COMP 8005 Assignment 2 Pseudocode

Christopher Eng

Feburary 24, 2015

tcp_clnt

main function

Read arguments for host, # of connections, # of data sends, # of seconds between each data send, optional server port to connect to
Create a thread for each connection
Join all created threads
Program end

Thread function

Open a socket with specified host and port
Connect to server over socket
Loop over # of sends
Store start time
Send default data to server
Loop to receive same amount of data back (echo)
Get end time, calculate elapsed time (end time – start time)
Sleep # of seconds
Close socket descriptor and return

tcp_svr

main function

Read arguments for optional server port
Apply mutex attribute to make file mutex process-shared
Initialize output file
Initialize all possible thread_ids to 0
Create a server socket
Bind server socket to a port
Set server socket to listening mode
Fork main into a set number of processes
Each process operation:
Create a default number of child threads running echo function
Initialize a timer signal for 10 seconds
Arm timer
Loop until server is stopped

Obtain connection mutex if all new connections have been started (new sd variable)
Accept a new client, set variable to new sd, increment process client count
Release connection mutex
If there are more clients (with a buffer of 5) than threads, create more threads and increment process thread count (require thread mutex)
If timer signal is triggered, print server details to output file and re-arm timer
Close socket descriptor and exit

Echo function

Loop until thread is cancelled
Obtain connection mutex if new sd variable has been set
Initialize client information storage and save sd variable
Release connection mutex
Set new sd to be non-blocking
Start timer
Loop until one condition breaks out
Receive from sd buffer or check elapsed time
If elapsed time is 5 seconds, assume connection is complete, break out of loop and close socket descriptor
If received a complete message, increment number of client requests, break out of loop and send data back
If connection is closed, decrement client count for process
If there are more threads than clients (with a buffer of 10), exit from this thread

select_svr

main function

Read arguments for optional server port
Setup signal handler to close server socket when CTRL-c is received
Initialize all possible thread parameters
Initialize output file
Create server socket
Make server socket non-blocking
Bind server socket to a port
Set server socket to listening mode
Initialize select set to zero
Add server socket to select set
Create default number of threads, running readerMethod
Initialize timer signal to 10 seconds
Arm timer
Loop until server is stopped
Obtain select set mutex
Initialize set to current server

Run select call on set
If timer signal is triggered, print server details to output file and re-arm timer
Check server socket for new connection and set global variable for thread index to accept
Release select set mutex
Close server socket, free thread parameters, and close

readerMethod function

Allocate memory for all possible child threads
Create default base number of threads, running echo function
Loop until thread is closed
Obtain select set mutex
If reader thread was assigned new connection
Obtain reader thread mutex
Assign new connection to an echo thread by setting global variable for thread
Release reader thread mutex
Add new connection to server set
If the number of clients equals number of echo threads, create more threads
Release select set mutex
Obtain reader thread mutex
Close free threads if there are more threads than clients (buffer of 5)
Release reader thread mutex
Free thread parameters and close

Echo function

Loop until thread is closed
Obtain reader thread mutex if thread does not have a connection
Set thread socket descriptor to new connection
Release reader thread mutex
Check if select has data to read on socket descriptor
Start timer
Loop until one condition breaks out
Receive from sd buffer or check elapsed time
If elapsed time is 5 seconds, assume connection is complete, break out of loop and close socket descriptor
If received a complete message, increment number of client requests, break out of loop and send data back

epoll_svr

main function

Read arguments for optional server port

Setup signal handler to close server socket when CTRL-c is received
Initialize all possible thread parameters
Initialize output file
Create server socket
Bind server socket to a port
Set server socket to listening mode
Create epoll file descriptor for new connections (server socket)
Add server socket to epoll set
Create default number of threads, running readerMethod
Initialize timer signal to 10 seconds
Arm timer
Loop until server is stopped
Wait for epoll event to trigger
Error check event
If timer signal is triggered, print server details to output file and re-arm timer
Obtain mutex
Accept new connection and assign to reader thread with the least connections
Release mutex
Close server socket, free thread parameters, and close

readerMethod function

Initialize pipe
Allocate memory for all possible child threads
Create default base number of threads, running echo function
Create epoll file descriptor for reader thread connections
Add main epoll fd to set
Loop until thread is closed
Wait for epoll event to trigger
Error check event
If fd is main fd, obtain mutex
Check if new connection was assigned to this thread
Add new connection to set
Release mutex
For each other event
Write file descriptor to pipe
Add more threads based on number of concurrent incoming fds
Close free threads if there are more threads than clients (by writing -1 down pipe)
Free thread parameters and close

Echo function

Loop until thread is closed
Read from reader thread pipe
If pipe data is -1, exit thread

Else perform echo read and send based on file descriptor sent