Distributed Systems
Lecture 1

Introduction and Review

Joseph Phillips

Last modified 2019 September 12

Topics

- Motivations
- Issues
- Low level I/O in Unix/C
- Low level I/O in Java
- Support for the class

Motivation

- Why distribute a system?
- Disadvantages:
  - Reliance on a network
  - Lack of a common clock
  - Inherently less stable
  - Inherently less secure

Advantage #1

- Ability to use non-local service
  - ssh into remote computer
  - ftp/sftp into remote computer
  - http into remote web-server

Advantage #2

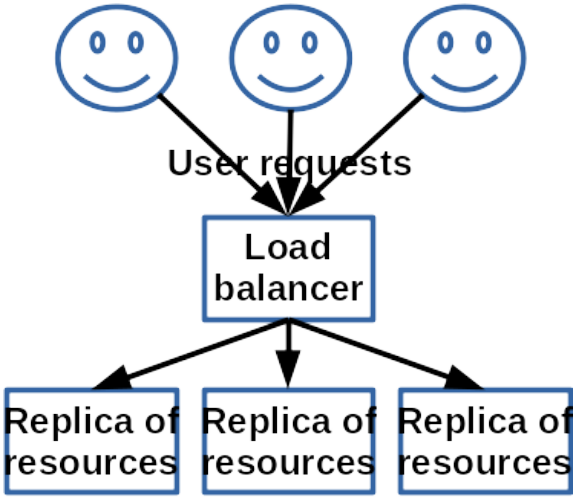
- Ability to scale solutions
  - Distributed computation
    - Notable example: SETI@home
    - Many others: https://en.wikipedia.org/wiki/List_of_distributed_computing_projects
  - Distributed responsibilities
    - Login server
    - Database server

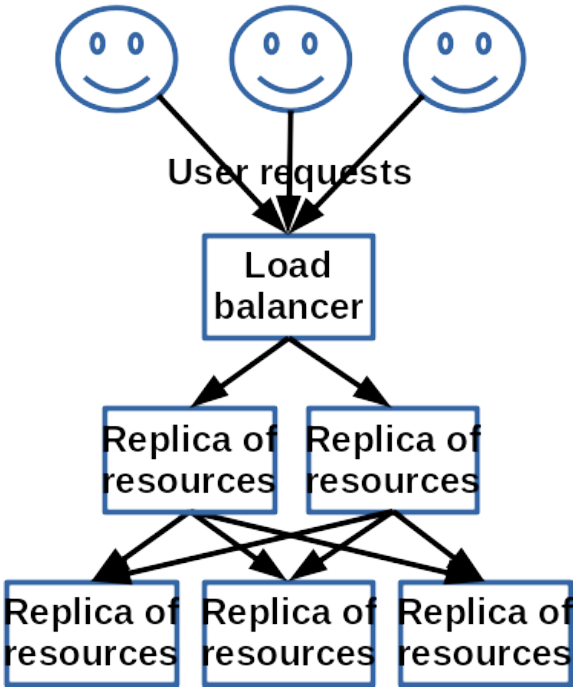
Advantage #3

- A different form of robustness
  - Use network to our advantage
  - If machine goes down, others still work

Advantage #4

- Pay for computing services only when they are needed
  - “Replicated load-balancing service” design pattern from Brendan Burns





Advantage #5

- Implementation language independence
  - Similar notion as “interfaces” in object-oriented programming
- You use your language, I’ll use mine
  - Your instructor is fond of C/C++, you may prefer Java or Python
- Some languages are better for some tasks
  - Fortran for matrix operations
  - Lisp for some A.I. applications
  - C for systems
  - Java or Python for getting something robust working quickly (programmer time)
- Support legacy systems / legacy languages
  - Cobol is still used

Issues: Protocols

- Internet Layer Protocols
  - IP: Internet Protocol
  - vs the earlier Network Control Protocol, a point-to-point protocol
- Transport Layer Protocols:
  - TCP: Transmission Control Protocol
  - Vs UDP (User Datagram Protocol)
- Application Layer Protocols:
  - SSH: Secure SHell
  - SNMP: Simple Network Management Protocol
- Above the application layer
  - XML
  - Json

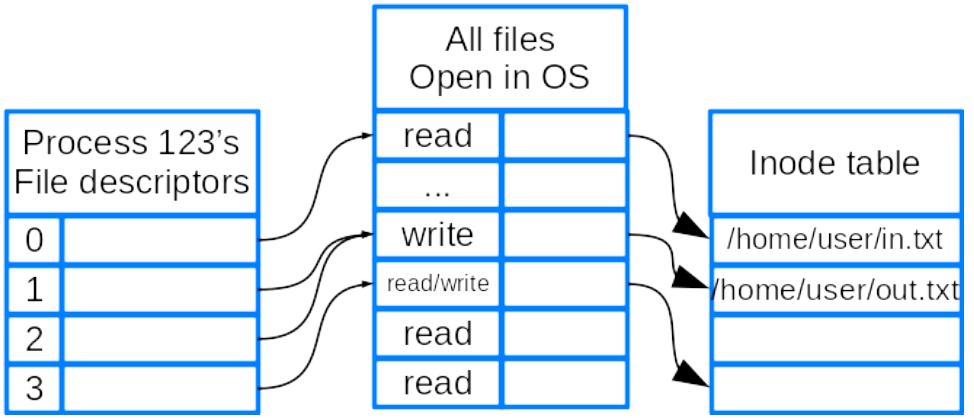
Issues: control and access

- May want to only allow
  - Certain users
  - From certain machines

Review: I/O in Unix/C

- File descriptors
  - Integers that refer to an index in an array of open files

Integer value	Symbolic constant	FILE* equivalent
0	STDIN_FILENO	stdin
1	STDOUT_FILENO	stdout
2	STDERR_FILENO	stderr



File descriptors for files

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open(const char *pathname, int flags);
```

- For read-only

```
int open(const char *pathname, int flags, mode_t mode);
```

- For read, write, read/write

```
int creat(const char *pathname, mode_t mode);
```

- For write-only

- Flags:
  - One of `O_RDONLY, O_WRONLY, O_RDWR`
  - Bitwise OR-ed with one or more of O_CREAT, O_TRUNC, O_APPEND (and some more)
- mode is permissions
- If return -1 then error

write()

```
int write(int fd, char* bufferPtr,
  size_t numBytes)
```

- Writes `numBytes` pointed to by `bufferSize` to `fd`.
- Returns number of bytes written, or `-1` on error.

read()

- `int read(int fd, char* bufferPtr, size_t bufferSize)`
  - Reads up to `bufferSize` bytes from `fd` and puts them into `bufferPtr`.
  - Returns number of bytes read from file, either
    - `0` (“*No more left!*”),
    - `bufferSize` (“*Here's a whole buffer full!*”),
    - somewhere inbetween (“*Here's all that's left*”), or,
    - `-1` (“*Error!*”)
- But, for network I/O, please use . . .

rio_read()

```
//  PURPOSE:  To be the "robust" version of
read()
//      in the manner advocated by Bryant and
//      O'Hallaron.  'fd' tells the file descriptor from
//      which to read.  'usrbuf' tells the buffer into
//      which to read.  'n' tells the length of 'usrbuf'.
//      Returns number of bytes read or -1 on
//      some type of error.
ssize_t rio_read  (int    fd,
                   char*  usrbuf,
                   size_t n
                  )
{
  //  I.  Application validity check:

  //  II.  Attempt to read:
  ssize_t       nread;
  size_t        nleft   = n;
  char*         bufp    = usrbuf;

  while (nleft > 0)
  {
    if  ((nread = read(fd, bufp, nleft)) < 0)
    {
      if (errno == EINTR)   // interrupt by sig handler?
        nread = 0;              // yes: read() again
      else
        return -1;              // no: errno set by read()
    }
    else
    if (nread == 0)
      break;                    // EOF

    nleft -= nread;
    bufp += nread;
  }

  //  III.  Finished:
  return (n - nleft);           // return >= 0
}
```

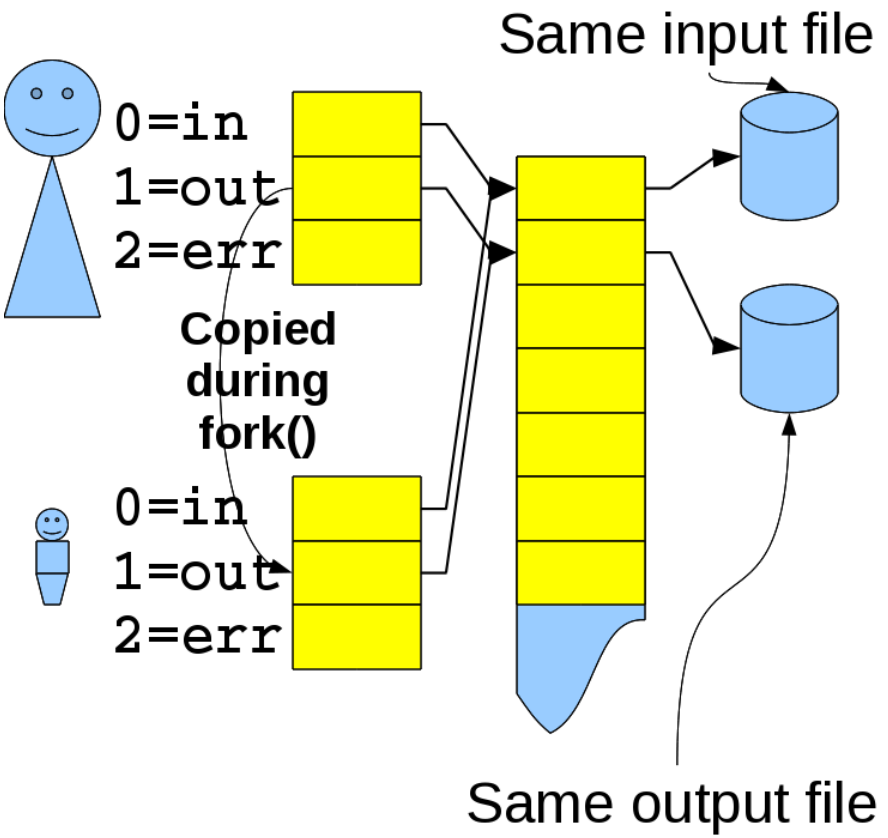
Your Turn!

Write a program to copy a file
given paths on the command line:

```
$ ourCopy sourceFile destFile
```

Remember:

- In Unix/C: `fork()` copies the file descriptor table
- A pipe or socket may still be open if only one process `close()`s it
- Therefore, immediately `close()` the file descriptor you won't use



Quick!
What's wrong with this program?

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

const int       BUFFER_LEN      = 256;

#define TEXT "Something to send down the
pipe."

int             main            ()
{
  int   fd[2];

  if  (pipe(fd) < 0)
    exit(EXIT_FAILURE);

  pid_t childId = fork();

  if  (childId < 0)
    exit(EXIT_FAILURE);

  if  (childId == 0)
  {
    char        buffer[BUFFER_LEN];
    int         numBytes;

    if  (read (fd[0],buffer,BUFFER_LEN) > 0)
    {
      printf("Child received \"%s\"\n",buffer);
    }

    close(fd[0]);
  }
  else
  {
    printf("Parent sending \"%s\"\n",TEXT);
    write(fd[1],TEXT,sizeof(TEXT));
    close(fd[1]);
    wait(NULL);
  }

  return(EXIT_SUCCESS);
}
```

Java streams

- Input and Output
- Often filters are chained together
- Filters for
  - buffering (efficiency)
  - compression/uncompression
  - encrypting/decrypting

Java output streams

- From java.io.OutputStream
  - public abstract class OutputStream
  - public abstract void write (int b) throws IOException
    - Writes lowest byte of integer ‘b’
  - public void write (byte[ ] data) throws IOException
  - public void write (byte[ ] data, int offset, int length) throws IOException
    - Write array (or subarray) of bytes
  - public void flush () throws IOException
  - public void close () throws IOException

OutputStream Example

```
public static void generateChars      (OutputStream out)
    throws IOException
 {
    int firstPrintChar  = 33;
    int numPrintChars   = 94;
    int numCharsPerLine = 72;
    int start           = firstPrintChar;

    while  (true)
    {
      for  (int i = start;  i < start + numCharsPerLine;  i++)
        out.write((i-firstPrintChar) % numPrintChars + firstPrintChar);
      out.write('\r');
      out.write('\n');
      start = ((start+1) - firstPrintChar) % numPrintChars + firstPrintChar;
    }
 }
```

But wait! Is that efficient?

- Sending bytes one-at-a-time is inefficient
  - Smallest TCP header 40 bytes.
  - Sending 1 application byte really means sending at least 40 more
  - Thus sending 100 bytes one-at-a-time could mean sending 4100 bytes total
- Better to buffer bytes into an array . . .

Improved OutputStream Example

```
//      From Elliote Rusty Harold
//      "Java Network Programming"
//      O'Reilly, 2014
import java.io.OutputStream;
import java.io.IOException;

public class GenChars
{

  public static void generateChars      (OutputStream out)
    throws IOException
  {
    int firstPrintChar  = 33;
    int numPrintChars   = 94;
    int numCharsPerLine = 72;
    int start           = firstPrintChar;

    byte[] line         = new byte[numCharsPerLine + 2];

    while  (true)
    {
      for  (int i = start;  i < start + numCharsPerLine;  i++)
        line[i-start] = (byte)
                        ((i-firstPrintChar) % numPrintChars +
firstPrintChar);
      line[numCharsPerLine+0]   = (byte)'\r';
      line[numCharsPerLine+1]   = (byte)'\n';
      out.write(line);
      start = ((start+1) - firstPrintChar) %
numPrintChars + firstPrintChar;
    }
  }

  public static void main(String[] args)
  {
    try
    {
      generateChars(System.out);
    }
    catch  (IOException except)
    {
    }
  }

}
```

Is buffering the end of our troubles?

- Not quite!
- `flush()` the output whenever you want to send it immediately
- Be sure to close() streams at end, even upon exceptions
  - Otherwise could memory leak file descriptors

```
OutputStream out = null;
try
{
  out = new FileOutputStream("out.txt");
  // work with out
}
catch (IOException ex)
{
   System.err.println(ex.getMessage());
}
finally
{
    if  (out != null)
    {
      try
      {
        out.close();
      }
      catch (IOException ex)
      {
        // Ignore
      }
    }
    }
```

InputStream

- public abstract class InputStream
- public abstract int read () throws IOException
  - Reads single byte
  - -1 means “no more”
  - Blocks (waits) until byte is available
- public int read (byte[ ] input) throws IOException
- public int read (byte[ ] input, int offset, int length) throws IOException
  -
- public long skip (long n) throws IOException
- public int available () throws IOException
  - Tells how many bytes can be read without blocking
- public int void close() throws IOException

To naively read one byte a time

```
byte [ ] input = new byte[10];
for (int i = 0; i < input.length;  i++)
{
  int b = in.read();
  if  (b < 0) break;
  input[i] = (byte)b;
}
```

More efficient

```
int bytesRead = 0;
int bytesToRead = 64;
byte [ ] input = new byte[bytesToRead];
while  (bytesRead < bytesToRead)
{
  int result = in.read(input,
bytesRead,bytesToRead - bytesRead);
  if  (result < 0) break;
  bytesRead += result;
}
```

File Streams

- FileInputStream:
  - FileInputStream(File file)
  - FileInputStream(FileDescriptor fdObj)
  - FileInputStream(String name)
- FileOutputStream:
  - FileOutputStream(File file)
  - FileOutputStream(File file, boolean append)
  - FileOutputStream(FileDescriptor fdObj)
  - FileOutputStream(String name)
  - FileOutputStream(String name, boolean append)

Your Turn!

Write a program to copy a file
given paths on the command line:

```
$ java CopyFile CopyFile.java CopyFileCopy.java
```

Your Turn! (continued)

```
import  java.io.InputStream;
import  java.io.FileInputStream;
import  java.io.OutputStream;
import  java.io.FileOutputStream;
import  java.io.IOException;

public class CopyFile
{
  public static void main (String[] args)
  {

    if  (args.length < 2)
    {
      System.err.println("Usage:\tjava CopyFile
<sourceFile> <destFile>");
      return;
    }

    copy(args[0],args[1]);
  }
```

```
  public static void    copy
    (String sourcePath,
     String destPath
    )
  {
    final int  MAX_BUFFER_LEN  = 256;
    FileOutputStream    out          = null;
    FileInputStream     in           = null;
    byte[ ] byteArray   = new
byte[MAX_BUFFER_LEN];

    //  YOUR CODE HERE
  }
}
```

FilterStreams and Chaining

- FilterStreams come in two types:
  - input
  - output
- They do processing on their streams
  - output stream compresses/input stream uncompresses
  - output stream encrypts/input stream decrypts
- They are meant to be chained together, e.g.

```
InputStream in = new FileInputStream("data.txt");
in = new BufferedInputStream(in);

// or

InputStream in = new BufferedInputStream(new FileInputStream("data.txt"));
```

- Buffered streams are a special case:
  - public BufferInputStream(InputStream in)
  - public BufferInputStream(InputStream in, int bufferSize);
  - public BufferOutputStream(OutputStream in)
  - public BufferOutputStream(OutputStream in, int bufferSize);

PrintStream

- Please do NOT use PrintStream over the network!
  - How say “next line”?
    - Unix: '\n'
    - Windows: '\r' '\n'
    - MAC: '\r'
    - println() on local system will be fine, but maybe not on remote system
  - Similarly, char encoding on local system may be different than remote (e.g. UTF-8, UTF-16, CP1252 (US Windows), SJIS (Japan), etc.)
  - Ignores exceptions, but these might be important in network communication

DataOutputStream

public final void writeBoolean (boolean b) throws IOException
public final void writeByte (int b) throws IOException
public final void writeShort (int s) throws IOException
public final void writeChar (int c) throws IOException
public final void writeInt (int i) throws IOException
public final void writeLong (long l) throws IOException
public final void writeFloat (float f) throws IOException
public final void writeDouble (double d) throws IOException
public final void writeChars (String s) throws IOException
public final void writeBytes (String s) throws IOException
public final void writeUTF (String s) throws IOException

DataInputStream

public final boolean readBoolean () IOException
public final byte readByte () IOException
public final char readChar () IOException
public final short readShort () IOException
public final int readInt () IOException
public final long readLong () IOException
public final float readFloat () IOException
public final double readDouble () IOException
public final String readUTF () IOException

DataStream

- Meant to transfer data to another system that understands Java datatypes
  - Another Java program
  - A program (perhaps C) that uses Java types
- Java types
  - Big (standard network) endian
  - 2's complement integers
    - boolean, byte = 1 byte
    - short = 2 bytes
    - int, float = 4 bytes
    - long, double = 8 bytes
    - char = 2 unsigned bytes
  - writeChars(): UTF-16
  - writeBytes(): low byte only
  - writeUTF(): Java's ***variant*** of UTF-8 (only use with other Java programs)

Readers and Writers

- Streams are for reading/writing ***buffers*** and ***data***
  - Analogous to ***file descriptors*** in C
- Readers and Writers specialize in ***text***
  - Analogous to ***FILE**** in C
- By default reads/writes unsigned 16-bit chars (UTF-16)

Writers

- protected Writer()
- protected Writer (Object lock)
- public abstract void (char[ ] text, int offset, int length) throws IOException
- public void write (int c) throws IOException
- public void write (char[ ] text) throws IOException
- public void write (String s) throws IOException
- public void write (String s, int offset, int length) throws IOException
- public abstract void flush () throws IOException
- public abstract void close () throws IOException

Readers

- protected Reader ()
- protected Reader (Object lock)
- public abstract int read (char[ ] text, int offset, int length)
- public int read () throws IOException
- public int read (char[ ] text) throws IOException
- public long skip (long n) throws IOException
- public boolean ready ()
- public boolean markSupport()
- public void mark (int readAheadLimit) throws IOException
- public void reset () throws IOException
- public abstract void close() throws IOException

Getting a Unix System

- Do you already have your own Linux, Mac OS or Unix machine? Awesome! Use it.
- See Prof Joe after class to get an account on the virtual machine: phillips373.cdm.depaul.edu.

References

- M. Tim Jones “*GNU/Linux Application Programming, 2nd Ed*” Course Technology, Cengage Learning. 2008
- Elliote Rusty Harold “*Java Network Programming, 4th Ed”* O'Reilly. 2014
- Brendan Burns “*Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services*” O'Reilly. 2018