Rattanai Sawaspanich
CS344 Summer2014

Write up: HW5

• Design of pthreads implementation

1. Programs starts
2. main() creates two threads -- one for each player
3. main() itself acts as the third thread -- a referee
4. The program hits the first barrier (all 3 processes have to wait)
5. GO (a global variable get sets to the index of the player who will be hitting the ball)
6. The barrier broadcasts a conditional signal to allow all of the thread to continue
7. Each thread check if GO is equal to its player id (If so, go ahead and hit the ball. If not, hit the barrier)
8. Hit the second barrier. The barrier keep counting up number of thread that hits the barrier
9. Once all thread are waiting, hit_tracker check if the player (with id = GO) hit the ball. If so, (hit_tracker[GO])++;
10. GO = (++GO)%2;
11. repeats 4 to 10
12. The program check if hit_tracker[0] - hit_tracter[1] == 1 to see if who misses the ball while another player hit the ball. Increment the score of the player accordingly
13. reset hit_tracker = 0
14. repeat 4 to 13 until the game ends (if score == 4 then break the forever-loop)

• Design of multiprocess/socket implementation

1. Server starts, creates two different sockets (in the cwd), wait for a signal which indicates the connection of each of the player.
2. Client (player) starts, creates its own sockets, connect its socket to server socket, ping to the server to indicate its connection (by sending "999"), wait for server to respond
3. Once the server receive "999" from a client, the sever store the address of the client's socket in an array (called player), and keep that socket going.
4. The server still waiting for its another socket to be connected. If both of the sockets are connected. The server goes into game-play mode.
5. The server send go_permit ("1") to player1 via server's first socket (sfd[0]), and wait for the result from the incoming data from sfd[0]
6. The client (player) receives "1", it starts hitting the ball with a given hitting rate and send a result to the server
7. repeats 5 and 6 with sfd[1];
8. The sever then check if one player hits, and the other misses; calculate the score; print the score; check if the game ends.
9. If the game is not ended, repeats 5 to 8.
10. If the game ends, the server sends release_permit ("2") to both of its socket to notify players that the game has ended
11. Each of the player breaks free from its forever-loop, and remove() its socket.
12. The server also break out of forever-loop, and remove its socket as well.

Rattanai Sawaspanich
CS344 Summer2014

• Compare and contrast both implementations

I think socket is much easier to implement than the pthread once you understand what is going with sockets and their connections. I mean even if you know how to use pthread, sometimes you can still runs into race-case problem if you do not design your program carefully. Though, sockets require a lot more set up, and connections than pthread. For pthread, all you need to do is to set up a function, make sure when you pthread_create your idex is in the right spot, put up your barrier, and you good to go. Compare to sockets which your server has to creates socket for each of its client, binds the socket, wait for its client to connect. Then, clients has to creates its own sockets, connect the socket to the server ...

• Tell me which implementation you enjoyed the most. If this means which of the implementations was easiest for you, so be it.

Socket is easier. (man page all the way)

• Tell me what kind of an assignment you would like to see from this class, if you could take it again.

HW1:It took a while but I don't think I get anything off of it.

HW2: In the memory of "myar". This assignment took me forever. Countless of hours of screaming, crying, banging head to the wall. It was a hardcore programming experience involving File I/O. With that being said, I learned a lot about file I/O from this assignment. And I think because of this assignment that makes me feel like the I understand class better. (Not only the class material-wise but also the structure of the pro-school classes in general)

HW3: This is a really cool assignment. You got ~~to copy the code from the book~~ to "implement" your own shell. Even though, I still have a feeling that I did not really learn much about signal handling from this assignment.

HW4: I like this assignment too. Tough getting used to multiprocessing can hurt your brain a little bit with all of the race-case conditions that you have never had to deal with them. Overall, I think this is a reasonable assignment. It includes pretty much all the basic stuffs you need to know about pthread.

HW5: I really enjoy working on this last assignment. I think it test you on your understand of concept fairly well and it comes in a manageable size. I like it.