2014 System Programming Assignment 2

Announce: Mon, Oct 27, 2014
Due: 12:00 Mon, Nov 24, 2014

I. Problem Description

In this assignment, you are going to practice communicating among processes. The goal of this assignment is to practice multiple -process control via pipes and FIFOs, and to understand the use of fork() and exec(). We will propose a game so that you can enjoy this. The game is described as follows. The game is "Old Maid", in general, we named it with "draw joker".

There are 53 cards: four couple of (1, 2, … , J, Q, K) and one joker.
There is a organizer to arrange the schedule of those games and it records that the score of those games. A judge who maintains the fairness of the game is distributed to a game which have four players participate in it.

The organzier communicated with the judges through pipes while the judges communicated with player through FIFOs

More detail rules wil be described as below.

http://zh.wikipedia.org/wiki/%E6%BD%9B%E7%83%8F%E9%BE%9C

II. Format for inputs and outputs

Three programs are required: "organizer.c", "judge.c", and "player.c"

● organizer.c (**$./organizer [judge_num] [player_num]**)

It requires two arguments, the number of judges (1 <= **judge_num** <= 12) and the number of player (8 <= **player_num** <= 16)
(we assume that a player can play different games at the same time)

At first, the organizer should fork and execute the number of games specified by the argument, with IDs from 1 to **judge_num**. The organizer must build pipes to communicate with each of them before executing them. The message coming from the judge would be the competition result (the loser of the game) presided by that judge, which will be described later in "judge.c" part.

After organizer executes judges, the organizer then has to distribute every 4 players to an available judge via pipe. The players are numbered from 1 to player_num, so the message sending to the judges are of the format shown in the following:

```
[p1_id] [p2_id] [p3_id] [p4_id]
```

If there is no available judge, then the organizer waits until one of the judges returns the competition result, so that it can assign another competition to that judge. There will be C(player_num, 4) competitions needed to be assigned. Ideally, the organizer should make full use of available judges but not let any available judge idle.

The organizer has to keep accumulative scores of all players. The accumulative scores of all players are initally set to 0. When the judge returns the result (the loser of the game) back to the organizer, the organizer should add score -1 to the players' accumulative scores.

After all competitions are done, the organizer should send the string "0 0 0 0" to all judges, indicating that all competitions are done and judges can exit. Then, the organizer outputs all players' ID sorted by their scores, from the lowest to the highest, separated by spaces. If two players have the same score, output the one with smaller ID first. For example, if there are 5 players with ID 1, 2, 3, 4, 5 and have accumulative scores -2, -1, -1, 0, and -1, the organizer should output "1 2 3 5 4".

● judge.c (**$./judge [judge_id]**)

It requires an argument, the ID of the judge. The judge should create a FIFO named judge[judge_id].FIFO, such as judge1.FIFO, to read responses from the players, and create four FIFOs named judge[judge_id]_A.FIFO, judge[judge_id]_B.FIFO, judge[judge_id]_C.FIFO, judge[judge_id]_D.FIFO, to write messages to the players in the competition held by this judge.

The judge should read from standard input, waiting for the organizer to assign four players in. After knowing the players, the judge forks four child processes, run exec() to execute the player programs.

First, the judge send 14 cards to player A, 13 cards to player B, C, D in the following format: (there are 53 cards in this game, four couple of (1, 2, … , 11, 12, 13) and one joker (0))

```
[card_1] [card_2] [card_3] … [card_14]        (if it sends to player A)
[card_1] [card_2] [card_3] … [card_13]        (if it sends to player B, C, D)
```

Next, after each player organizes its cards, the judge should read messages from judge[judge_id].FIFO to get the number of cards of each player.

game flow:
In the beginning, all of the four players are sorted by their ID from the lowest to the highest. Assume that the player A has the smallest ID and player B has the second small ID, and so on.

1. The judge told player A the number of cards (n_{card}) of player B
2. Player A told judge the card ID (0 <= card ID < n_{card}) that he want to draw
3. The judge told player B the card ID had been drawn and player B updated his cards
4. Player B told judge the card (ex: the card is J) of the card ID
5. The judge told player A the card of the card ID (ex: the card is J)

6. Player A told judge whether he/she eliminated the two card with the same number, that is, whether the number of his/her cards decreased by one and updated his cards
7. And then the action of player A is finished and player B start its action (from 1. to 7.) on player C (player C do action on player D and player D do action on player A)

(note 1: The judge didn't need to know the cards of the all players. It just needed to record the number of cards of all players)
(note 2: If the player have no cards, it should be skipped.)

When all pairs of the card are eliminated except for the joker, the game is completed. The player who hold the joker is the loser of this game.
The judge should write the result to standard output (sending to organizer) in the following format:

    [the_loser_ID]

Noted that **the_loser_ID** is the ID of a player that got from the organizer rather than the index of the player (A, B, C, D) in a game

After sending out the competition result, the judge shall kill all player(A, B, C, D) processes (usage: ***kill(pid_t pid, int sig)***) and wait until the organizer assigns another competition, and do what is described above again. But when judge got "0 0 0 0" from the organizer, it indicates that all competitions are done, so the judge should exit.

● player.c (**$./player [judge_id] [player_index] [random_key]**)

It requires three arguments. **judge_id** is the judge ID that holds the competition. **player_index** would be a character in {'A', 'B', 'C', 'D'}. It is the index of this player has in this competition. **random_key** would be an integer in [0, 65536), used for this player in this competition, and should be randomly generated unique for four players in the same competition. It is used to verify if a response really comes from that player.

Notice that the player index here is NOT the same as the player ID in judge/organizer. It just means which index the player has in this competition. The player should open a FIFO named judge[judge_id]_[player_index].FIFO, which should be already created by the judge. The player reads messages from judge[judge_id]_[player_index].FIFO, such as judge1_A.FIFO, and writes responses to judge[judge_id].FIFO, such as judge1.FIFO.

After read the messages from judge[judge_id]_[player_index].FIFO, each player should eliminate those two-card pairs which have the same number.
for example, if a player get 13 cards: (0, 3, 5, 6, 6, 7, 7, 8, 9, 11, 12, 12, 12).
After eliminated process, the cards of the player should be: (0, 3, 5, 8, 9, 11, 12)

And then, players should send the number of cards to judge through judge[judge_id].FIFO in the following format:

    [player_index] [random_key] [number_of_cards]

Now, the game start. In the first round, player A's turn, following by the steps below:
Blue argument is sent from the judge and player A should read from judge[judge_id]_A.FIFO

1. the judge send the number of cards of player B to player A in the following format:
   [type] [number_of_cards]
   The value of type" is "<" to indicate that it is the player A's turn to draw.

2. the player A should send the card ID that it wants to draw to judge in the following format:
   [player_index] [random_key] [card_ID]

3. The judge send the card ID had been drawn (by random choose) to player B in the following format:
   [type] [card_ID]
   The value of type" is ">" to indicate that the card_ID of player B have been drawn.

4. player B send the card of the card ID to judge in the following format:
   [player_index] [random_key] [the_card_number]
   Note that we use 11 to indicate the card of J, 12 to indicate the card of Q, 13 to indicate the card of K and 0 to indicate the joker.
   player B should update its card before it send message to the judge

5. The judge send the card of the card ID to player A in the following format:
   [the_card_number]

6. player A told judge whether he/she eliminated the two card with the same number, that is, whether the number of his/her cards decreased by one in the following format:
   [player_index] [random_key] [eliminated]
   if **eliminated** is 0, indicated player A doesn't have two cards with the same number
   if **eliminated** is 1, indicated player A have two cards with the same number

one round contains 4 turns (player A, B, C, D).
Repeat the round several times until only one player have cards.

III. Sample execution

All of the information sent through FIFO or pipe should be add "\n" to the end of your string for your convenience.

$ ./organizer 1 4
This will run 1 judge and 4 players. The organizer will fork and execute:
$ ./judge 1
The judge will create:

judge1.FIFO
judge1_A.FIFO
judge1_B.FIFO
judge1_C.FIFO
judge1_D.FIFO

The organizer sends judge 1 (judge 1 reads from standard input):
1 2 3 4
The judge executes:
$ ./player 1 A 9
$ ./player 1 B 2014
$ ./player 1 C 10000
$ ./player 1 D 65535

The judge 1 sends 14 or 13 cards to every players through judge1_{A, B, C, D}.FIFO
0 3 5 6 6 7 7 8 9 11 12 12 12 (ex: sends player B through judge1_B.FIFO)

each player send the number of cards to judge through judge1.FIFO
A 9 5
B 2014 7
C 10000 10
D 65535 8

In round 1, player A's turn:
judge 1 sends player A through judge1_A.FIFO:
< 7 (player B has 7 cards)

player A sends judge 1 through judge1.FIFO:
A 9 6 (player A wants the 6th card of player B)

judge 1 sends player B through judge1_B.FIFO:
> 4 (the 4th card of yours is picked)

player B sends judge 1 through judge1.FIFO:
B 2014 11 (the card is J)

judge 1 sends player A through judge1_A.FIFO:
11 (player A gets J)

player A sends judge 1 through judge1.FIFO:
A 9 0 (player A doesn't have the same cards)

one round contains 4 turns (player A, B, C, D).
Repeat the round several times until only one player have cards.

judge 1 writes the result to standard output (sending to organizer):
2 (means that the player with ID 2 is the loser of this game)

judge 1 kill all player processes

The organizer reads the result, and does the calculation:
player A’s accumulative score + 0 = 0
player B’s accumulative score + (-1) = -1
player C’s accumulative score + 0 = 0
player D’s accumulative score + 0 = 0

All competitions are over.
The organizer sends judge 1:
0 0 0 0
The judge 1 terminates.

The organizer outputs the result:
2 1 3 4 (player with ID 2 got most negative score)

IV. Tasks and scoring

There are 6 subtasks in this assignment. By finishing all subtasks you earn the full 7 points.

1. Make to generate three execution file. (1 point)
2. organizer.c correctness (1 point)
3. judge.c correctness (2 points)
4. player.c correctness (1 point)
5. Processes handling (1 point)
6. Player’s cheating detection (1 point)

V. Notes

● Remember that every time you writes a message to a pipe or a FIFO, you should use fflush()to ensure the message being correctly passed out.
● For the judge, remember to clear the FIFO before a new competition begins, in case any player died in last competition and didn't read all message.
● You are not allowed to shuffle cards in player.c.

VI. Submission

Your assignment should be submitted to the course website by the due. Or you will receivepenalty. At least five files should be included:

1. **organizer.c**
2. **judge.c**
3. **player.c**
4. **Makefile (as well as other *.c)**
5. **README.txt**

Since we will directly run your Makefile, therefore you can modify the names for .c files, but Makefile should compile your source into three executable files named organizer, judge, and player. In README.txt, please briefly state how to compile your program, and anything you have done besides the basic functionality. These files should be put inside a folder named with your student ID and you should compress the folder into a .tar.gz before submission. Please do not use .rar or other file type.

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

```
$ mkdir b02902000
$ cp Makefile README.txt *.c b02902000
$ tar –zcvf SPHW2_b02902000.tar.gz b02902000
$ rm –r b02902000
```

Please do NOT add executable files to the compressed file. Error in the submission file (such as files not in a directory named with your student ID, compiled binary not named organizer, judge, and player, or so on) may cause deduction of your credits.

VII. Punishments

● Plagiarism

Plagiarism is strictly prohibited.

● Late punishment

Your credits of this assignment will be deducted 5% for each day of delay submission. (That is, you will lose all your credits on this assignment after 20-day delaying!) Even though your credits will be deducted for delaying, a late submission will still be much better than absence.