DV1457 - Programming in Unix Environments

Assignment 1330

(2.5 Credits) ETCS, A-F

October 16, 2024

Deadline 1	October 25th, 2024 18:00
Deadline 2	November 22nd, 2024 18:00
Deadline 3	January 10th, 2024 18:00
Submission	ZIP file uploaded to Canvas

---

1 Description

Your task is to implement a playable snake game in x86-64 assembly in a Linux environment (Fedora). This assignment is meant to assess your understanding of assembly programming, the use of libraries, and the ability to develop a complete program in assembly.

2 Instructions

Before you start working on the assignment, please make sure you have the following tools installed on your computer:

- `GNU as` assembler.

- `gcc` compiler.
- `make` tool.
- `ncurses` library. This library is used to create text-based user interfaces. We use this to create the game board and to read user input.You can install this library using the following command in Fedora:
  `sudo dnf install ncurse-devel`.

2.1 Game Requirements

The requirements for the game are as follows:

- The snake game must be playable.
  - You must support a configurable number of apples (more than 1) that will be specified as a command-line argument when running the game.
  - The snake should grow when eating an apple.
  - The snake should be controlled by the arrow keys. If no arrow key is pressed, the snake should continue to move in the direction it was last moving. It cannot move backwards.
  - The snake should die if it hits itself.
  - If the snake hits the end of the field (above, below, right, or left) it should either:
    * Die and the game should end.
    * Appear on the other side of the field.
  - Apples should be placed at random positions on the screen and when an apple is eaten, a new apple should reapper. You do not need to handle the situation when the apple appears on the snake (i.e. only the head eats).
- The size of the playfield (board) is not specified, but should be possible to display on the screen.
- The snake should start at the middle of the board.
- The game must be implemented *completely* in x86-64 assembly.
- The game must be compilable with *GNU as* and linked with *GNU GCC*, i.e. you should use AT&T syntax-style assembly.
- The soulce code should be well-commented but *not* have comments on every line.
- It should be possible to compile the game using the provided `Makefile` (You can modify the `Makefile` if needed).

- It should be possible to start the game by running the compiled binary with length of the snake and the number of apples as command-line argument (e.g. `./snake 5 2` to start the game with a snake of length 5 and 2 apples).
- It should be possible to start the game from either assembly or C. It must follow the C calling convention.

You are allowed to use tricks, hardcoding values, or any other method to simplify the implementation of the game.

The game must be implemented in x86-64 assembly language using the AT&T syntax. You are not allowed to use any other programming language to implement the game. Meaning, you are **not** allowed to generate the assembly code from C code.

2.1.1 Non-requirements

The following features should **not** be implemented in the game:

- A high score system.
- A pause feature.
- A start screen.
- A game over screen.
- Multiple levels, multiplayr options, or any other advanced features.

2.2 Extra Features

- A speed increase feature, where the speed of the snake increases every time the snake eats an apple.
- Include an implementation of a timer that will end the game after a certain amount of time. The timer is reset every time the snake eats an apple. Tips is to use the `usleep` function inbetween each game loop iteration and use the number of iterations since the game started as the time.
- Limit the apples to only appear in a space where neither the snake nor any other apple is present.
- Implement the helper functions in `helpers.c` in assembly (still calling the `ncurses` functions). Modify the `Makefile` to compile the helper functions in assembly instead of C.

3 Provided Functionality

You are provided with a number of C functions which you can call from your game implementation. The functions perform initialising, getting keyboard input and printing characters on the screen.

3.1 board_init

```
void board_init();
```

This function sets up the ncurses library, creates a window for the game, initialises the randomiser, etc. You *must* call this function before any other provided function are called.

3.2 game_exit

```
void game_exit();
```

This function cleans up the ncurses library and closes the window and exits to the shell. It should be called when the game is over (i.e. the snake dies).

3.3 board_get_key

```
int board_get_key();
```

This function returns the key pressed by the user. If no key is pressed, the function returns -1. The following keys are defined in Table 1.

Table 1: Key Codes

-1	There was no keyboard input.
‘a‘...‘z‘	A letter key was pressed.
258	Down arrow key was pressed.
259	Up arrow key was pressed.
260	Left arrow key was pressed.
261	Right arrow key was pressed.

3.4 board_put_char

```
void board_put_char(int x, int y, int ch);
```

This function prints a character `ch` at position `x` and `y` on the screen. The top-left corner of the screen is at position (0, 0). The character `ch` is a normal ASCII character code.

3.5 board_put_str

```
void board_put_str(int x, int y, const char *str);
```

This function prints a string `str` at position `x` and `y` on the screen. The top-left corner of the screen is at position (0, 0). This can be useful for printing debug information when developing the game (I recommend also to make use of the `sprintf` function in *libc* to format the string with the debug information).

4 Useful *libc* functions

There are also a few useful functions in *libc* that you might want to use when developing the game. The most obvious ones are `int rand()` and `void usleep(unsigned long usec)`. See the man pages of `rand` and `usleep` for information on how to use them.

5 Files

You are provided with a number of files for the game. The `helpers.c` file which contains the implementation of the functions above, `main.c` file which is a small C program which calls your game. A Makefile is also provided which should be easy to customise to your needs. You compile the game by running `make`. You are expected to implement `snake.asm` and `start.asm` by yourselves.

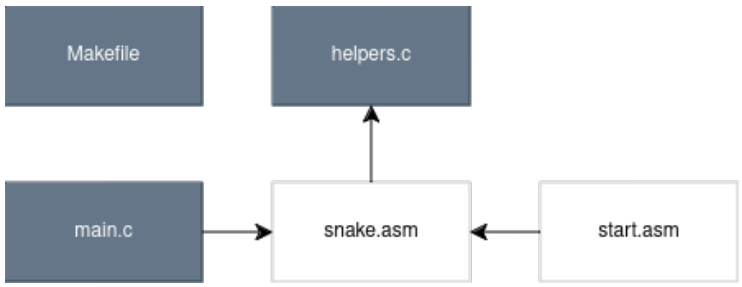

Figure 1: Files in the project

6 Interface

Your game must implement the following interface:

`void start_game(int len, int n_apples)`: Start the snake game with an initial snake length of `len` and `n_apples` apples on the screen. Note that this function will be called from C, so that you have to comply with the C calling convention.

`void _start(void)`: The `_start` symbol, i.e. the entry point of the program. This function should in turn call the `start_game` function with reasonable parameters.

You should also note that these two functions (i.e., global lables in assembly) must be implemented in different files (i.e., `snake.asm` and `start.asm`) since the game should be possible to start from C. This is because the C-program will be linked with `crt0.s` which already contains the *_start*-symbol (and would therefore cause a collision if linked with the file containing your *_start*-symbol).

7 Requirements and Assessment

The development of this assignment is done in groups of 3 students and, despite the desired grade, every submission must:

- Be implemented using the x86-64 assembly language using the AT&T syntax.
- Present the solution in a filed named `snake.asm`.
- Comply with all specifications described in Section 2.
- Include a .txt file with the students' names, the desired grade, which features are supported by the game (solution), and a brief description of the approach used to implement it.
- Include your game code (assembly files), the provided helper.c file, and the .txt file with students' information, within in a single .zip file.

Each implemented supported feature will award students with one point (see Section 2.2 for the features). The assignment is graded with the following scale: E, D, C, B, and A. The grade for this assignment is determined by the final number of points achieved by the students, based on the grading criteria shown in Table 2.

Table 2: Grading criteria for Assignment 1330

Points	**Grade**
1	E
2	D
3	C
4	B
5	A

If the result produced by a feature is incorrect or is incomplete, no point will be awarded for it. Also, in accordance with BTH regulations, submissions graded with Fx can only be complemented up to E grade.

Submissions will only be accepted through Canvas and within deadline limits. Lastly, plagiarism is prohibited and any found case of plagiarism will have disciplinary actions taken against the students involved.

Developed by *Simon Kågström* and *Håkan Grahn*, 2010.
Modified by *Christoffer Åleskog*, 2024.

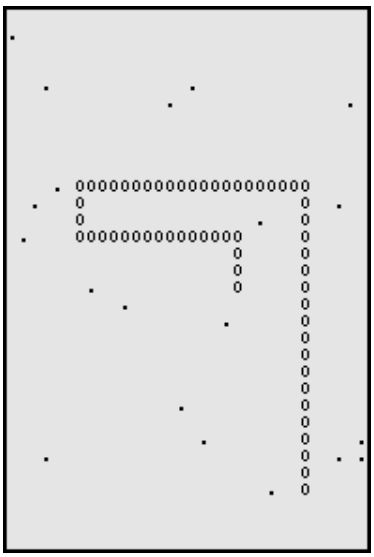

Figure 2: A typical Snake game