TNE30019/TNE80014 – Unix for Telecommunications

Writing Networked Applications – Examples

Dr. Jason But

Swinburne University

Outline

- Simple TCP Server (sample code)
- Issues for more complex programs
- Creating child processes
- Scalable TCP Server (sample code)
- Programming with threads

Simple TCP Echo Server

See sample code `echo_server.cpp`

Adding Complexity with Non-blocking Sockets

Blocking Sockets

- Call to `send()` will block until the kernel accepts data and queues it to send
- Call to `recv()` will block until data arrives over the network and has been processed by kernel

Non-Blocking Sockets

- Created using socket options
- Returns special error code when normal call would block
- Allows single threaded code to manage multiple concurrent input sources

Complex Networked Applications

- Most networked applications are **not** single-threaded
- Multiple threads of execution allow dynamic responses
- Typical in server systems which must respond to potentially hundreds or thousands of clients at same time

Examples

- One process/thread for each connected client
- One thread to manage input while another manages output

- Multi-threading often results in simpler coding of servers cause each client can be treated independently
- However, programming with multiple processes/threads has its own complications

Launching Child Processes – `fork()`

- Traditional way of spawning child process
- Common amongst all Unix systems
- Platform/Unix version independent
- Used in many server products (e.g. Apache)
- Implemented within kernel
- Unix makes exact copy of current process space – creating second process with exactly same state

Original Process

- `fork()` returns process ID of spawned child
- Execution continues at next command following `fork()`

Child Process

- `fork()` returns **0**
- Execution continues at next command following `fork()`

TCP Echo Server with `fork()`

See sample code `echo_server_1.cpp`

Lightweight Threads

Problems with `fork()`

- Completely new process – high usage of system resources
- Entire process space needs to be copied
- Processes cannot directly share data/variables

Lightweight threads

- Creates new process
  - New process state (program counter, registers)
  - Same memory pages (less to copy)
- Shared memory – easier inter-thread communication
- Multi-thread programming issues

POSIX threads (**pthread**) API

- Mostly standardised
- Many newer applications (no history) are using pthreads