Distributed Systems
Lecture 8

Server-side Sockets

Joseph Phillips

Last modified 2019 October 29

Reading

- Chapter 9 “*Sockets for Servers*” of Elliotte Rusty Harold “*Java Network Programming: 4th Ed.*”

Topics

- Server-side sockets in Java
- Server-side sockets in C

Motivation

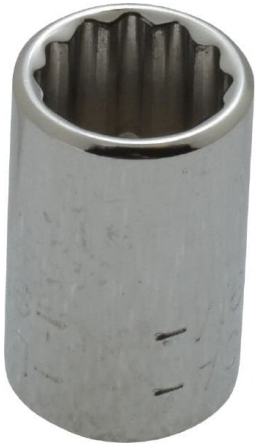
Last lecture we talked about implementing clients without the new-fangled Java tools of URLConnection, etc.

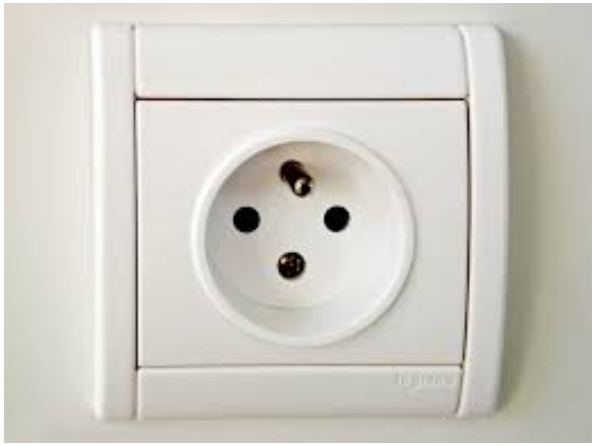
- Very nice
- But who is the client talking to?

Answer: "***The server!***"

Sockets!

- Remember . . .
- Duplex!
  - Can both read and write to same object
- Use ports
  - An integer from 1..65535
  - Acts like a “mailbox”





Java: the ServerSocket class

- Constructors:
  - public ServerSocket(int port) throws BindException, IOException
  - public ServerSocket(int port, int queueLength) throws BindException, IOException
  - public ServerSocket(int port, int queueLen, InetAddress bindAddr) throws IOException
  - public ServerSocket() throws IOException
- Accessors:
  - public Socket accept()
  - public void bind (SocketAddress endpoint) throws IOException
  - public void bind (SocketAddress endpoint, int queueLen) throws IOException

DaytimeServer.java

```
import java.net.*;
import java.io.*;
import java.util.Date;

public class DaytimeServer
{
  public final static int PORT  = 1024;

  public static void main (String[] args)
  {
    System.out.println("Connect to port " +
PORT);
    try (ServerSocket server = new
ServerSocket(PORT))
    {
      while  (true)
      {
        try (Socket connection = server.accept())
        {
          Writer out;
          out = new OutputStreamWriter
                (connection.getOutputStream());
          Date   now = new Date();
          out.write(now.toString() + "\r\n");
          out.flush();
          connection.close();
        }
        catch (IOException ex)
        { }
      }
    }
    catch (IOException ex)
    {
      System.err.println(ex);
    }
  }
}
```

But wait!

This server is single-threaded!
(Slower than it needs to be)

Two approaches for basic threads in Java:

- Extend class Thread, and `@Override public void run()`.
- Then make a new Thread of that class, and have it do `start()` method.

- Implement interface Runnable (which also needs `@Override public void run()`)
- Make a new Thread, giving instance of the new class as an argument to the constructor

MultithreadedDaytimeServer.java

```
import java.net.*;
import java.io.*;
import java.util.Date;

public class MultithreadedDaytimeServer
{
  public final static int PORT  = 1024;

  public static void main (String[] args)
  {
    System.out.println("Connect to port " + PORT);
    try (ServerSocket server = new ServerSocket(PORT))
    {
      while  (true)
      {
        try
        {
          Socket connection = server.accept();
          Thread task = new DaytimeThread(connection);
          task.start();
        }
        catch (IOException ex)
        { }
      }
    }
    catch (IOException ex)
    {
      System.err.println(ex);
    }
  }

  private static class DaytimeThread extends
Thread
  {
    private Socket connection;

    DaytimeThread(Socket connection)
    {
      this.connection = connection;
    }

    @Override
    public void run ()
    {
      try
      {
```

MultithreadedDaytimeServer.java

```
      Writer  out = new OutputStreamWriter(connection.getOutputStream());
      Date    now = new Date();
      out.write(now.toString() + "\r\n");
      out.flush();
    }
    catch (IOException ex)
    {
      System.err.println(ex);
    }
    finally
    {
      try
      {
        connection.close();
      }
      catch (IOException e)
      {
        // ignore
      }
    }
  }
}
```

Your turn!

From a security point of view:
what is the problem with spawning a new thread every time someone connect()s?

More advanced Threads in Java

- Make a class that implements Callable<WhateverClass>

A better idea: Thread Pools!

```
import java.net.*;
import java.io.*;
import java.util.*;
import java.util.concurrent.*;

public class PooledDaytimeServer
{
  public final static int PORT  = 1024;

  public static void main (String[] args)
  {
    System.out.println("Connect to port " + PORT);
    ExecutorService pool = Executors.newFixedThreadPool(16);

    try (ServerSocket server = new ServerSocket(PORT))
    {
      while  (true)
      {
        try
        {
          Socket         connection = server.accept();
          Callable<Void> task       = new DaytimeTask(connection);

          pool.submit(task);
        }
        catch (IOException ex)
        { }
      }
    }
    catch (IOException ex)
    {
      System.err.println(ex);
    }
  }

  private static class DaytimeTask implements Callable<Void>
  {
    private Socket connection;

    DaytimeTask(Socket connection)
    {
      this.connection = connection;
    }

    @Override
    public Void call ()
    {
      try
      {
```

A better idea: Thread Pools!

```
      Writer  out = new OutputStreamWriter(connection.getOutputStream());
      Date    now = new Date();
      out.write(now.toString() + "\r\n");
      out.flush();
    }
    catch (IOException ex)
    {
      System.err.println(ex);
    }
    finally
    {
      try
      {
        connection.close();
      }
      catch (IOException e)
      {
        // ignore
      }
    }
    return null;
  }
 }
}
```

Your turn!

Write an uppercasing server.

Client connects and gives line.

Server uppercases text
and sends back to client

More class Socket goodies

- public void setSoTimeout (int milliseconds) throws SocketException
- public int getSoTimeout () throws SocketException
  - How long to wait for read()
  - milliseconds == 0 means “Wait forever”
  - After time expires throws InterruptedIOException
    - Prepare to catch it!
    - Socket still open, next read() might succeed

More class Socket goodies

- public void setReuseAddress (boolean on) throws SocketException
- public boolean getReuseAddress () throws SocketException
  - Allow another socket to bind to same port immediately after close()?
  - false by default

More class Socket goodies

- public void setReceiveBufferSize (int size) throws SocketException, IllegalArgumentException
- public int getReceiveBufferSize () throws SocketException

More class Socket goodies

- public void setPerformancePreferences (int connectionTime, int latency, int bandwidth)
  - Sets relative preferences among connection-time, latency and bandwidth
  - ss.setPerformancePreferences(2,1,3)
    - Absolute values do no matter, only relative to each other
    - Example:
      - connection-time (=2) 2nd most important
      - latency (=1) least important
      - bandwidth (=3) most important
  - Implementation depends on particular Java VM

Server sockets in C

```
#include <sys/socket.h>//For socket()
#include <netinet/in.h>//For sockaddr_in and htons()
#include <netdb.h>     //For getaddrinfo()
#include <errno.h>     //For errno var
#include <sys/stat.h>  //For open(), read(),write()
#include <fcntl.h>     // and close()
```

- `socket()`: Ask OS for a socket
- `bind()`: Bind socket and port together
- `listen()`: Tell how many clients may queue
- `accept()`: Wait until a client connects
- `write()`: Write to client/server
- `read()`: Read from client/server
- `close()`: Close socket with client/server.

socket()

```
// Create a socket
int socketDescriptor =
          socket(AF_INET,     // AF_INET domain
                 SOCK_STREAM, // Reliable TCP
                 0);
```

- Returns
  - A file descriptor that the server uses to see if a client has connected, or,
  - `-1` on error
- There's also `SOCK_DGRAM` for UDP
- Last parameter type if used for `SOCK_RAW`

bind()

```
// Bind socket to port
// We'll fill in this datastruct
struct sockaddr_in socketInfo;

// Fill socketInfo with 0's
memset(&socketInfo,'\0',sizeof(socketInfo));
// Use std TCP/IP
socketInfo.sin_family = AF_INET;
// Tell port in network endian with htons()
socketInfo.sin_port = htons(portNumber);
  // (1) Allow connections from same machine only:
  struct in_addr addr;
  if (inet_aton("127.0.0.1",&addr)==0)  exit(EXIT_FAILURE);
  socketInfo.sin_addr.s_addr = addr.s_addr;
  // or (2) Allow machine to connect to this service
  socketInfo.sin_addr.s_addr = INADDR_ANY;
//  Try to bind socket with port and other specifications
int status = bind(socketDescriptor, // from socket()
                  (struct sockaddr*)&socketInfo,
                  sizeof(socketInfo));
status == -1 on error
```

bind()

```
// Bind socket to port
// We'll fill in this datastruct
struct sockaddr_in socketInfo;

// Fill socketInfo with 0's
memset(&socketInfo,'\0',sizeof(socketInfo));
// Use std TCP/IP
socketInfo.sin_family = AF_INET;
// Tell port in network endian with htons()
socketInfo.sin_port = htons(portNumber);
  // (1) Allow connections from same machine only:
  struct in_addr addr;
  if (inet_aton("127.0.0.1",&addr)==0)  exit(EXIT_FAILURE);
  socketInfo.sin_addr.s_addr = addr.s_addr;
  // or (2) Allow machine to connect to this service
  socketInfo.sin_addr.s_addr = INADDR_ANY;
//  Try to bind socket with port and other specifications
int status = bind(socketDescriptor, // from socket()
                  (struct sockaddr*)&socketInfo,
                  sizeof(socketInfo));
status == -1 on error
```

bind()

```
// Bind socket to port
// We'll fill in this datastruct
struct sockaddr_in socketInfo;

// Fill socketInfo with 0's
memset(&socketInfo,'\0',sizeof(socketInfo));
// Use std TCP/IP
socketInfo.sin_family = AF_INET;
// Tell port in network endian with htons()
socketInfo.sin_port = htons(portNumber);
  // (1) Allow connections from same machine only:
  struct in_addr addr;
  if (inet_aton("127.0.0.1",&addr)==0)  exit(EXIT_FAILURE);
  socketInfo.sin_addr.s_addr = addr.s_addr;
  // or (2) Allow machine to connect to this service
  socketInfo.sin_addr.s_addr = INADDR_ANY;
//  Try to bind socket with port and other specifications
int status = bind(socketDescriptor, // from socket()
                  (struct sockaddr*)&socketInfo,
                  sizeof(socketInfo));
status == -1 on error
```

bind()

```
// Bind socket to port
// We'll fill in this datastruct
struct sockaddr_in socketInfo;

// Fill socketInfo with 0's
memset(&socketInfo,'\0',sizeof(socketInfo));
// Use std TCP/IP
socketInfo.sin_family = AF_INET;
// Tell port in network endian with htons()
socketInfo.sin_port = htons(portNumber);
  // (1) Allow connections from same machine only:
  struct in_addr addr;
  if (inet_aton("127.0.0.1",&addr)==0)  exit(EXIT_FAILURE);
  socketInfo.sin_addr.s_addr = addr.s_addr;


//  Try to bind socket with port and other specifications
int status = bind(socketDescriptor, // from socket()
                  (struct sockaddr*)&socketInfo,
                  sizeof(socketInfo));
status == -1 on error
```

bind()

```
// Bind socket to port
// We'll fill in this datastruct
struct sockaddr_in socketInfo;

// Fill socketInfo with 0's
memset(&socketInfo,'\0',sizeof(socketInfo));
// Use std TCP/IP
socketInfo.sin_family = AF_INET;
// Tell port in network endian with htons()
socketInfo.sin_port = htons(portNumber);




  // or (2) Allow machine to connect to this service
  socketInfo.sin_addr.s_addr = INADDR_ANY;
//  Try to bind socket with port and other specifications
int status = bind(socketDescriptor, // from socket()
                  (struct sockaddr*)&socketInfo,
                  sizeof(socketInfo));
status == -1 on error
```

bind()

```
// Bind socket to port
// We'll fill in this datastruct
struct sockaddr_in socketInfo;

// Fill socketInfo with 0's
memset(&socketInfo,'\0',sizeof(socketInfo));
// Use std TCP/IP
socketInfo.sin_family = AF_INET;
// Tell port in network endian with htons()
socketInfo.sin_port = htons(portNumber);
  // (1) Allow connections from same machine only:
  struct in_addr addr;
  if (inet_aton("127.0.0.1",&addr)==0)  exit(EXIT_FAILURE);
  socketInfo.sin_addr.s_addr = addr.s_addr;
  // or (2) Allow machine to connect to this service
  socketInfo.sin_addr.s_addr = INADDR_ANY;
//  Try to bind socket with port and other specifications
int status = bind(socketDescriptor, // from socket()
                  (struct sockaddr*)&socketInfo,
                  sizeof(socketInfo));
status == -1 on error
```

What are those structs?

```
typedef uint32_t in_addr_t;

struct in_addr
{
  in_addr_t s_addr;
};

struct sockaddr_in
{
  sa_family_t    sin_family; // addr family: AF_INET
  in_port_t      sin_port;   // port (in network
                             //  byte order)
  struct in_addr sin_addr;   // internet addr
};
```

listen()

- (Should be called “setQueueSize()”)
- Tell OS how many clients may queue up while server busy

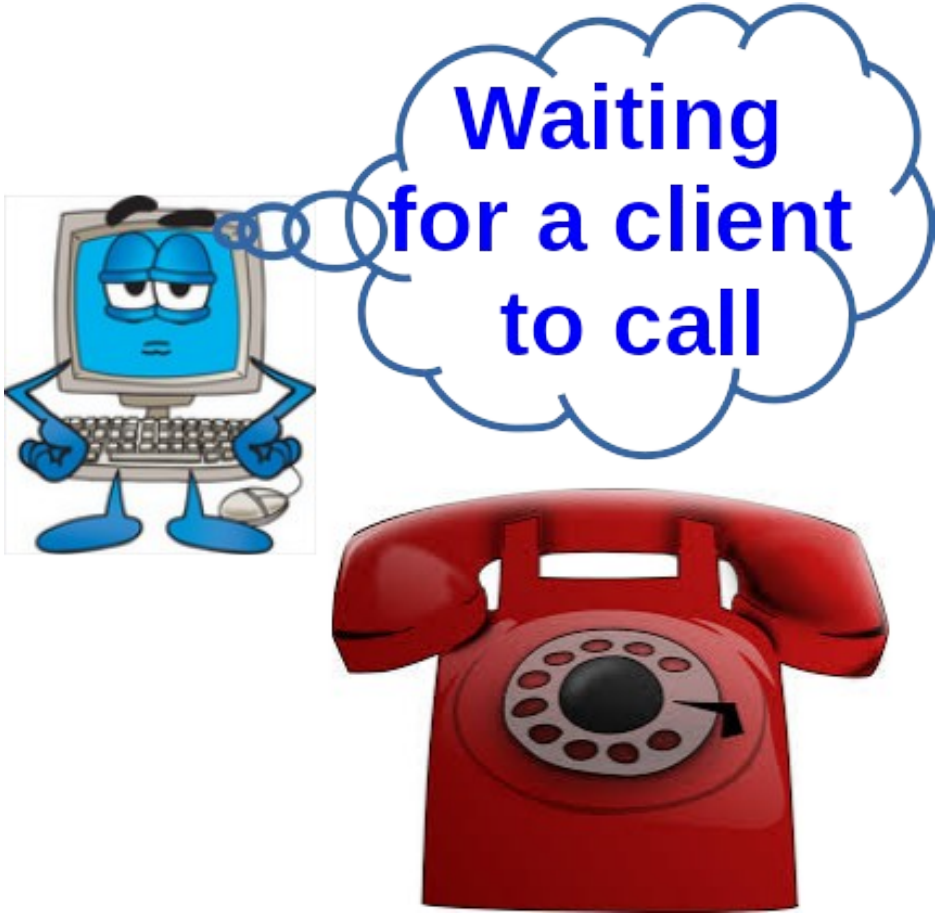
```
int status =
      listen(socketDescriptor,
             maxNumPendingClients);
```

- (Almost) ready to listen to port!
- `8` is a good default for `maxNumPendingClients`.
- If `status==-1` then error

Basic accept()

```
// Accept connection to client
int clientDescriptor =
  accept(socketDescriptor,NULL,NULL);
```

- Wait (by default) for someone to actual connect
- Returns
  - a file descriptor for talking with one particular client, or
  - `-1` for error
- `connectionDescriptor` for talking with that one client (there may be others for other clients)
- `socketDescriptor` is for listening to socket.



Intermediate accept()

Say! Just who are these clients?

```
struct sockaddr_in  clientAddr;
socklen_t  clientAddrLen = sizeof(clientAddr);//Init to clientAddr
  len
int clientFd = accept(listenFd,
  (sockaddr*)&clientAddr,&clientAddrLen);

if  (clientFd < 0)
{
  perror("accept()");
  exit(EXIT_FAILURE);
}

printf("Server: connect from host %s, port %d.\n",
       inet_ntoa(clientAddr.sin_addr),
       (int)ntohs(clientAddr.sin_port)
      );
```

Advanced accept()

- Do accept() only when necessary!
- Bust out of accept() after specified time!
  - Can have accept()-ing thread check other things
  - Like flag for "***should I still be running?***"
- **int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout);**
  - **readfds**: set of file descriptors to watch for input
  - **writefds**: set of file descriptors to watch to see if output will not block
  - **exceptfds**: set of file descriptors to watch for exceptions
  - **nfds**: highest number file descriptor in all 3 sets, plus 1.
  - **timeout**: timeout time (returns -1 and sets errno == ?)
- On exit, sets are modified to "these are the file descriptors to handle"
  - Therefore, have to set it back before next select()
- If set is NULL then, "Do not care about these file descriptors"

Advanced accept()

- void FD_ZERO(fd_set *set);
  - Clears (initializes) set of file descriptors
- void FD_SET(int fd, fd_set *set);
  - Sets ‘fd’ on in set
- void FD_CLR(int fd, fd_set *set);
  - Turn ‘fd’ off in set
- int FD_ISSET(int fd, fd_set *set);
  - Tells if ‘fd’ is on in given set
- struct timeval
  {
    long   tv_sec;   /* seconds */
    long   tv_usec;   /* microseconds */
  };

Advanced accept() Example usage

```
struct fd_set  originalSet;
struct fd_set  modifiedSet;
struct timeval time;  time.tv_sec = 1; time.tv_usec = 0;
int listenFd = getListenFd(port);

FD_ZERO(&originalSet);
FD_SET(listenFd,&originalSet);
while  (shouldRun)
{
  modifiedSet = originalSet;
  if  (select(listenFd+1,&modifiedSet,NULL,NULL,&time) < 0)
  {
    if  (errno == EINTR)
      //  No biggie, just SIGCHLD or SIGINT or something
    else
      //  Ruh-roh!  More serious
  }
  if  ( !FD_ISSET(listenFd,&modifiedSet) )
    continue; // Probably just timed out
  int clientFd = accept(listenFd, (sockaddr*)&clientAddr,&clientAddrLen);
  // Etc. etc.
}
```

Advanced accept()

```
//  From https://www.gnu.org/software/libc/manual/html_node/Server-Example.html
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#define PORT    5555
#define MAXMSG  512

int read_from_client (int filedes)
{
  char buffer[MAXMSG];
  int nbytes;
  nbytes = read (filedes, buffer, MAXMSG);
  if (nbytes < 0)
    {
      /* Read error. */
      perror ("read");
      exit (EXIT_FAILURE);
    }
  else if (nbytes == 0)
    /* End-of-file. */
    return -1;
  else
    {
      /* Data read. */
      fprintf (stderr, "Server: got message: `%s'\n", buffer);
      return 0;
    }
}
```

```
int  main (void)
{
  extern int make_socket (uint16_t port);
  int sock;
  fd_set active_fd_set, read_fd_set;
  int i;
  struct sockaddr_in clientname;
  size_t size;

  /* Create the socket and set it up to accept
connections. */
  sock = make_socket (PORT);
  if (listen (sock, 1) < 0)
    {
      perror ("listen");
      exit (EXIT_FAILURE);
    }

  /* Initialize the set of active sockets. */
  FD_ZERO (&active_fd_set);
  FD_SET (sock, &active_fd_set);
```

Advanced accept()

```
  while (1)
    {
      /* Block until input arrives on one or more active sockets.
*/
      read_fd_set = active_fd_set;
      if (select (FD_SETSIZE, &read_fd_set, NULL, NULL,
NULL) < 0)
        {
          perror ("select");
          exit (EXIT_FAILURE);
        }

      /* Service all the sockets with input pending. */
      for (i = 0; i < FD_SETSIZE; ++i)
        if (FD_ISSET (i, &read_fd_set))
          {
            if (i == sock)
              {
                /* Connection request on original socket. */
                int new;
                size = sizeof (clientname);
                new = accept (sock,
                              (struct sockaddr *) &clientname,
                              &size);
                if (new < 0)
                  {
                    perror ("accept");
                    exit (EXIT_FAILURE);
                  }
                fprintf
                  (stderr,
                   "Server: connect from host %s, port %hd.\n",
                   inet_ntoa (clientname.sin_addr),
                   (int)ntohs (clientname.sin_port));
                FD_SET (new, &active_fd_set);
              }
            else
              {
                // Data arriving on an already-connected socket.
                if (read_from_client (i) < 0)
                  {
                    close (i);
                    FD_CLR (i, &active_fd_set);
                  }
              }
          }
    }
```

More C socket goodies

- Just like Java, C lets you get and set socket-related parameters
  - int getsockopt(int sockfd, int level, int optname, void *optval, socklen_t *optlen);
  - int setsockopt(int sockfd, int level, int optname, const void *optval, socklen_t optlen);

And thread pools in C/C++? (1)

```
#include   <stdlib.h>
#include   <stdio.h>
#include   <string.h>
#include   <unistd.h>
#include   <pthread.h>

const int   NUM_THREADS = 16;

const int   BUFFER_SIZE    = 16;

static
bool shouldRun__    = true;

bool getShouldRun ()
{
  return(shouldRun__);
}


class    Buffer
{
  //  I.  Member vars:
  int  array_[BUFFER_SIZE];

  size_t inIndex_;

  size_t outIndex_;

  size_t count_;

  pthread_mutex_t       lock_;

  pthread_cond_t      notEmpty_;

  //  II.  Disallowed auto-generated methods:
  //  No copy constructor:
  Buffer         (const Buffer&);

  //  No copy assignment op:
  Buffer&   operator= (const Buffer&);

protected :
  //  III.  Protected methods:

public :
  //  IV.  Constructor(s), assignment op(s), factory(s) and destructor:
  Buffer         () :
inIndex_(0),
outIndex_(0),
count_(0)
{
  pthread_mutex_init(&lock_,NULL);
  pthread_cond_init(&notEmpty_,NULL);
}

  //  PURPOSE  To release the resources of '*this'.  No parameters.
  //No return value.
  ~Buffer           ()
{
  pthread_cond_destroy(&notEmpty_);
  pthread_mutex_destroy(&lock_);
}
```

And thread pools in C/C++? (2)

```
  //  V.  Accessors:

  //  VI.  Mutators:

  //  VII.  Methods that do main and misc. work of class:
  //  PURPOSE:  To insert file descriptor 'fd', address 'addr' and the
  //    length of the data in 'addr' noted in 'addrLen' in '*this',
  //    *ONLY* if there is space.  No return value.
  void    put       (int   value
)
{
  pthread_mutex_lock(&lock_);

  if  (count_ < BUFFER_SIZE)
  {
    array_[inIndex_]   = value;

    if  (++inIndex_ >= BUFFER_SIZE)
      inIndex_ = 0;

    count_++;
    pthread_cond_signal(&notEmpty_);
  }

  pthread_mutex_unlock(&lock_);
}
```

```
  //  PURPOSE:  To wait until data is available, and then to set 'value' equal
  //to the next available values stored in '*this'.  No return value.
  void    get   (int   id,
 int& value
)
{
  pthread_mutex_lock(&lock_);

  while  ( getShouldRun() &&
   (count_ == 0)
 )
  {
    pthread_cond_wait(&notEmpty_,&lock_);

    if  ( !getShouldRun() )
    {
      printf("Ending worker thread %d\n",id);
      pthread_mutex_unlock(&lock_);
      pthread_exit(NULL);
    }

  }

  value  = array_[outIndex_];

  if  (++outIndex_ >= BUFFER_SIZE)
    outIndex_ = 0;

  count_--;
  pthread_mutex_unlock(&lock_);
}
```

And thread pools in C/C++? (3)

```
  //  PURPOSE:  To notify all waiting threads to wake up.  No parameters.
  //    No return value.
  void    wakeWaiters    ()
        {
  pthread_mutex_lock(&lock_);
  pthread_cond_broadcast(&notEmpty_);
  pthread_mutex_unlock(&lock_);
}
};


Buffer   buffer;

void*    obtainerThreadFnc  (void*   vPtr)
{
  int  i;
  char   line[BUFFER_SIZE];

  while  (getShouldRun())
  {
    printf("Enter an integer, or to 0 quit: ");
    fflush(stdout);
    fgets(line,BUFFER_SIZE,stdin);
    i   = strtol(line,NULL,0);

    if  (i == 0)
      break;

    buffer.put(i);
  }

  shouldRun__ = false;
  printf("Ending obtaining thread.\n");
}
```

```
void*    workerThreadFnc    (void*   vPtr
)
{
  int  id = *(int*)vPtr;
  int  value;

  printf("Thread %d \"Ready for work!\"\n",id);
  fflush(stdout);

  while  (getShouldRun())
  {
    buffer.get(id,value);

    printf("Thread %d: \"2 * %d = %d\"\n",id,value,2*value);
    fflush(stdout);
  }

  return(NULL);
}


void initializeThreads  (pthread_t*  obtainerThreadPtr,
 pthread_t*   workerThreadArray
)
{
  int  idInfo[NUM_THREADS];

  for  (int i = 0;  i < NUM_THREADS;  i++)
  {
    idInfo[i]= i;
    pthread_create(workerThreadArray+i,NULL,workerThreadFnc,(void*)(idInfo+i));
  }

  pthread_create(obtainerThreadPtr,NULL,obtainerThreadFnc,NULL);
}
```

And thread pools in C/C++? (3)

```
int main   ()
{
  pthread_t   obtainerThread;
  pthread_t   workerThreadArray[NUM_THREADS];

  initializeThreads(&obtainerThread,workerThreadArray);

  while  ( getShouldRun() )
    sleep(1);

  shouldRun__ = false;
  buffer.wakeWaiters();

  for  (int index = 0;  index < NUM_THREADS;  index++)
    pthread_join(workerThreadArray[index],NULL);

  pthread_join(obtainerThread,NULL);

  return(EXIT_SUCCESS);
}
```

Your turn!

C would benefit from thread pools too.

Write a multi-threaded uppercasing server in C/C++ that:
(1) Uses a thread pool
(2) Records clients IPs
(3) Times-out after 1 sec/sets a global flag to tell when still running

References:

- Elliotte Rusty Harold "*Java Network Programming: 4th Ed.*"
- M. Tim Jones "*Gnu/Linux Application Programming, 2nd Ed.*" Charles River Media/Course Technology Cengage Learning.
- https://www.gnu.org/software/libc/manual/html_node/Server-Example.html