OPERATING SYSTEMS EXERCISE 3

Aufgabenstellung – 2048

In dieser Aufgabe werden Sie eine Terminal Version des Spiels „2048“ (auch bekannt als „1024“ und „Threes“) implementieren.

Das Spielfeld besteht aus 16 Feldern, die in einem 4x4 Quadrat angeordnet sind. Jedes davon kann entweder leer sein oder einen Wert aus der Menge $\{2^i | i \in \mathbb{N}^+\}$ enthalten.

Benachbarte Felder, welche den selben Wert beinhalten, können kombiniert werden und verschmelzen zu einem Feld mit dem Wert der Summe beider vorherigen Felder. Zum Beispiel ergibt eine Kombination von 2 Feldern mit Wert `32` ein Feld mit Wert `64`.

Ziel des Spiels ist, Felder geschickt zu kombinieren, um Felder mit möglichst hohen Werten entstehen zu lassen. Ein Spiel ist gewonnen sobald ein Feld mit Wert `2048` entsteht (bzw. der Wert, mit dem der Server als Optionsargument gestartet wurde, siehe Abschnitt). Falls das Spielfeld voll ist und keine weiteren Kombinationsmöglichkeiten bestehen, hat der Spieler verloren.

In einer Spielrunde werden folgende Schritte durchlaufen:

1. Ein freies Feld wird zufällig ausgewählt und mit einem neuen Wert befüllt. Dabei soll der Wert `2` mit einer Wahrscheinlichkeit von 0.75 oder der Wert `4` mit einer Wahrscheinlichkeit von 0.25 erzeugt werden.
2. Der Spieler gibt eine Richtung an, in welche alle Felder mit Werten auf dem Spielfeld verschoben werden.
3. Daraufhin wird der angegebene Zug ausgeführt. Im folgenden steht r für die vom Spieler angegebene Richtung. Angefangen auf der Seite r wird nun für jedes Feld f folgende Logik angewandt:
   (a) Solange das Nachbarfeld von f in Richtung r leer ist, verschiebe f dorthin.
   (b) Falls das Nachbarfeld von f in Richtung r den selben Wert hat wie f, entferne f und verdopple den Wert vom Nachbarfeld.
4. Falls das Spielfeld durch Schritt 3 unverändert geblieben ist, setzen Sie mit Schritt 2 fort; andernfalls beginnt eine neue Spielrunde.

Bei Fragen zu den Spielregeln und für weitere Inspiration, orientieren Sie sich am einfachsten an `http://gabrielecirulli.github.io/2048/`.

Anleitung

Das Spiel soll als Client/Server Programm realisiert werden, wobei ein Server beliebig viele Clients (das heißt auch beliebig viele gleichzeitige Spiele) bedient. Kommunikation zwischen Clients und Server soll per Shared Memory erfolgen.

Server

```
USAGE: 2048-server [-p power_of_two]
    -p:     Play until 2^power_of_two is reached (default: 11)
```

Über die Option `-p` kann angegeben werden, wann das Spiel als gewonnen gilt, also bei welcher Zweierpotenz, die als Wert auf einem Feld erreicht wird. Per default beträgt dieser Wert $2^{11} = 2048$.

Spielelogik findet ausnahmslos am Server statt, während Clients lediglich für Darstellung und Benutzereingaben verantwortlich sind. Eine Nachricht zum Client soll also das aktuelle Spielfeld und Status Codes (z.B. `ST_WON`, `ST_LOST`, `ST_NOSUCHGAME`) enthalten, während eine Nachricht zum Server nur Befehle (z.B. `CMD_LEFT`, `CMD_QUIT`) enthält.

Der Server soll eine unbegrenzte Anzahl an gleichzeitigen Spielen verwalten können. Sobald ein Client ein Request für ein neues Spiel sendet, sollen lokal benötigte Ressourcen angelegt werden. Bei Beendigung eines Spiels müssen Sie diese natürlich wieder freigeben.

Clients kommunizieren mit dem Server über Shared Memory. Zugriffe darauf sollen mit Semaphoren synchronisiert werden. Stellen Sie sicher, dass nach einem Request von Client C_i die Antwort unbedingt nur C_i (und nicht einen anderen Client $C_j, j \neq i$) erreicht.

Jedem Spiel wird vom Server eine ID zugeteilt, und Clients können sich entweder zu bestehenden Spielen verbinden oder ein neues erstellen. Insbesondere werden Spiele am Server nur gelöscht falls sie entweder vom Client explizit beendet werden oder verloren sind. Ein Client soll sich also vom Spiel trennen und später wieder zum selben Spiel verbinden können. Sie dürfen annehmen, dass sich zu einem Spiel nie mehr als ein Client gleichzeitig verbindet.

Falls der Server ein `SIGTERM` oder `SIGINT` Signal erhält, sollen alle angelegten Ressourcen (lokal angelegter Speicher, Shared Memory, Semaphoren, etc) korrekt freigegeben und terminiert werden.

Client

```
USAGE: 2048-client [-n | -i <id>]
    -n:     Start a new game
    -i:     Connect to existing game with the given id
```

Der Client ist für Darstellung der Spielfelds und Behandlung von Benutzereingaben zuständig.

Die Gestaltung der Oberfläche ist nicht weiter spezifiziert, außer dass mindestens das Spielfeld, die ID des aktuellen Spiels, und eine kurze Anleitung mit den möglichen Aktionen angezeigt werden müssen.

Mögliche Aktionen sind:

- *Links/Rechts/Oben/Unten*: Angabe der Richtung eines Spielzugs. Taste `a` steht für links, `d` für rechts, `w` für oben, `s` für unten.
- *Spiel Löschen*: Das Spiel wird aufgegeben, Ressourcen werden am Server gelöscht, der Client terminiert.
- *Verbindung Trennen*: Der Client terminiert ohne Spielressourcen am Server zu löschen.

Der Client soll zusätzlich ordnungsgemäß terminieren sobald das Spiel gewonnen oder verloren ist, oder der Server (d.h. das Shared Memory oder die Semaphore) nicht mehr erreichbar ist.

Coding Rules and Guidelines

Your score depends upon the compliance of your submission to the presented guidelines and rules. Violations result in deductions of points. Hence, before submitting your solution, go through the following list and check if your program complies.

Rules

Compliance with these rules is essential to get any points for your submission. A violation of any of the following rules results in 0 points for your submission.

1. All source files of your program(s) must compile via

   ```
   $ gcc -std=c99 -pedantic -Wall -D_DEFAULT_SOURCE -D_BSD_SOURCE -D_SVID_SOURCE
                                  -D_POSIX_C_SOURCE=200809L -g -c filename.c
   ```

   without *errors* and your program(s) must link without *errors*. The compilation flags must be used in the Makefile. The feature test macros must not be bypassed (i.e., by undefining these macros or adding some in the C source code).

2. The functionality of the program(s) must conform to the assignment. The program(s) shall operate according to the specification/assignment given the test cases in the respective assignment.

General Guidelines

Violation of following guidelines leads to a deduction of points.

1. All source files of your program(s) must compile with

   ```
   $ gcc -std=c99 -pedantic -Wall -D_DEFAULT_SOURCE -D_BSD_SOURCE -D_SVID_SOURCE
                                  -D_POSIX_C_SOURCE=200809L -g -c filename.c
   ```

   without *warnings and info messages* and your program(s) must link without warnings.

2. There must be a Makefile implementing the targets: `all` to build the program(s) (i.e. generate executables) from the sources (this must be the first target in the Makefile); `clean` to delete all files that can be built from your sources with the Makefile.

3. The program shall operate according to the specification/assignment without major issues (e.g., segmentation fault, memory corruption).

4. Arguments have to be parsed according to UNIX conventions (we strongly encourage the use of `getopt(3)`). The program has to conform to the given synopsis/usage in the assignment. If the synopsis is violated (e.g., unspecified options or too many arguments), the program has to terminate with the usage message containing the program name and the correct calling syntax. Argument handling should also be implemented for programs without arguments.

5. Correct (=normal) termination, including a cleanup of resources.

6. Upon success the program has to terminate with exit code 0, in case of errors with an exit code greater than 0. We recommend to use the macros `EXIT_SUCCESS` and `EXIT_FAILURE` (defined in `stdlib.h`) to enable portability of the program.

7. If a function indicates an error with its return value, it *should* be checked in general. If the subsequent code depends on the successful execution of a function (e.g. resource allocation), then the return value *must* be checked.

8. Functions that do not take any parameters have to be declared with `void` in the signature, e.g., int get_random_int(void);.

9. Procedures (i.e., functions that do not return a value) have to be declared as `void`.

10. Error messages shall be written to `stderr` and should contain the program name `argv[0]`.

11. It is forbidden to use the functions: `gets`, `scanf`, `fscanf`, `atoi` and `atol` to avoid crashes due to invalid inputs.

FORBIDDEN	USE INSTEAD
gets	fgets
scanf	fgets, sscanf
fscanf	fgets, sscanf
atoi	strtol
atol	strtol

12. Documenation is mandatory. Format the documentation in Doxygen style (see Wiki and Doxygen's intro).

13. Write meaningful comments. For example, meaningful comments describe the algorithm, or why a particular solution has been chosen, if there seems to be an easier solution at a first glance. Avoid comments that just repeat the code itself
(e.g., i = i + 1; /* i is incremented by one */).

14. The documentation of a module must include: name of the module, name and student id of the author (`@author` tag), purpose of the module (`@brief`, `@details` tags) and creation date of the module (`@date` tag).

    Also the Makefile has to include a header, with author and program name at least.

15. Each function shall be documented either before the declaration or the implementation. It should include purpose (`@brief`, `@details` tags), description of parameters and return value (`@param`, `@return` tags) and description of global variables the function uses (`@details` tag).

    You should also document `static` functions (see `EXTRACT_STATIC` in the file `Doxyfile`). Document visible/exported functions in the header file and local (`static`) functions in the C file. Document variables, constants and types (especially `structs`) too.

16. Documentation, names of variables and constants shall be in English.

17. Internal functions shall be marked with the `static` qualifier and are not allowed to be exported (e.g., in a header file). Only functions that are used by other modules shall be declared in the header file.

18. All exercises shall be solved with functions of the C standard library. If a required function is not available in the standard library, you can use other (external) functions too. Avoid reinventing the wheel (e.g., re-implementation of `strcmp`).

19. Name of constants shall be written in upper case, names of variables in lower case (maybe with fist letter capital).

20. Use meaningful variable and constant names (e.g., also semaphores and shared memories).

21. Avoid using global variables as far as possible.

22. All boundaries shall be defined as constants (macros). Avoid arbitrary boundaries. If boundaries are necessary, treat its crossing.

23. Avoid side effects with `&&` and `||`, e.g., write if (b != 0) c = a/b; instead of if (b != 0 && c = a/b).

24. Each `switch` block must contain a `default` case. If the case is not reachable, write `assert(0)` to this case (defensive programming).

25. Logical values shall be treated with logical operators, numerical values with arithmetic operators (e.g., test 2 strings for equality by strcmp (...) == 0 instead of !strcmp (...)).

26. Indent your source code consistently (there are tools for that purpose, e.g., `indent`).

27. Avoid tricky arithmetic statements. Programs are written once, but read more times. Your program is not better if it is shorter!

28. For all I/O operations (read/write from/to `stdin`, `stdout`, files, sockets, pipes, etc.) use *either* standard I/O functions (`fdopen(3)`, `fopen(3)`, `fgets(3)`, etc.) *or* POSIX functions (`open(2)`, `read(2)`, `write(2)`, etc.). Remember, standard I/O functions are buffered. Mixing standard I/O functions and POSIX functions to access a common file descriptor can lead to undefined behaviour and is therefore forbidden.

29. If asked in the assignment, you must implement signal handling (`SIGINT`, `SIGTERM`). You must only use *async-signal-safe* functions in your signal handlers.

30. Close files, free dynamically allocated memory, and remove resources after usage.

31. Don't waste resources due to inconvenient programming. Header files shall not include implementation parts (exception: macros).

Exercise 3 Guidelines

Violation of following guidelines leads to a deduction of points in exercise 3.

1. It is forbidden to use the deprecated functions `gethostbyname`, `gethostbyaddr`, `gethostbyname2`, `gethostbyname_r`, `gethostbyname2_r` or any other function of that family documented on the man page `gethostbyname(3)`. Use `getaddrinfo(3)` instead.