


**Week 12 - Tuesday**

---

**Network Programming  
(cont.)**

---


School of Information Technology and Electrical Engineering  
The University of Queensland



**Week 12**

- Tuesday
  - Network Programming (cont.)
- Friday
  - Ethernet
- Assignment 4
  - Due Friday 5 June @ 11pm (plus any available grace days)
  - No submissions after 11pm Wednesday 10 June


3



**Outline**

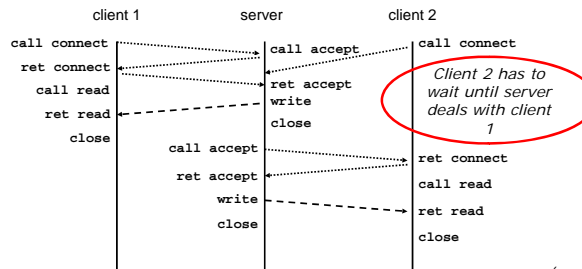
- C Network Programming
  - Concurrent servers – single process (`select()` function)
  - TEVAL
  - UDP Example
  - Concurrent servers – multi process
- Credits:
  - Glass and Ables, "UNIX for Programmers and Users"
  - Bryant and O'Halloran, "Computer Systems: A Programmer's Perspective"
  - Rochkind, "Advanced UNIX Programming"
  - Tanenbaum, "Computer Networks"

5




**Iterative Servers**

- Iterative servers process one request at a time




6



**3 Basic Mechanisms for Creating Concurrent Flows**

1. Threads
  - Kernel automatically interleaves multiple logical flows
  - Each flow shares the same address space
  - Hybrid of processes and I/O multiplexing!
2. Processes
  - Kernel automatically interleaves multiple logical flows
  - Each flow has its own private address space
3. I/O multiplexing with `select()`
  - User manually interleaves multiple logical flows
  - Each flow shares the same address space
  - Popular for high-performance server designs

7



**Event-Based Concurrent Servers Using I/O Multiplexing**

- Maintain a set of connected descriptors and service each as new data arrives
- Repeat the following forever:
  - Use the Unix `select()` function to block until:
    - (a) New connection request arrives on the listening descriptor, or
    - (b) New data arrives on an existing connected descriptor
  - If (a), add the new connection to the pool of connections
  - If (b), read any available data from the connection
    - Close connection on EOF and remove it from the set

8

COMP2303  
COMP7306

## The select() Function

- select() sleeps until one or more file descriptors in the set `readset` are ready for reading

```
#include <sys/select.h>
int select(int maxfdp1, fd_set *readset, NULL, NULL, NULL);
```

- `readset`
  - Opaque bit vector (max `FD_SETSIZE` bits) that indicates membership in a *descriptor set*
  - If bit `k` is 1, descriptor `k` is a member of the descriptor set
- `maxfdp1`
  - Maximum descriptor in descriptor set plus 1
  - Tests descriptors 0, 1, 2, ..., `maxfdp1 - 1` for set membership
- select() returns number of ready descriptors and sets each bit of `readset` to indicate the ready status of corresponding descriptor

9

COMP2303  
COMP7306

## Macros for Manipulating Set Descriptors

- `void FD_ZERO(fd_set *fdset);`
  - Turn off all bits in `fdset`
- `void FD_SET(int fd, fd_set *fdset);`
  - Turn on bit `fd` in `fdset`
- `void FD_CLR(int fd, fd_set *fdset);`
  - Turn off bit `fd` in `fdset`
- `int FD_ISSET(int fd, *fdset);`
  - Is bit `fd` in `fdset` turned on?

11

COMP2303  
COMP7306

## Sample select() Server Code

- *To be discussed in class*

13

COMP2303  
COMP7306

## Break

15

COMP2303  
COMP7306

## UDP Example Code

- *To be discussed in class*

16

COMP2303  
COMP7306

## Process-Based Concurrent Server

```
...
/* main server loop */
while (1) {
    connfd = accept(listenfd, (struct sockaddr *) &clientaddr,
                    &clientlen);

    if (fork() == 0) {
        close(listenfd); /* child closes its listening socket */
        echo(connfd); /* child reads and echoes input line */
        close(connfd); /* child is done with this client */
        exit(0); /* child exits */
    }
    close(connfd); /* parent must close connected socket! */
}
...

```

18

COMP2303  
COMP7306

## Process-Based Concurrent Server (cont.)

```
...
signal(SIGCHLD, handler); /* parent must reap children! */
...

/* handler - reaps children as they terminate */
void handler(int sig) {
    pid_t pid;
    int stat;

    while ((pid = waitpid(-1, &stat, WNOHANG)) > 0)
    {
        ;
    }
    return;
}
```

19

COMP2303  
COMP7306

## Implementation Issues With Process-Based Designs

- Server should restart `accept` call if it is interrupted by a transfer of control to the `SIGCHLD` handler
  - Not necessary for systems with POSIX signal handling.
  - Required for portability on some older Unix systems.
- Server must reap zombie children
  - to avoid fatal memory leak
- Server must `close` its copy of `connfd`
  - Kernel keeps reference for each socket
  - After fork, `refcnt(connfd) = 2`
  - Connection will not be closed until `refcnt(connfd)=0`

21

COMP2303  
COMP7306

## Pros and Cons of Process-Based Designs

- + Handles multiple connections concurrently
- + Simple and straightforward
- Additional overhead for process control
- Nontrivial to share data between processes
  - Requires IPC (interprocess communication) mechanisms
    - FIFO's (named pipes), System V shared memory and semaphores
- *I/O multiplexing provides more control with less overhead...*

22

COMP2303  
COMP7306

## Pro and Cons of Event-Based Designs

- + One logical control flow
- + Can single-step with a debugger
- + No process or thread control overhead
- More complex to code than process or thread-based designs
- Can be vulnerable to denial of service attack
  - How?

23

COMP2303  
COMP7306

## Resources

- Beej's Guide to Network Programming
  - <http://beej.us/guide/bgnet/>
- UNIX manual pages
  - On Solaris: `man -s 3socket <name>`
  - where `<name>` is `socket`, `bind`, `connect`, `listen`, `accept`, `recv`, `send`, ...
- Glass & Ables, "UNIX for Programmers and Users"
- Rochkind, "Advanced UNIX Programming"
- Bryant and O'Halloran, "Computer Systems: A Programmer's Perspective"
- Other UNIX Programming books...
  - See Reference text list in course profile

24

COMP2303  
COMP7306

## Coming Up

- Week 12 – Friday
  - Ethernet
- Week 13 – Tuesday
  - Internet Layer
- Week 13 – Friday
  - Review

25