System Programming Assignment 2

Due: 23:59 Tue, Nov 21, 2017

1 Problem Description

In assignment 2, you are required to implement a bidding system which will handle a sequence of competitions. The goal of this assignment is to practice how to communicate between processes through **pipe** and **FIFO**, and to understand how to use **fork** to create processes.

There is only one **bidding system** which has a list of players. The bidding system will distribute every four **players** to a competition held by a **host**. A competition will be held ten rounds, and every player will get 1000 dollars each round. In each round, each player need to announce how much he would like to spend on each item. For example, four players, A, B, C, D attend this competition. In the first round, they tell the host 900, 900, 500, and 300 separately. After the host gets their announcements, it will judge that C wins the item. This is because both A and B announce the same number of money in this round, the host will find the second large and unique one to win the item. In this round, C will pay 500 dollars, and the other three, A, B, and D will get their money back. In the second round, A, B, and D will have 2000 dollars, while C only has 1500 dollars. The result depends on the number of items they got after ten rounds in the competition.

One announcement is called a round. At the beginning of each round, the host will tell players how much money do they have as well as others', and they cannot announce more than they really have. A competition will be held ten rounds, and only one player will win the item in each round.

The bidding system, needs to schedule every four players to each host. That is, if there are N players, there will be **C(N,4)** competitions. However, the number of host may be more than or less than the number of competitions, and a host can only hold a competition at a time.

After a competition is finished, the host should return the rankings of the competition to the bidding system. The bidding system should then add the scores to the corresponding player accumulatively. The bidding system should wait until all competitions completed and rank all players according to their accumulative scores.

To sum up, you are required to do the following subtasks,

1. Execute bidding system, and fork a given number of hosts. Bidding system should communicate to hosts through pipes.

2. Each host should create 5 FIFOs, a FIFO to read from players, 4 FIFOs to write to players. Once the host receives 4 players from bidding system, it needs to fork 4 players, and starts the competition.

3. A host should keep the result of a competition. Once the competition is finished, it writes the result to bidding system via pipe. Once all competitions is finished, the bidding system shows the rankings of all players, and closes all hosts.

2 format of inputs and outputs

There are three programs: **bidding_system.c**, **host.c**, **player.c**

2.1 bidding_system.c

./bidding_system [host_num] [player_num]

It requires two arguments, the number of hosts($1 \le host\_num \le 12$), and the number of players($4 \le player\_num \le 20$).

At first, the bidding system should fork and execute the number of hosts specified by the argument, with id from 1 to *host_num*. The bidding system must build pipes to communicate before executing them. The message coming from the host would be the rankings of players of the competition held by the host(described in the host.c part). After bidding system executes a desired number of hosts, it should then distribute 4 players to an available host via pipe. The players' id are numbered from 1 to *player_num*, so the messages sent to the host are of the format in the following,

[*player*1_*id*] [*player*2_*id*] [*player*3_*id*] [*player*4_*id*]\n

Please keep every four players' id in **ascending** order.

If there is no available host, the bidding system should wait until one of the hosts return the competition result, and assign another 4 players to that host. There will be C(*player_num*, 4) competitions in total. You need to make sure that you make full use of all hosts and try not to let any available host idle.

The bidding system should keep accumulative scores of all players. The accumulative scores are initialized to 0. When a host return the competition result, the bidding system should add scores to the corresponding player accumulatively. The player in the first, second, third, and fourth place gets 3, 2, 1, 0 separately.

After all competitions are done, the bidding system should send "-1 -1 -1 -1\n" to all hosts, telling them that all competitions are finished so that they can exit. Then, the bidding system outputs all players' id in increasing order and their corresponding ranks, separated by a

space. For example, if there are five players and their accumulative scores are 7, 10, 3, 7, 3, the bidding system should output

```
1 2\n
2 1\n
3 4\n
4 2\n
5 4\n
```

2.2 host.c

./host [host_id]

It requires an argument, the id of the host. The host should create five FIFOs:

1. host[*host_id*].FIFO: read responses from 4 players
2. host[*host_id*]_A.FIFO: write message to player_A
3. host[*host_id*]_B.FIFO: write message to player_B
4. host[*host_id*]_C.FIFO: write message to player_C
5. host[*host_id*]_D.FIFO: write message to player_D

The host should read from **standard input**, waiting bidding system to assign 4 players in. After knowing the players, the host forks 4 child processes, and executes 4 player programs, starting a ten-round competition. In each round, the host first tells all these 4 player how much money do they have(as well as others) via specific FIFO to help them make their decisions. Notice that players cannot announce an amount of money larger than they really have. Hosts should judge which player wins the item in each round, and calculate how much money do they have in the next round, which is defined as

$$money_{next} = 1000 + money_{prev} - \alpha \times pay$$

where $money_{next}$ is the money in the next round, $money_{prev}$ is the money in the previous round. $\alpha = 1$ if the player wins the item in the previous round, otherwise $\alpha = 0$. *pay* is an amount of money that the player was willing to pay in the previous round. Messages from hosts to players should be of the format in the following,

[*player_A money*] [*player_B money*] [*player_C money*] [*player_D money*]\n After giving out the message to each player, the host should continue to collect money coming from 4 players. The format of the message will be describe in the player.c part.

The host accumulates each player's score, which means the number of items got in this competition. After 10 rounds, the host needs to output the following to **standard output**,

[*playerA_id*] [*playerA_rank*]\n
[*playerB_id*] [*playerB_rank*]\n
[*playerC_id*] [*playerC_rank*]\n
[*playerD_id*] [*playerD_rank*]\n

where the ranks are ordered from 1 to 4. If some players get the same scores, they will be ranked at the same place, and the following one will be ranked according to the number of players whose scores are higher. For example, if four players, player_1, player_2, player_3, and player_4 get 3, 4, 3, 0 respectively. Then the host will rank player_2 the first place, and rank both player_1 and player_3 the second place, and rank player_4 the fourth place.

After sending out the result to bidding system, the host should wait until bidding system assigns another competition. However, when the host receives "-1 -1 -1 -1" from bidding system, it indicates that all competitions are done, so the host should exit.

2.3 player.c

./player [*host_id*] [*player_index*] [*random_key*]

It requires three arguments. *host_id* is the host id that holds the competition. *player_index* should be one of {'A', 'B', 'C', 'D'}, the index of a player in a competition. *random_key* would be an integer for a player($0 \le random\_key \le 65535$), and should be randomly generated unique for 4 players in the same competition. It is used to verify if a response really comes from that player.

Note that the player index is NOT the same as the player id. It is just a index of each player and used to indicate a certain player in a competition.

The player should open a FIFO named host[*host_id*]_[*player_index*].FIFO, such as host1_A.FIFO, which was already created by the host. The player reads the message from host[*host_id*]_[*player_index*].FIFO and writes the message to host[*host_id*] .FIFO.

Take host1 and player_A as an example,

$$host1.FIFO \xleftarrow{\text{read}} host1 \xrightarrow{\text{write}} host1\_A.FIFO \xleftarrow{\text{read}} player\_A \xrightarrow{\text{write}} host1.FIFO$$

At the beginning of each round, the players should read messages from the host in certain format(which we defined in host.c). Then they should write their responses to the host in the following format,

[*player_index*] [*random_key*] [*money*]\n

indicating the index of the player, the random key given in the argument, and the money that players want to pay.

The above process will be repeated. The players must guarantee that it correctly gives out 10 responses. The player will exit after it gives out 10 responses, and will be executed again when competing in another competition.

[*] In order to make grading criteria simpler, please write two versions of your player.c.

1. *player.c*
   Players in the same competition will pay all their money in turn, while others pay 0. The order should follow $A \rightarrow B \rightarrow C \rightarrow D \rightarrow A \cdots$. For example, in round 1 , player_A pays 1000, while others pay 0. In round 2, player_B pays 2000, while others pay 0. There should be a unique result if you follow this rule, and we will grade your codes in this version.

2. *player_bonus.c*
   You can implement all possible algorithms you come up with. This version of player.c will be only used in the BONUS part.

3 Sample execution

`$ ./bidding_system 1 4`

This will run 1 host and 4 players. The bidding system will fork and execute:

`$ ./host 1`

The host will create:

host1.FIFO host1_A.FIFO
host1_B.FIFO host1_C.FIFO
host1_D.FIFO

The bidding_system sends to host 1(host 1 reads from stdin)

`1 2 3 4`

The host executes:

`$ ./player_1 1 A 302`
`$ ./player_2 1 B`
`2015 $ ./player_3 1 C`
`1126`
`$ ./player_4 1 D 65535`

Round 1, host 1 sends to player 1 through *host1_A.FIFO*:

`1000 1000 1000 1000`

Round 1, player 1 sends to host 1 through *host1.FIFO*:

`A 302 900`

Round 2, host 1 sends to player 1 through *host1_A.FIFO*:

(assume A pays 900, B pays 900, C pays 500, D pays 300 in round 1)

`2000 2000 1500 2000`

Round 2, player 1 sends to host 1 through *host1.FIFO*:

`A 302 1500`

The above process runs 10 rounds.

Host 1 writes the result to stdout(sending to bidding_system):

```
1 3
2 4
3 1
4 2
```

The bidding_system reads the result, and does the calculation:

player 1's accumulative score + 1 = 1 player
2's accumulative score + 0 = 0 player 3's
accumulative score + 3 = 3 player 4's
accumulative score + 2 = 2

All competitions are over. The bidding_system sends to host 1:

```
-1 -1 -1 -1
```

The host 1 terminates.

The bidding_system outputs the result:

```
1 3
2 4
3 1
4 2
```

4 Grading

There are 7 subtasks in this assignment, you can get 7 points if you finish all of them.

1. (1pt)Your *bidding_system* works fine.
   We will use your *bidding_system* as well as TA's *host* and *player* to testify whether your *bidding_system* could successfully communicate with TA's *host* and output correct result.
2. (0.5pt)Your *bidding_system* schedules *host* effectively.
   That is, you should implement *select* to make sure not to let available *host* idle.
3. (0.5pt)Your *bidding_system* executes *host* correctly.
   You should not fork new *host* when you want to hold new competition. That is, you can only fork *host host_num* times, and assign new competition to one of them.
4. (2pt)Your *host* works fine.
   We will use your *host* as well as TA's *bidding_system* and *player* to testify whether your *host* could successfully communicate with TA's *bidding_system* and *player*, and output correct result.
5. (1pt)Your *player* works fine.

We will use your *player*(version 1) as well as TA's *bidding_system* and *host* to testify whether your *player* could successfully communicate between TA's and *host*, and output correct result.

6. (1.5 pt)Completeness.
   If you successfully produce correct result with all your *bidding_system*, *host* and *player*(version 1).

7. (0.5pt)Produce executable files successfully.
   Your Makefile can generate *bidding_system*, *host* and *player*.

5 Bonus

Your player(version 2) will compete with other's players under TA's bidding_system and judge. We will randomly partition all of you into 6 groups, and randomly assign player to each group.

In each group, top 2 players will get 0.5 extra bonus. We will then get all top 2(total 12 players) and run the competition again. The overall top 2 will get another 0.5 bonus points.

You don't need to implement the partition and multiple group part. This is just a reminder on how the bonus would work.

For those players do not work correctly, they will automatically lose the competition. Please do not do things evil to other players, like open others' FIFOs, and do not announce the amount money larger than you really have. We will detect this and it will also result in an automatical lose.

6 Notes

1. Remember to flush(*fflush*(), *fsync*()) whenever you write messages to pipes or FIFOs to ensure the message being correctly pass out.

2. For the host, remember to clear the FIFOs before a new competition begins, in case any player died in the last competition and did not read all message.

7 Submission

Your assignment should be submitted to CEIBA before the deadline, or you will receive penalty.

At least 6 files should be included:

1. bidding_system.c
2. host.c

3. player.c (version 1)
4. player_bonus.c (version 2)
5. Makefile(as well as other *.c files)
6. readme.txt

Since we will directly execute your Makefile, therefore you can modify the names of .c files, but Makefile should compile your source code(bidding_system.c, host.c, player.c, player_bonus.c) into three executable files named *bidding_system*, *host*, *player* and *player_bonus*. Please put your student ID(lower-case) and name in a comment at the first line of your source code, with the following format

/*STUDENT_ID NAME*/

In readme.txt, it should contain three parts.

1. Execution
   Please teach us how to run your program in case we could not run your code successfully.
2. Description
   Please briefly state how do you finish your program and something valuable you want to explain. What algorithm do you use in the bonus?
3. Self-Examination
   Please write down which grading criteria do you finish and briefly describe how you make it. We will grade your assignment by the points you mentioned.

These files should be put inside a folder named with your student ID (in lower case) and you should compress the folder into a .tar.gz before submission. Please do not use .rar or any other file types.

The commands below will do the trick. Suppose your student ID is b02902000:

```
$ mkdir b02902000
$ cp Makefile readme.txt *.c b02902000/
$ tar -zcvf SP_HW2_b02902000.tar.gz b02902000/
$ rm -r b02902000/
```

At the first line of bidding_system.c, host.c, player.c would be

/*b02902000 王小明 */

Please do NOT add executable files to the compressed file. Errors in the submission file (such as files not in a directory named with your student ID (in lower case), executable files not named correctly, and so on) may cause deduction of your credits. Submit the compressed file SP_HW2_b02902000.tar.gz to CEIBA.

8 Reminder

1. Plagiarism is STRICTLY prohibited.
2. Your credits will be deducted 5% for each day delay, but a late submission is still better than absence. For those late submissions, he or she will lose the chance to get bonus points.
3. If you have any question, you can contact us via email or come to R302.
4. Please start your work ASAP and do not leave it until the last day!