server continued

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

```

1 is alive, cnt = 0
2 is alive, cnt = 0
0 received 1
1 accepted request from ('127.0.0.1', 49235)
1 waits for data
0 handled request 1
restarting 0
1 is alive, cnt = 1
2 is alive, cnt = 1
1 received 2
0 is alive, cnt = 1
1 handled request 2
restarting 1
2 is alive, cnt = 2
. . . .

```

2 Apr 2010

# Running the Server continued

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

```

1 is alive, cnt = 0
2 is alive, cnt = 0
0 received 1
1 accepted request from ('127.0.0.1', 49235)
1 waits for data
0 handled request 1
restarting 0
1 is alive, cnt = 1
2 is alive, cnt = 1
1 received 2
0 is alive, cnt = 1
1 handled request 2
restarting 1
2 is alive, cnt = 2
....

```

# Multithreaded Servers

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

- 1 Server for Multiple Clients  
avoid to block clients with waiting  
using sockets and threads
- 2 Waiting for Data from 3 Clients  
running a simple multithreaded server  
code for client and server
- 3 n Handler Threads for m Client Requests  
managing threads dynamically  
code to manage handler threads
- 4 Pleasingly Parallel Computations

## Modified main()

```
def main():
    """
    Prompts for number of connections,
    starts the server and handler threads.
    """
    n = input('give #clients : ')
    m = input('give #requests : ')
    server = connect(n,80)
    print 'server is ready for %d clients' % n
    T = []
    for i in range(0,n):
        T.append(Handler(str(i),server,80))
    print 'server starts %d threads' % n
    for t in T: t.start()
    manage(T,m,server)
    server.close()
```

Server for  
Multiple  
Clients

avoid to block clients  
with waiting  
using sockets and  
threads

Waiting for  
Data from 3  
Clients

running a simple  
multithreaded server  
code for client and  
server

n Handler  
Threads for m  
Client  
Requests

managing threads  
dynamically  
code to manage  
handler threads

Pleasingly  
Parallel  
Computations

## Modified main()

```
def main():
    """
    Prompts for number of connections,
    starts the server and handler threads.
    """
    n = input('give #clients : ')
    m = input('give #requests : ')
    server = connect(n,80)
    print 'server is ready for %d clients' % n
    T = []
    for i in range(0,n):
        T.append(Handler(str(i),server,80))
    print 'server starts %d threads' % n
    for t in T: t.start()
    manage(T,m,server)
    server.close()
```

Server for  
Multiple  
Clients

avoid to block clients  
with waiting  
using sockets and  
threads

Waiting for  
Data from 3  
Clients

running a simple  
multithreaded server  
code for client and  
server

n Handler  
Threads for m  
Client  
Requests

managing threads  
dynamically  
code to manage  
handler threads

Pleasingly  
Parallel  
Computations

# Code to manage Handler Threads

A thread cannot be restarted,  
but we can create a new thread with same name.

Recall that the server maintains a list of threads.

```
def restart(T,s,i):
    """
    Deletes the i-th dead thread from T
    and inserts a new thread at position i.
    The socket server is s.  Returns new T.
    """
    del T[i]
    T.insert(i,Handler(str(i),s,80))
    T[i].start()
    return T
```

Server for  
Multiple  
Clients

avoid to block clients  
with waiting  
using sockets and  
threads

Waiting for  
Data from 3  
Clients

running a simple  
multithreaded server  
code for client and  
server

n Handler  
Threads for m  
Client  
Requests

managing threads  
dynamically  
code to manage  
handler threads

Pleasingly  
Parallel  
Computations

# Code to manage Handler Threads

A thread cannot be restarted,  
but we can create a new thread with same name.

Recall that the server maintains a list of threads.

```
def restart(T,s,i):
    """
    Deletes the i-th dead thread from T
    and inserts a new thread at position i.
    The socket server is s.  Returns new T.
    """
    del T[i]
    T.insert(i,Handler(str(i),s,80))
    T[i].start()
    return T
```

Server for  
Multiple  
Clients

avoid to block clients  
with waiting  
using sockets and  
threads

Waiting for  
Data from 3  
Clients

running a simple  
multithreaded server  
code for client and  
server

n Handler  
Threads for m  
Client  
Requests

managing threads  
dynamically  
code to manage  
handler threads

Pleasingly  
Parallel  
Computations

# Code to manage Handler Threads

A thread cannot be restarted,  
but we can create a new thread with same name.

Recall that the server maintains a list of threads.

```
def restart(T,s,i):
    """
    Deletes the i-th dead thread from T
    and inserts a new thread at position i.
    The socket server is s.  Returns new T.
    """
    del T[i]
    T.insert(i,Handler(str(i),s,80))
    T[i].start()
    return T
```

Server for  
Multiple  
Clients

avoid to block clients  
with waiting  
using sockets and  
threads

Waiting for  
Data from 3  
Clients

running a simple  
multithreaded server  
code for client and  
server

n Handler  
Threads for m  
Client  
Requests

managing threads  
dynamically  
code to manage  
handler threads

Pleasingly  
Parallel  
Computations

# Code to manage Handler Threads

A thread cannot be restarted,  
but we can create a new thread with same name.

Recall that the server maintains a list of threads.

```
def restart(T,s,i):
    """
    Deletes the i-th dead thread from T
    and inserts a new thread at position i.
    The socket server is s.  Returns new T.
    """
    del T[i]
    T.insert(i,Handler(str(i),s,80))
    T[i].start()
    return T
```

Server for  
Multiple  
Clients

avoid to block clients  
with waiting  
using sockets and  
threads

Waiting for  
Data from 3  
Clients

running a simple  
multithreaded server  
code for client and  
server

n Handler  
Threads for m  
Client  
Requests

managing threads  
dynamically  
code to manage  
handler threads

Pleasingly  
Parallel  
Computations

## the Function manage()

Server for  
Multiple  
Clients

avoid to block clients  
with waiting  
using sockets and  
threads

Waiting for  
Data from 3  
Clients

running a simple  
multithreaded server  
code for client and  
server

n Handler  
Threads for m  
Client  
Requests

managing threads  
dynamically  
code to manage  
handler threads

Pleasingly  
Parallel  
Computations

```
def manage(T,m,s):
    """
    Manages threads in T, restarts threads
    until m requests have been handled.
    The socket server is s.
    """
    cnt = 0; n = len(T); dead = []
    while cnt < m:
        time.sleep(2)
        for i in range(0,n):
            if i in dead:
                pass
            elif T[i].isAlive():
                print i, 'is alive, cnt =', cnt
            else:
                # to be continued
```

## the Function manage()

Server for  
Multiple  
Clients

avoid to block clients  
with waiting  
using sockets and  
threads

Waiting for  
Data from 3  
Clients

running a simple  
multithreaded server  
code for client and  
server

n Handler  
Threads for m  
Client  
Requests

managing threads  
dynamically  
code to manage  
handler threads

Pleasingly  
Parallel  
Computations

```
def manage(T,m,s):
    """
    Manages threads in T, restarts threads
    until m requests have been handled.
    The socket server is s.
    """
    cnt = 0; n = len(T); dead = []
    while cnt < m:
        time.sleep(2)
        for i in range(0,n):
            if i in dead:
                pass
            elif T[i].isAlive():
                print i, 'is alive, cnt =', cnt
            else:
                # to be continued
```

Server for  
Multiple  
Clients

avoid to block clients  
with waiting  
using sockets and  
threads

Waiting for  
Data from 3  
Clients

running a simple  
multithreaded server  
code for client and  
server

n Handler  
Threads for m  
Client  
Requests

managing threads  
dynamically  
code to manage  
handler threads

Pleasingly  
Parallel  
Computations

## Code for manage() continued

```
else:
    cnt = cnt + 1
    print '%d handled request %d' \
          % (i,cnt)
    if cnt >= m:
        break
    elif cnt <= m-n:
        print 'restarting', i
        T = restart(T,s,i)
    else:
        dead.append(i)
```

Observe that the counter is manipulated only by main server, locks are not needed.

Server for  
Multiple  
Clients

avoid to block clients  
with waiting  
using sockets and  
threads

Waiting for  
Data from 3  
Clients

running a simple  
multithreaded server  
code for client and  
server

n Handler  
Threads for m  
Client  
Requests

managing threads  
dynamically  
code to manage  
handler threads

Pleasingly  
Parallel  
Computations

## Code for manage() continued

```
else:
    cnt = cnt + 1
    print '%d handled request %d' \
          % (i,cnt)
    if cnt >= m:
        break
    elif cnt <= m-n:
        print 'restarting', i
        T = restart(T,s,i)
    else:
        dead.append(i)
```

Observe that the counter is manipulated only by main server, locks are not needed.

2 Apr 2010

# The Mandelbrot Set

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

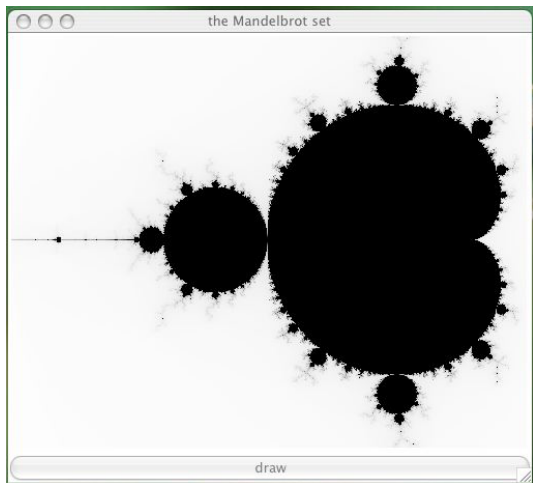
## Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations



# Definition of the Set

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

The Mandelbrot set is defined by an iterative algorithm:

- A point in the plane with coordinates  $(x, y)$  is represented by  $c = x + iy$ ,  $i = \sqrt{-1}$ .
- Starting at  $z = 0$ , count the number of iterations of the map  $z \rightarrow z^2 + c$  to reach  $|z| > 2$ .
- This number  $k$  of iterations determines the inverted grayscale  $255 - k$  of the pixel with coordinates  $(x, y)$  in the plot for  $x \in [-2, 0.5]$  and  $y \in [-1, +1]$ .

To display on a canvas of 400 pixels high and 500 pixels wide we need 16,652,580 iterations.

Pleasingly parallel: different threads on different pixels require no communication.

2 Apr 2010

# One Fifth of the Plot

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

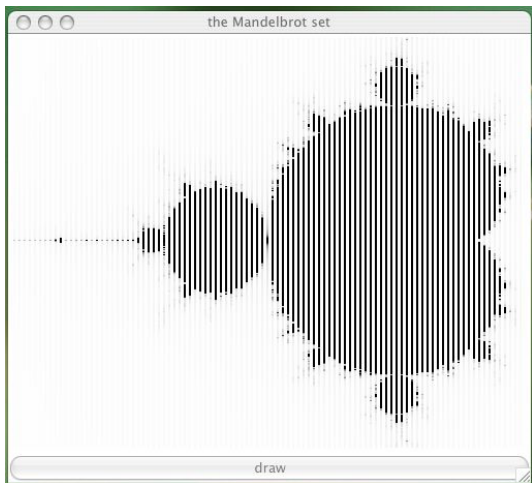
## Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations



2 Apr 2010

# Manager/Worker Paradigm

implemented via client/server parallel computations

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

The server *manages* a job queue, e.g.: number of columns on canvas to compute pixel grayscales of.

The  $n$  handler threads serve  $n$  clients. The clients perform the computational *work*, scheduled by the server.

Here we use a simple static workload assignment.

Handler thread  $t$  manages all columns  $k$ :  $k \bmod n = t$ .

E.g.:  $n = 2$ , first thread takes even columns, second thread takes odd columns.

A client receives the column number from the server and returns a list of grayscales for each row.

For code, see Project 4 solution of Spring 2008.

# Summary + Assignments

## Assignments:

- 1 Only threads can terminate themselves. Write a script that starts 3 threads. One thread prompts the user to continue or not, the other threads are busy waiting, checking the value of a shared boolean variable. When the user has given the right answer to terminate, the shared variable changes and all threads stop.
- 2 Write a multithreaded server with the capability to stop all running threads. All handler threads stop when one client passes the message “stop” to one server thread.
- 3 Use a multithreaded server to estimate  $\pi$  counting the number of samples  $(x, y) \in [0, 1] \times [0, 1]: x^2 + y^2 \leq 1$ . The job queue maintained by the server is a list of tuples. Each tuple contains the number of samples and the seed for the random number generator.

Server for  
Multiple  
Clients

avoid to block clients  
with waiting  
using sockets and  
threads

Waiting for  
Data from 3  
Clients

running a simple  
multithreaded server  
code for client and  
server

n Handler  
Threads for m  
Client  
Requests

managing threads  
dynamically  
code to manage  
handler threads

Pleasingly  
Parallel  
Computations

# Homework Collection

## Server for Multiple Clients

avoid to block clients with waiting  
using sockets and threads

## Waiting for Data from 3 Clients

running a simple multithreaded server  
code for client and server

## n Handler Threads for m Client Requests

managing threads dynamically  
code to manage handler threads

## Pleasingly Parallel Computations

Homework collection on Friday 9 April, at 1PM:

- 1 exercise 4 of lecture 21
- 2 exercise 1 of lecture 22
- 3 exercise 2 of lecture 23
- 4 exercise 1 of lecture 25
- 5 exercise 3 of lecture 26