Client-server architectures

- Have *server* process, which waits for client requests and processes them once they arrive
- Multiple clients may connect and request service
- Standard paradigm for services on the internet
- Have in addition to IP-addresses *port numbers* assigned to particular services
- Examples:
- Port 80 for http
- Port 25 for sending mail (smtp)
- Port 143 for reading mail via imap
- Port assignments listed in `/etc/services`
- Client-Server architecture implemented via *sockets*

Sockets

How to setup a client/server connection via sockets:
Initialisation phase for the server:

- Server creates endpoint via `socket`-system call
- Server specifies port number and protocol in structure `sockaddr_in`
- Server assigns information in `sockaddr_in`-structure to socket via `bind`-system call

Now server waits for incoming connection via `accept`-system call
When connection received, server reads data via `read`-system call and writes data back via `write`-system call
When server is finished with current connection, server closes connection via `close`-system call

Concurrency

- Good handling of concurrency vital for implementing sockets
- Will use `pthread`-library for this
  Library implements kernel-level threads together with synchronisation mechanisms
- Key point: Program may create arbitrary number of threads, which share memory and may run concurrently
- Synchronisation achieved by mutual exclusion:
  A section of code satisfies mutual exclusion if it can executed by only one thread at a time, and in addition no switching between threads happens while this section of code is executed.
- Such a section of code is called critical section

Important operations on threads

- `pthread_create` creates new threads
- `pthread_exit` exits threads
- `pthread_mutex_lock`(mutex) starts a critical section involving those threads using the specified mutex
- `pthread_mutex_unlock`(mutex) ends a critical section involving those threads using the specified mutex
- `pthread_attr_init` sets up default attributes
- `pthread_join` wait for termination for another thread
- `pthread_attr_setdetachstate` cause thread to terminate immediately when `pthread_exit` is called
  $\Rightarrow$ cannot use `pthread_join`