TCP/IP Intro

TCP/IP beyond scope of this course - take COMP[39]331.
But easier to understand CGI using TCP/IP from Perl
Easy to establish a TCP/IP connection.
Server running on host williams.cse.unsw.edu.au does this:

```
use IO::Socket;
$server = IO::Socket::INET->new(LocalPort => 1234,
                Listen => SOMAXCONN) or die;
$c = $server->accept()
```

Client running anywhere on internet does this:

```
use IO::Socket;
$host = "williams.cse.unsw.edu.au";
$c = IO::Socket::INET->new(PeerAddr=>$host,
                           PeerPort=>1234) or die;
```

Then $c effectively a bidirectional file handle.

Time Server

A simple TCP/IP server which supplies the current time as an ASCII string.

```
use IO::Socket;
$server = IO::Socket::INET->new(LocalPort => 4242,
                Listen => SOMAXCONN) or die;
while ($c = $server->accept()) {
    printf "[Connection from %s]\n", $c->peerhost;
    print $c scalar localtime,"\n";
    close $c;
}
```

Source: http://cgi.cse.unsw.edu.au/~cs2041/18s2/code/webserver/timeserver.pl

Time Client

Simple client which gets the time from the server on host `$ARGV[0]` and prints it.
See NTP for how to seriously distribute time across networks.

```
use IO::Socket;
$server_host =  $ARGV[0] || 'localhost';
$server_port = 4242;
$c = IO::Socket::INET->new(PeerAddr => $server_host,
                           PeerPort  => $server_port) or die;
$time = <$c>;
close $c;
print "Time is $time\n";
```

Source: http://cgi.cse.unsw.edu.au/~cs2041/18s2/code/webserver/timeclient.pl

Well-known TCP/IP ports

To connect via TCP/IP you need to know the port. Particular services often listen to a standard TCP/IP port on the machine they are running. For example:

- 21 ftp
- 22 ssh (Secure shell)
- 23 telnet
- 25 SMTP (e-mail)
- 80 HTTP (Hypertext Transfer Protocol)
- 123 NTP (Network Time Protocol)
- 443 HTTPS (Hypertext Transfer Protocol over SSL/TLS)

So web server normally listens to port 80 (http) or 443 (https).

Uniform Resource Locator (URL)

Familiar syntax:

```
scheme://domain:port/path?query_string#fragment_id
```

For example:

```
http://en.wikipedia.org/wiki/URI_scheme#Generic_syntax
http://www.google.com.au/search?q=COMP2041&hl=en
```

Given a http URL a web browser extracts the hostname from the URL and connects to port 80 (unless another port is specified).
It then sends the remainder of the URL to the server.
The HTTP syntax of such a request is simple:

`GET` *path* `HTTP/`*version*

We can do this easily in Perl

Simple Web Client in Perl

A very simple web client - doesn't render the HTML, no GUI, no ... - see HTTP::Request::Common for a more general solution

```
use IO::Socket;
foreach $url (@ARGV) {
    $url =~ /http:\/\/([^\/]+)(:(\d+))?(.*)/ or die;
    $c = IO::Socket::INET->new(PeerAddr => $1,
          PeerPort => $2 || 80) or die;
    # send request for web page to server
    print $c "GET $4 HTTP/1.0\n\n";
    # read what the server returns
    my @webpage = <$c>;
    close $c;
    print "GET $url =>\n", @webpage, "\n";
}
```

Source: http://cgi.cse.unsw.edu.au/~cs2041/18s2/code/webserver/webget.pl

Simple Web Client in Perl

```
$ cd /home/cs2041/public_html/lec/cgi/examples
$ ./webget.pl http://cgi.cse.unsw.edu.au/
GET http://cgi.cse.unsw.edu.au/ =>
HTTP/1.1 200 OK
Date: Sun, 21 Sep 2014 23:40:41 GMT
Set-Cookie: JSESSIONID=CF09BE9CADA20036D93F39B04329DB
Last-Modified: Sun, 21 Sep 2014 23:40:41 GMT
Content-Type: text/html;charset=UTF-8
Content-Length: 35811
Connection: close

<!DOCTYPE html>
<html lang='en'>
  <head>
...
```

Notice the web server returns some header lines and then data.

Web server in Perl - getting started

This Perl web server just prints details of incoming requests & always returns a 404 (not found) status.

```
use IO::Socket;
$server = IO::Socket::INET->new(LocalPort => 2041,
        ReuseAddr => 1, Listen => SOMAXCONN) or die;
while ($c = $server->accept()) {
   printf "HTTP request from %s =>\n\n", $c->peerhost;
   while ($request_line = <$c>) {
      print "$request_line";
      last if $request_line !~ /\S/;
   }
   print $c "HTTP/1.0 404 This server always 404s\n";
   close $c;
}
```

http://cgi.cse.unsw.edu.au/ cs2041/18s2/code/webserver/webserver-404.pl

Web server in Perl - getting started

```
use IO::Socket;
$server = IO::Socket::INET->new(LocalPort => 2041,
        ReuseAddr => 1, Listen => SOMAXCONN) or die;
$content = "Everything is OK - you love COMP(2041|9044).\n";
while ($c = $server->accept()) {
   printf "HTTP request from %s =>\n\n", $c->peerhost;
   while ($request_line = <$c>) {
      print "$request_line";
      last if $request_line !~ /\S/;
   }
   my $request = <$c>;
   print "Connection from ", $c->peerhost, ": $request";
   $request =~ /^GET (.+) HTTP\/1.[01]\s*$/;
   print "Sending back /home/cs2041/public_html/$1\n";
   open my $f, '<', "/home/cs2041/public_html/$1";
   $content = join "", <$f>;
   print $c "HTTP/1.0 200 OK\nContent-Type: text/html\n";
   print $c "Content-Length: ",length($content),"\n";
   print $c $content;
   close $c;
}
```

Web server in Perl - too simple

A simple web server in Perl.
Does fundamental job of serving web pages but has bugs, securtity holes and huge limitations.

```
while ($c = $server->accept()) {
    my $request = <$c>;
    $request =~ /^GET (.+) HTTP\/1.[01]\s*$/;
    open F, "</home/cs2041/public_html/$1";
    $content = join "", <F>;
    print $c "HTTP/1.0 200 OK\n";
    print $c "Content-Type: text/html\n";
    print $c "Content-Length: ",length($content),"\n";
    print $c $content;
    close $c;
}
```

Source: http://cgi.cse.unsw.edu.au/~cs2041/18s2/code/webserver/webserver-too-simple.pl

Web server in Perl - mime-types

Web servers typically determine a file's type from its extension (suffix) and pass this back in a header line.
ON Unix-like systems file /etc/mime.types contains lines mapping extensions to mime-types, e.g.:

```
application/pdf                 pdf
image/jpeg                      jpeg jpg jpe
text/html                       html htm shtml
```

May also be configured within web-server e.g cs2041's .htaccess file contains:

```
AddType text/plain pl py sh c cgi
```

Web server in Perl - mime-types

Easy to read /etc/mime.types specifications into a hash:

```
open MT, '<', "/etc/mime.types") or die;
while ($line = <MT>) {
    $line =~ s/#.*//;
    my ($mime_type, @extensions) = split /\s+/, $line;
    foreach $extension (@extensions) {
     $mime_type{$extension} = $mime_type;
    }
}
```

Web server in Perl - mime-types

Previous simple web server with code added to use the `mime_type` hash to return the appropriate `Content-type`:

```
$url =~ s/(^|\/)\.\.(\/|$)//g;
my $file = "/home/cs2041/public_html/$url";
# prevent access outside 2041 directory
$file =~ s/(^|\/)..(\/|$)//g;
$file .= "/index.html" if -d $file;
if (open my $f, '<', $file) {
    my ($extension) = $file =~ /\.(\w+)$/;
    print $c "HTTP/1.0 200 OK\n";
    if ($extension && $mime_type{$extension}) {
        print $c "Content-Type: $mime_type{$extension}\n";
    }
    print $c <my $f>;
}
```

Source: http://cgi.cse.unsw.edu.au/~cs2041/18s2/code/webserver/webserver-mime-types.pl

Web server in Perl - multi-processing

Previous web server scripts serve only one request at a time.
They can not handle a high volume of requests.
And slow client can deny access for others to the web server, e.g our previous web client with a 1 hour sleep added:

```
$url =~ /http:\/\/([^\/]+)(:(\d+))?(.*)/ or die;
$c = IO::Socket::INET->new(PeerAddr => $1,
      PeerPort => $2 || 80) or die;
sleep 3600;
print $c "GET $4 HTTP/1.0\n\n";
```

Source: http://cgi.cse.unsw.edu.au/~cs2041/18s2/code/webserver/webget-slow.pl

Simple solution is to process each request in a separate process.
The Perl subroutine fork duplicates a running program.
Returns 0 in new process (child) and process id of child in original process (parent).

Web server in Perl - multi-processing

We can add this easily to our previous webserver:

```
while ($c = $server->accept()) {
    if (fork() != 0) {
        # parent process loops to wait for next request
        close($c);
        next;
    }
    # child processes request
    my $request = <$c>;
    ...
    close $c;
    # child must terminate here otherwise
    # it would compete with parent for requests
    exit 0;
}
```

Source: http://cgi.cse.unsw.edu.au/~cs2041/18s2/code/webserver/webserver-parallel.pl

Web server - Simple CGI

Web servers allow dynamic content to be generated via CGI (and other ways).
Typically they can be configure to execute programs for certain URIS.
for example cs2041's .htaccess file indicates files with suffix `.cgi` should be executed.

```
<Files *.cgi>
SetHandler application/x-setuid-cgi
</Files>
```

Web server - Simple CGI

We can add this to our simple web-server:

```
if ($url =~ /^(.*\.cgi)(\?(.*))?$/) {
    my $cgi_script = "/home/cs2041/public_html/$1";
    $ENV{SCRIPT_URI} = $1;
    $ENV{QUERY_STRING} = $3 || '';
    $ENV{REQUEST_METHOD} = "GET";
    $ENV{REQUEST_URI} = $url;
    print $c "HTTP/1.0 200 OK\n";
    print $c `$cgi_script` if -x $cgi_script;
    close F;
}
```

Source: http://cgi.cse.unsw.edu.au/~cs2041/18s2/code/webserver/webserver-simple-cgi.pl

Web server - CGI

A fuller CGI implementation implementing both GET and POST requests can be found here:

Source: http://cgi.cse.unsw.edu.au/~cs2041/18s2/code/webserver/webserver-cgi.pl