System Programming 2020 Fall

Programming Assignment 2: Auction System

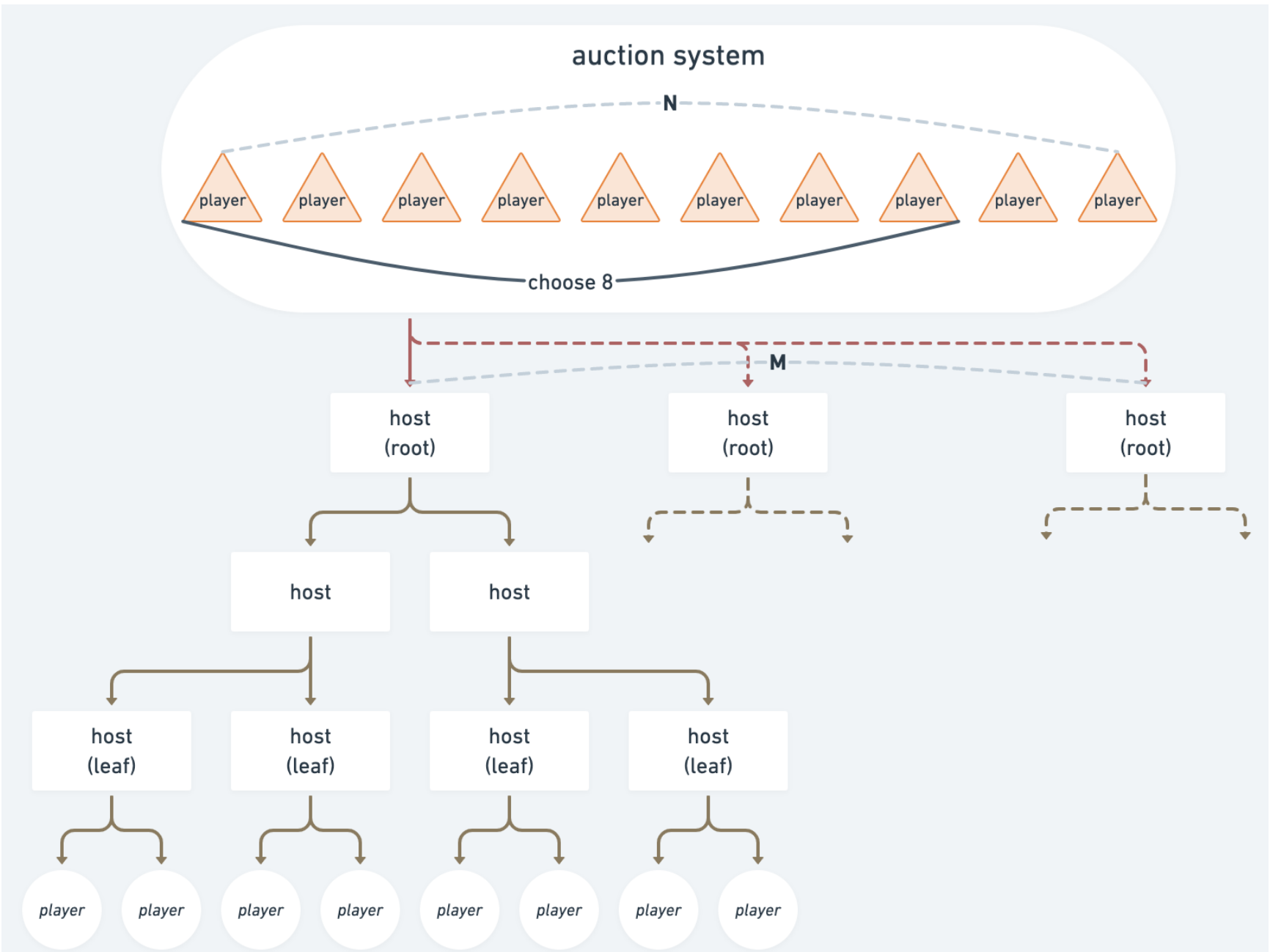
Due: 2020-11-22 23:00 (UTC+8)

1. Problem Description

You are required to implement an auction (競標) system which handles a number of auctions simultaneously.

Goal

1. Understand how to communicate between processes through pipe and FIFO
2. Practice using fork to create processes
3. Write a program in shell script



There are N players using **one** auction system. The auction system has M hosts. Every host holds an auction at a time. The auction system will continuously choose **8** players and distribute them to every host, until all combinations of players are distributed. Every combination will only be distributed once, i.e., there will be $\binom{N}{8}$ auctions to hold. If $\binom{N}{8} > M$, there will not be enough hosts to hold all auctions at once. Once running out of host, the auction system has to wait until an auction has ended, and it can distribute another 8 players to the available host.

Once a host received 8 players from the auction system, it will recursively fork itself **2** times. In other words, the host (root) first forks itself into two child hosts. The two child hosts then fork themselves into a total of four leaf hosts, two each. So, Each leaf host will be distributed to 2 players.

After the tree structure is constructed, there will be **10** rounds of bid comparison in an auction to decide the winner. For each round, child hosts have to report winner to their parents recursively. The root host collects the winners of all 10 rounds, and it ranks the players. Afterwards, the root host reports the rankings to the auction system. The auction system then transforms the rankings into scores and sum up scores of all $\binom{N}{8}$ auctions. The implementation details will be explained later.

For example, suppose there are 9 players, A, B, C, D, E, F, G, H, I, and 2 hosts: H1, H2. We get $\binom{9}{8}$ = 9 combinations of players. Say we first assign (A, B, C, D, E, F, G, H) to H1 and (A, B, C, D, E, F, G, I) to H2, then other combinations have to wait for H1 or H2 to finish.

For H1, it will fork itself into two child hosts: H11, H12. They will be assigned (A, B, C, D) and (E, F, G, H) respectively. Same for H2.

For H11, it will further fork itself into H111 and H112. They will be assigned (A, B) and (C, D) respectively. Same for H12, H21, H22.

In a round, H111 compares the bid of A and B. It reports the winner to H11. Then H11 compares the bid of the winners of H111 and H112, and reports the winner. Eventually, H1 (root host) compares the bid of the winners of H11 and H12, and gets the winner of all 8 players.

After 10 rounds, the root host reports the ranking to the auction system.

The auction system sums up scores of 9 auctions, and outputs the result.

2. Implementation & Specs

You will have to write 3 programs. You have to strictly follow the implementation as follows. However, there may be details that are not written below. You have to ensure your program runs smoothly. If you think the specs are not clear enough, you can ask in the discussion forum on NTU COOL.

auction_system.sh

- **You must write this program totally in shell script.**
  - Some useful commands for this program:
    - local
    - exec
    - read
    - seq
    - awk
    - mkfifo
    - wait
  - Great resources for self-studying shell script
    - Chinese (Taiwanese Mandarin): http://linux.vbird.org/linux_basic/0340bashshell-scripts.php
    - English: https://www.learnshell.org/
- Running the program

```
bash auction_system.sh [n_host] [n_player]
```

  - It requires two arguments
    - `n_host`: the number of hosts (1 ≤ M ≤ 10)
    - `n_player`: the number of players (8 ≤ N ≤ 12)
- Implementation (in order)
  1. Prepare **FIFO** files for all M hosts.
     - You have to use FIFO to communicate between auction system and hosts. So also create FIFOs named: `fifo_0.tmp` for read, and `fifo_{host_id}.tmp` for write.

- The host ids are numbered from 1 to M.
- Suppose M = 2, your auction system should create 3 FIFOs.
  - `fifo_0.tmp`: read responses from host 1 and host 2
  - `fifo_1.tmp`: write message to host 1
  - `fifo_2.tmp`: write message to host 2

2. Generate $\binom{N}{8}$ combinations of N players and assign to hosts via FIFO. Collect rankings from hosts and calculate the scores.
   - The messages sent to hosts are of the format:

     ```
     [player1_id] [player2_id] [player3_id] [player4_id] ... [player8_id]\n
     ```

     - The player ids are numbered from 1 to N.
     - Suppose players 1, 3, 5, 6, 7, 8, 9, 10 are chosen, the message should be like

       ```
       1 3 5 6 7 8 9 10\n
       ```

     - If there is no available host, the auction system should wait until one of the hosts returns the result, and assign another 8 players to that host.
       - **Do not kill the host and fork a new one**.
       - Do not make any host idle.
     - **The player in the first, second, ..., eighth place gets score of 7, 6, 5, 4, 3, 2, 1, 0 respectively.** Simply sum the scores up to get the final score.
3. After all combinations are hosted, send an ending message to all hosts forked.

   ```
   -1 -1 -1 -1 -1 -1 -1 -1\n
   ```

4. Print the final scores ordered by player id (ascending) to stdout.
   - Format:

     ```
     [player_id] [score]\n
     ```

5. Remove FIFO files.
6. Wait for all forked process to exit.

host.c

- The compiled execution file must be named `host`.
- Running the program

  ```
  ./host [host_id] [key] [depth]
  ```

  - It requires three arguments
    - `host_id`: the id of host
    - `key`: a key randomly generated for each host (0 ≤ key ≤ 65535)
      - Used for auction system to map the responses from hosts, since all hosts write to the same FIFO file.
      - The host_id and key of forked child hosts should be the same as their parents.
    - `depth`: the depth of hosts
      - Starting from 0 and incrementing by 1 per fork

- Implementation (in order)
  1. For root host, prepare to read and write FIFO file.
  2. For hosts except leaf hosts, Fork child hosts.
     - You have to use **pipe** to communicate between hosts. So also create 2 pipes each. One for writing to child host. The other for reading from child host.
     - You also have to make the forked child hosts read from **stdin** when sending player id to them, and make them write to **stdout** when they send back the winner and bid.
  3. Loop and get player list from parent until receiving the ending message.
     1. Send half the players to each child host if not leaf host.
        - e.g. the root host have to send

          ```
          [player1_id] [player2_id] [player3_id] [player4_id]\n
          ```

          to the first child host, and

          ```
          [player5_id] [player6_id] [player7_id] [player8_id]\n
          ```

          to the second child host. Note that the order of players has to be reserved.
     2. Fork *player*s for leaf host. ("*player*" here refers to the program compiled from `player.c`)
        - You have to use **pipe** to communicate between leaf hosts and *player*s. So also create 2 pipes each. One for writing to *player*. The other for reading from *player*.
        - You also have to make the forked *player*s write to **stdout** when they send back the player and bid.
        - Details of arguments passing to the *player*s are in the next part.
     3. Iterate 10 rounds of bid comparison
        - In every round, read the 2 players and their bids from child hosts or *player*s via pipe. Compare their bids, and then send the winner and their bid to parent host, except the root host.
          - **Do not kill the any of the child host or *player* and fork a new one**.
        - For root host, record the winner in every round. In other words, **the final winner of a round gets 1 point. The others don't get any point. Sum them up after 10 rounds.**
        - ***Player* should end after all 10 rounds are completed, and you have to fork it again next time** (step 3-2).
     4. For root host, transform the final score to ranking and send to the auction system.
        - Format:

          ```
          [key]\n
          [player1_id] [player1_rank]\n
          [player2_id] [player2_rank]\n
          ... ...
          [player8_id] [player8_rank]\n
          ```

        - e.g. if player 1, 3, 5, 6, 7, 8, 9, 10 gets 1, 2, 1, 0, 3, 2, 1, 0 respectively, the message should be

          ```
          [key]\n
          1 4\n
          3 2\n
          5 4\n
          6 7\n
          7 1\n
          8 2\n
          9 4\n
          10 7\n
          ```

     5. Wait for forked *player*s to exit and close pipes.

4. Wait for forked child hosts to exit and close pipes. For root host, close the FIFO files.

player.c

- The compiled execution file must be named `player`.
- Running the program

```
./player [player_id]
```

  - It requires one argument
    - `player_id`: the id of player
- Implementation (in order)
  1. Iterate 10 rounds of bid comparison
     - In every round, send the player id and bid to leaf host.
       - Format:

```
[player_id] [bid]\n
```

       - **The bid must follow the rules** for TAs to check your work:

```
bid_list[21] = {
    20, 18, 5, 21, 8, 7, 2, 19, 14, 13,
    9, 1, 6, 10, 16, 11, 4, 12, 15, 17, 3
};
bid = bid_list[player_id + round - 2] * 100;
```

         - `round`: 1 to 10
         - e.g. player_id = 1, round = 1 → bid = 2000
         - There must be a winner (no draw) in every round under this rule.
- You don't need to make all rounds start and end at the same time among hosts and players. Just keep sending messages whenever you can do so.

3. Sample Execution

```
$ bash auction_system.sh 1 8
```

This auction system runs with 1 host and 8 players. It creates `fifo_0.tmp` and `fifo_1.tmp`. Then, it forks and executes root host. (suppose the key generated randomly is 3427)

```
./host 1 3427 0
```

The auction system then assigns players to root host via `fifo_1.tmp`.

```
1 2 3 4 5 6 7 8\n
```

The root host forks two child hosts.

```
./host 1 3427 1
```

```
./host 1 3427 1
```

It then assigns players to child hosts via pipes.

```
1 2 3 4\n
```

```
5 6 7 8\n
```

Each child host forks two leaf hosts.

```
./host 1 3427 2
```

```
./host 1 3427 2
```

...

It then assigns players to leaf hosts via pipes.

```
1 2\n
```

```
3 4\n
```

...

Each leaf host forks and executes two *player*s.

```
./player 1
```

```
./player 2
```

...

Assume that player 1 bids 100, player 2 bids 200, ..., player 8 bids 800 for all 10 rounds. (Actually, you have to follow the rules in part 2.) Each player sends their bid to leaf host.

```
1 100\n
```

```
2 200\n
```

...

For the leaf host that receives message from players 1 and 2, it compares the bid and finds that player 2 wins. So, it send the winner, player 2, to its parent.

```
2 200\n
```

...

For the mid-level host that receives the above message and another message from another leaf host (player 3 and 4), it compares the bid and finds that player 4 wins. So, it send the winner, player 4, to its parent.

```
4 400\n
```

...

At last, root host finds out the winner, player 8, in this round.

After 10 rounds, the root host sends the ranking to the auction system via `fifo_0.tmp`.

```
3427\n
1 2\n
2 2\n
3 2\n
4 2\n
5 2\n
6 2\n
7 2\n
8 1\n
```

Since all auctions (combinations of players) are done, the auction system sends ending message to root host.

```
-1 -1 -1 -1 -1 -1 -1 -1\n
```

Then root host to its child host.

```
-1 -1 -1 -1\n
```

Then to leaf host.

```
-1 -1\n
```

The auction system outputs the final score.

```
1 6\n
2 6\n
3 6\n
4 6\n
5 6\n
6 6\n
7 6\n
8 7\n
```

Note that if you follow the rules of bid, you should get

```
1 7\n
2 7\n
3 7\n
4 7\n
5 1\n
6 3\n
7 3\n
8 1\n
```

4. Grading

There are 6 subtasks in this assignment. You can get 8 points if you finish all of them. We will test your code on CSIE workstation.

1. (0.5 point) Produce executable files successfully.

   Your Makefile can generate host and player. Running `$ make` should generate all executables.

2. (3 points) Your auction system works fine.

   We will test if your auction system could successfully communicate with TA's host and player, and output correct results.

Not writing this program totally in shell script won't get any point.

3. (3 points) Your host works fine.

   We will test if your host could successfully communicate with TA's auction system and player, and output correct results.

4. (1 point) Your player works fine.

   We will test if your player could successfully communicate with TA's auction system and host, and output correct results.

5. (0.5 point) Your programs work fine together.

   We will test if your auction system, host, and player could successfully communicate with each other, and output correct results.

There are host and player binaries by TA (compiled on CSIE workstation) in your GitHub repository. You could make use of them to write and test your code.

5. Submission

Your assignment should be submitted to Github before deadline. The submission should include at least four files:

1. Makefile
2. auction_system.sh
3. host.c
4. player.c

You can submit other .c, .h files, as long as running `$ make` can generate all executables needed.

6. Reminder

1. Note that you have to do `fflush()` if you use `printf()` to write to pipe.
2. There is a simple auto-grading in your GitHub repository. We won't grade your work by that, but you could make use of the test cases provided.
   - It will run auto-grading every time you push your local commits to the remote repository.
   - Test cases can be find in `.github/classroom/autograding.json`
   - Note that if you look into the test cases, you will find that the newlines are `\r\n`. Simply ignore it. Stick to `\n` in your work. Also, the tests will still pass on GitHub if you use `\n`.
3. Plagiarism is STRICTLY prohibited.
4. Your credits will be deducted 20% for each day delay. A late submission is better than absence.
5. If you have any question, feel free to contact us via email ntucsiesp@gmail.com or come during TA hours.
6. Please start your work as soon as possible, do NOT leave it until the last week!