The University of Queensland
School of Information Technology and Electrical Engineering
Semester Two, 2017
CSSE2310 / CSSE7231 - Assignment 3
Due: 11:10pm 29th September, 2017
Marks: 50
Weighting: 25% of your overall assignment mark (CSSE2310)
Revision 3.4

Introduction

In this assignment, you will write two types of programs `C99` programs to run and play a game (described later). The first type of program (players) will listen on their `stdin` for information about the game and give their actions to `stdout`. These types of program will send information about the state of the game to `stderr`. The second program (`2310express` — also known as the *hub*) will start a number of processes to be players and communicate with them via pipes. The hub will be responsible for running the game (sending information to players; processing their moves and determining the winner(s)). The hub will also ensure that information sent to `stderr` by the players, is discarded.

The idea is that you will produce a number of player programs (with different names) which implement different playing strategies. However all players will probably share a large amount of common code (with the other players *you* write).

Your programs must not create any files on disk not mentioned in this specification or in command line arguments. Your assignment submission must comply with the C style guide (version 2.0.3) available on the course blackboard area. This is an individual assignment. You should feel free to discuss aspects of C programming and the assignment specification with fellow students. You should not actively help (or seek help from) other students with the actual coding of your assignment solution. It is cheating to look at another student's code and it is cheating to allow your code to be seen or shared in printed or electronic form. You should note that all submitted code may be subject to automated checks for plagiarism and collusion. If we detect plagiarism or collusion, formal misconduct proceedings will be initiated against you. A likely penalty for a first offence would be a mark of 0 for the assignment. Don't risk it! If you're having trouble, seek help from a member of the teaching staff. Don't be tempted to copy another student's code. You should read and understand the statements on student misconduct in the course profile and on the school web-site: `http://www.itee.uq.edu.au/itee-student-misconduct-including-plagiarism`

As with Assignment 1, we will use the subversion (svn) system to deal with assignment submissions. Do not commit any code to your repository unless it is your own work or it was given to you by teaching staff. **If you have questions about this, please ask.**

While you are permitted to use sample code supplied by teaching staff this year in this course. Code supplied for other courses or other offerings of this course is off limits — it may be deemed to be without academic merit and removed from your program before testing.

The game

The game consists of a number of players attempting to rob a train (and possibly each other) armed with water pistols. The train has a number of carriages each with two levels. "loot" is distributed throughout the train (all pieces of loot have the same value). The game consists of 10 rounds. In each round players (in turn) choose from a set of possible actions. When all players have submitted their actions, the actions are carried out (possibly asking them for more information).

Possible actions:

- move horizontally — one carriage at a time. When the order is executed the player will be asked which direction to move in.
- move vertically — Since there only two levels no extra information is required.
- loot — pick up one piece of loot from the ground, no extra information is requried. However, all loot could have been taken between the time when the action is chosen and when it is executed.
- short — Force another player in the same location to drop one piece of loot. When this action is executed, the player is asked to choose a target player (if one is available). If there are no other players in the same location or the target player has no loot, nothing happens.
- long — A targeted player gains a hit from the water pistol. The player needs to choose an eligible target. If the player is on the lower level, then they can only target players in the lower level of one carriage to the left or right of their current position (note this does not include the carriage they are in). If the player is on the upper level, they can target players on the upper level any distance away provided there are no other players closer in that direction (again, you may not target players in the same carriage as you).
- dry out - player loses all their hits. If they have (or more) 4 hits, they must take this action. They will not be asked for a choice at all in this case. They can not choose to take this action at any other time.

The winners of the game are the ones with the maximal amount of loot at the end of the game.

Initial set up

All players will start on the lower level of the train, in carriage $n\%W$ (where n is the player number and W is the number of carriages in the train.

Loot will be distributed as follows: The total amount of loot $T = ((S \bmod 4) + 1) \times W$ (where S is the seed). The first piece of loot will be placed at $X = \lceil W/2 \rceil$, $Y = 0$. X indicates the carriage, $Y = 0$ indicates the lower level and $Y = 1$ is the upper level. For each successive piece of loot $X' = (X + (S \bmod 101)) \bmod W$, $Y' = (Y + (S \bmod 2)) \bmod 2$.

Invocation - players

The parameters to start a player are: the number of players (in total); the number of this particular player (starting at 0); the number of carriages in the train; the seed (an unsigned integer) used to determine loot distribution. For example:
`./player 3 1 5 234`

The maximum number of players permitted in a game will be 26.
After argument checking has completed successfully, players will output a single ! to `stdout`.

Invocation - hub

The parameters to start the hub are:

- The seed to generate loot distribution (an unsigned integer).
- The number of carriages in the train (there must be at least 3).
- The names of programs to run as players (one for each player). There must be at least two players.

For example: `./2310express 234 5 ./typeA ./typeA ./typeA`
Would start a game with 3 players (each running `./typeA`).

Message Representations

When the players and hub communicate, they must do so using messages encoded in the following way.

Messages from the hub

Encoded form	Parameters	Purpose
game_over		Inform player of end of game
round		A new round is starting
yourturn		Ask the player for their choice of action
orderedXV	X — the player who submitted the action. V — the action they chose ie one of v,l,h,s,$	Inform all players that an action has been submitted.
execute		All actions have been submitted. They will now be carried out.
h?		Choose to move left or right (if there is only one option you still need to reply)
s?		Choose a short range target
l?		Choose a long range target
hmoveXY	X — the player who moved. Y $\in \{-, +\}$ for left and right respectively	Inform all players about a movement
vmove$X$	X — the player who moved.	
longXT	X — the player who targeted. T — the player who was targeted	
shortXT		
looted$X$		
driedout$X$		Player has dried off

Whenever player's identity is communicated in messages, player 0 is indicated by 'A', 1 by 'B' and so on. When directions are to be communicated, negative numbers indicate leftwards, positive numbers rightwards.

Messages from players (to the hub)

Encoded form	Parameters	Purpose
playX	X — action chosen ie one of v,l,h,s,$	Tell the hub player's choice of action
sidewaysX	X — + or − to indicate direction	Tell hub which direction player wants to move.
target_shortX	X — ID of target or - if none available	
target_longX	X — ID of target or - if none available	

Output — hub

At the end of each round, the hub will output a summary like this:

```
A@(1,0): $=0 hits=0
B@(2,0): $=0 hits=0
Carriage 0: $=2 : $=0
Carriage 1: $=0 : $=0
```

The numbers in parentheses are the position of that player (carriage first). The number after $ for a player indicates the amount of loot that player has. The first $ in each carriage entry indicates the amount of loot in the lower level, the second indicates the amount of loot on the upper level.

When the game is over, the winner(s) will be output like this:

```
Winner(s):A,C
```

Output — players

After recieving each message, the players will send the following to standard error.

Message	**stderr**	**Params**
ordered	P ordered A	P — who did it, A — the choice of action
horizontal move	P moved to X/Y	X — carriage, Y — 0 for lower floor, 1 for upper floor
long range	P targets Q who has N hits **OR** P has no target	P — who shot, Q — who was shot, N — how many hits Q has now
short	P makes Q drop loot **OR** P tries to make Q drop loot they don't have **OR** P has no target	
loot	P picks up loot (they now have N) **OR** P tries to pick up loot but there isn't any	
driedout	P dries off	

Players

The following describes players which you will create. This does not mean that every player your hub is run with will be one of these three.

acrophobe

The preference order is:

1. Loot — if there is loot here
2. Move left — if not in the left most carriage
3. Move right

bandit

1. Loot — if there is loot here
2. Short — if there is another player here

- Choose the target with the highest ID

3. Move horizontally — if either left or right has more loot, move in that direction
4. Long — if there is a possible target (at time of action submission).
   - Choose target with the lowest ID
5. Move vertically

spoiler

1. If Short or Long was not used by this player last turn
   (a) Short — if there is another player here
   (b) Long — if there is a target
      - Choose target with highest ID
2. Move vertically — If there is a player below/above us.
3. Loot — If there is loot here.
4. Move horizontally
   (a) If one side has more players, move in that direction.
   (b) Move left.
   (c) Move right.

Exits

Exit status for player

All messages to be printed to `stderr`.

Condition	**Exit**	**Message**
Normal exit due to game over	0	
Wrong number of arguments	1	`Usage: player pcount myid width seed`
Invalid number of players	2	`Invalid player count`
Invalid player ID	3	`Invalid player ID`
Invalid number of carriages	4	`Invalid width`
Invalid seed	5	`Invalid seed`
Pipe closed before gameover message received or invalid message received	6	`Communication Error`

Exit status for the hub

All messages to be printed to `stderr`.

Condition	Exit	Message
Normal exit due to game over	0	
Wrong number of arguments	1	`Usage: hub seed width player player [player ...]`
Invalid commandline argument	2	`Bad argument`
Failure starting any of the player processes	3	`Bad start`
Player closed before gameover message was sent	4	`Client disconnected`
Protocol error by player	5	`Protocol error by client`
Illegal move by player	6	`Illegal move by client`
The hub received SIGINT	9	`SIGINT caught`

Note that both sides detect the loss of the other by a read failure. Write calls could also fail, but your program should ignore these failures and wait for the associated read failure. Such write failures must not cause your program to crash.

Shutting down the hub

Whether the hub shuts down in response to a SIGINT or of its own volition, it should follow the same procedure. It should ensure that all child processes have terminated. If they have not all terminated within 2 seconds of the hub sending the gameover message, then the hub should terminate them with SIGKILL. For each player (in order), do one of the following:

1. If the process terminated normally with exit status 0, Then don't print anything.
2. If the process terminated normally with a non-zero exit status, then print (to `stderr`):

   ```
   Player ? ended with status ?
   ```

3. If the process terminated due to a signal, then print (to `stderr`):

   ```
   Player ? shutdown after receiving signal ?
   ```

(Fill in ? with the appropriate values)

This reporting on children should only happen when:

1. all children started successfully (defined as when all children started and sent a `!` to their standard out).
2. the hub is shutting down normally (not as a result of SIGINT).

Compilation

Your code must compile (on a clean checkout) with the command:
`make`

Each individual file must compile with at least `-Wall -pedantic -std=gnu99`. You may of course use additional flags but you must not use them to try to disable or hide warnings. You must also not use pragmas to achieve the same goal. Your code must be compiled with the `gcc` compiler.

If the make command does not produce one or more of the required programs, then those programs will not be marked. If none of the required programs are produced, then you will receive 0 marks for functionality. Any code without academic merit will be removed from your program before compilation is attempted [This will be done even if it prevents the code from compiling]. If your code produces warnings (as opposed to errors), then you will lose style marks (see later).

Your solution must not invoke other programs apart from those listed in the command line arguments for the hub. Your solution must not use non-standard headers/libraries.

Submission

Submission must be made electronically by committing using subversion. In order to mark your assignment, the markers will check out `/trunk/ass3/` from your repository on `source.eait.uq.edu.au` [1]. Code checked in to any other part of your repository will not be marked.

The due date for this assignment is given on the front page of this specification. (Only the contents of the `trunk/ass3` directory at the deadline will be marked).

Test scripts will be provided to test the code on the trunk. Students are *strongly advised* to make use of this facility after committing.

Note: Any .h or .c files in your `trunk/ass3` directory will be marked for style *even if they are not linked by the makefile*. If you need help moving/removing files in svn, then ask. Consult the style guide for other restrictions.

You must submit a `Makefile` or we will not be able to compile your assignment. Remember that your assignment will be marked electronically and strict adherance to the specification is critical.

Marks

Marks will be awarded for both functionality and style.

Functionality (42 marks)

Provided that your code compiles (see above), you will earn functionality marks based on the number of features your program correctly implements, as outlined below. Partial marks may be awarded for partially meeting the functionality requirements. Not all features are of equal difficulty. If your program does not allow a feature to be tested then you will receive 0 marks for that feature, even if you claim to have implemented it. For example, if your program can never open a file, we can not determine if your program would have loaded input from it. The markers will make no alterations to your code (other than to remove code without academic merit). Your programs should not crash or lock up/loop indefinitely. Your programs should not run for unreasonably long times.

- For *acrophobe* player

[1] That is, `https://source.eait.uq.edu.au/svn/csse2310-$USER/trunk/ass3`

  - argument checking (2 marks)
  - correctly handles early loss of hub and ending messages (2 marks)
  - respond correctly to their first turn (2 marks)
  - (Player) Correctly handle complete game (6 marks)
  - (Player) Detect invalid messages (2 marks)
- For hub
  - argument checking (2 marks)
  - Detect failure to start players (2 marks)
  - Correctly handle players which close early (3 marks)
  - Correctly handle 2 player games with *acrophobe* players (4 marks)
- play complete games with 3 *bandit* players. (2 marks)
- play complete games with 4 *spoiler* players. (4 marks)
- play complete games with a mixture of player types. (11 marks)

Style (8 marks)

Style marks will be calculated as follows:
Let A be the number of style violations detected by simpatico plus the number of build warnings. Let H be the number of style violations detected by human markers. Let F be the functionality mark for your assignment.

- If $A > 10$, then your style mark will be zero and M will not be calculated.
- Otherwise, let $M_A = 4 \times 0.8^A$ and $M_H = M_A - 0.5 \times H$ your style mark S will be $M_A + \max\{0, M_H\}$.

Your total mark for the assignment will be $F + \min\{F, S\}$.

Late Penalties

Late penalties will apply as outlined in the course profile.

Specification Updates

It is possible that this specification contains errors or inconsistencies or missing information. It is possible that clarifications will be issued via the course website. Any such clarifications posted 5 days (120 hours) or more before the due date will form part of the assignment specification. If you find any inconsistencies or omissions, please notify the teaching staff.

Test Data

Test data and scripts for this assignment will be made available. (`testa3.sh`, `reptesta3.sh`) The idea is to help clarify some areas of the specification and to provide a basic sanity check of code which you have committed. *They are not guaranteed to check all possible problems nor are they guaranteed to resemble the tests which will be used to mark your assignments.* Testing that your assignment complies with this specification is still *your* responsibility.

Notes and Addenda:

1. This assignment implicitly tests your ability to recognise repeated operations/steps and move them into functions to be reused. If you rewrite everything each time it is used, then writing and debugging your code will take much longer.
2. Start early.
3. Write simple programs to try out fork(), exec() and pipe().
4. Be sure to test on moss.
5. You should not assume that system calls always succeed.
6. You are not permitted to use any of the following functions in this assignment.
   - `system()`
   - `popen()`
   - `prctl()`
   - `setjmp()`
7. You may not use any `#pragma` in this assignment.
8. Neither program should assume that the other will be well behaved. That is, if the hub sends a valid message, you may believe it. However, if the hub sends an invalid message, your player should not crash.
9. You will need to do something with SIGPIPE.
10. Valid messages contain no leading, trailing or embedded spaces. Messages must be exactly the correct length.
11. Make a separate player to test the hub against badly behaved players rather than breaking an existing one.
12. You should only report on the exit statuses of the players if all players started successfully.
13. `struct`s and `enum`s are your friends use them.
14. Just because communication is required to be in a particular format, does not mean that you must store information internally in that form. Instead, convert from the external format to the internal format as soon as possible and convert to the external form as late as possible.
15. There will be a number of different communication channels in a complete system. Make sure that you can monitor any of the channels.

Updates

3.3 → 3.4

1. Modified due date so time travel is no longer required.
2. Added text to the dryout action to indicate that players can not choose this action and that if it is required, they are not asked to make a choice.

3. Add clarifying text to “Initial set up”.
4. Explained the output format for players.
5. Added parenthetical that seed should be an unsigned integer.
6. The maximum number of carriages must fit in an unsigned int.
7. Player programs must have the names given in their description when compiled.
8. Gave encoding for left/right movement.
9. Added encoding for playX message.
10. Bolded table header