Operating Systems
Jacobs University Bremen
Dr. Jürgen Schönwälder

Module: CO-562
Date: 2021-10-28
Due: 2021-11-04

OS 2021 Problem Sheet #8

Problem 8.1: *math quiz using an online oracle* (3+2+3+2 = 10 points)

Write a program that writes random mathematical questions to the standard output and then reads answers from the standard input until the correct answer has been given or the number of attempts to provide an answer has been exceeded. Answers are restricted to the range $[1..100]$. The game repeats until the the user ends the standard input (by pressing CTRL-D on Linux/MacOS) or interrupts the game by pressing CTRL-C. The game then shows the final score.

```
$ ./quiz
Answer questions with numbers in the range [1..100].
You score points for each correctly answered question.
If you need multiple attempts to answer a question, the
points you score for a correct answer go down.

Q1: What is the number of inches in a yard?
8 pt> 36
Congratulation, your answer 36 is correct.
Your total score is 8/8 points.

Q2: What is the smallest number that can be written as the sum of of 2 squares in 2 ways?
8 pt> 42
Too small, try again.
4 pt> 70
Too large, try again.
2 pt> 55
Too large, try again.
1 pt> 50
Congratulation, your answer 50 is correct.
Your total score is 9/16 points.

Q3: What is the largest cube in the Fibonacci sequence?
8 pt> 1
Too small, try again.
4 pt> 2
Too small, try again.
2 pt> 3
Too small, try again.
1 pt> 5
Too small, the correct answer was 8.
Your total score is 9/24 points.

Q4: What is a composite number, its proper divisors being 1, 2, 3, 6 and 9?
8 pt> ^D
Your total score is 9/24 points.

Thanks for playing today.
Your final score is 9/24 points.
```

Your program obtains the questions from a web service. Since you do not want to implement a network protocol yourself (yet), you simply run a tool like `curl` to fetch questions and you parse the output generated by curl. An example invocation of `curl` in your shell may look like this:

```
$ curl -s 'http://numbersapi.com/random/math?min=1&max=100&fragment&json'
```

```
{
 "text": "a composite number, its proper divisors being 1, 2, 3, 6 and 9",
 "number": 18,
 "found": true,
 "type": "math"
}
```

The URL requests a math question (there are different categories) with the answer being in the range 1 to 100. If the request is successful, the program returns a JSON document, which includes the question text, the correct answer, the indication that an answer was found, and the category of the lookup. (In addition to math, you can ask for trivia questions.)

Further information how to structure your implementation is provided in the following header file:

```c
/*
 * quiz/quiz.h --
 */

#ifndef QUIZ_H
#define QUIZ_H

typedef struct {
    unsigned n;          /* current question number (starting at 1) */
    unsigned score;      /* current total score */
    unsigned max;        /* possible max score */
    char *text;          /* next question (dynamically allocated) */
    int number;          /* next expected correct answer */
} quiz_t;

/*
 * Fetch the content of the given url by running 'curl' as a child
 * process and reading the response from a pipe. The response is
 * returned as a malloc'ed string, or NULL if there was an error.
 *
 * Implement the fetch function in a fetch.c module.
 */

extern char* fetch(char *url);

/*
 * Parse a JSON encoded question add fill the next question into the
 * quiz_t. Use a JSON parsing library (e.g., jansson or json-c). The
 * function returns 0 or -1 if there was a parsing error.
 *
 * Implement the parse function in a parse.c module.
 */

extern int parse(char *json, quiz_t *quiz);

/*
 * Play one round of the quiz game by first fetching and parsing a
 * quiz question and then interacting with the user. The function
 * returns 0 or -1 if there was an error.
 *
 * Implement the play function in a play.c module.
 */

extern int play(quiz_t *quiz);

#endif
```

a) Implement a function `char* fetch(char *url)` in the file `fetch.c` that opens a pipe to `curl` running as a child process. The function returns the response obtained by `curl`.

   Do not use `popen()` or `system()`. Do not use the `libcurl` C API.

b) Implement a function `int parse(char *json, quiz_t *quiz)` in the file `parse.c` that parses the JSON input and adds the next question and the expected answer to the quiz data structure. Consider to use a JSON parsing library such as `jansson` or `json-c`.

c) Implement a function `int play(quiz_t *quiz)` in the file `play.c` that plays one iteration of the game.

d) Implement a program in the file `quiz.c` that plays the quiz game in a loop until either the user did end the standard input or the user pressed CTRL-C to interrupt the program.

Make sure that your program handles error situations and is robust regarding malicious inputs. It is recommended that you test your program using `valgrind`, a tool that can identify a number of programming errors.

Solution:

```c
/*
 * quiz/fetch.c --
 */

#define _POSIX_C_SOURCE 200809L

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <assert.h>

#include "quiz.h"

#define CURL    "/usr/bin/curl"

static char* readall(int fd)
{
    char *result = NULL;
    ssize_t result_len = 0;
    char buf[256];
    ssize_t len;

    while ((len = read(fd, buf, sizeof(buf))) > 0) {
        result_len = result_len + len;
        result = realloc(result, result_len + 1);
        if (! result) {
            perror("realloc");
            exit(EXIT_FAILURE);
        }
        if (result_len == len) {
            result[0] = 0;
        }
        strncat(result, buf, len);
    }

    return result;
}

char* fetch(char *url)
{
    int fds[2];
    pid_t pid;
    char *cmd[] = { CURL, "-s", NULL, NULL };
    char *result = NULL;

    assert(url);
    cmd[2] = url;

    if (pipe(fds) == -1) {
        perror("pipe");
        exit(EXIT_FAILURE);
    }

    pid = fork();
    if (pid == -1) {
        (void) close(fds[0]);
        (void) close(fds[1]);
        exit(EXIT_FAILURE);
    }

    if (pid == 0) {
        (void) close(fds[0]);
        dup2(fds[1], STDOUT_FILENO);
        (void) close(fds[1]);
        execv(CURL, cmd);
        perror("execv");
        exit(EXIT_FAILURE);
    } else {
        close(fds[1]);
        result = readall(fds[0]);
        close(fds[0]);
    }

    return result;
}


/*
 * quiz/parse.c --
 */

#include <string.h>
#include <assert.h>
#include <jansson.h>

#include "quiz.h"

int parse(char *json, quiz_t *quiz)
{
    json_error_t error;
    json_t *root, *text, *number;

    assert(json && question);

    root = json_loads(json, 0, &error);
    if (! root) {
        fprintf(stderr, "json parse error: on line %d: %s\n",
                error.line, error.text);
        fprintf(stderr, "json: '%s'\n", json);
        return -1;
    }
    if(! json_is_object(root))
    {
        fprintf(stderr, "json parse error: root is not an object\n");
        json_decref(root);
        return -1;
    }
    text = json_object_get(root, "text");
    if (! json_is_string(text)) {
        fprintf(stderr, "json parse error: text is not a string\n");
        json_decref(root);
        exit(EXIT_FAILURE);
    }
    number = json_object_get(root, "number");
    if (! json_is_integer(number)) {
        fprintf(stderr, "json parse error: number is not an integer\n");
        json_decref(root);
        exit(EXIT_FAILURE);
    }

    quiz->text = strdup(json_string_value(text));
    quiz->number = json_integer_value(number);
    json_decref(root);
    return 0;
}


/*
 * quiz/play.c --
 */

#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

#include "quiz.h"

#define URL     "http://numbersapi.com/random/math?min=1&max=100&fragment&json"

static long getanswer()
{
    char *line = NULL;
    size_t len = 0;
    long num = 0;

    if (getline(&line, &len, stdin) == -1) {
        if (! feof(stdin)) {
            perror("getline");
            exit(EXIT_FAILURE);
        }
    }
    if (line) {
        if (! feof(stdin)) {
            num = strtol(line, NULL, 10);
        }
        free(line);
    }
    return num;
}

static void interact(quiz_t *quiz)
{
    int num;
    unsigned points;
    const unsigned max = 8;

    quiz->n++;
```

```c
    quiz->max = quiz->n * max;
    printf("\nQ%d: What is %s?\n", quiz->n, quiz->text);
    for (points = max; points && !feof(stdin); points /= 2) {
        printf("%d pt> ", points);
        fflush(stdout);

        num = getanswer();
        if (feof(stdin)) {
            puts("");
            quiz->n--;
            quiz->max = quiz->n * max;
            break;
        }
        if (num <= 0 || num > 100) {
            printf("Impossible answer.\n");
            continue;
        }

        if (num == quiz->number) {
            printf("Congratulation, your answer %d is correct.\n", num);
            quiz->score += points;
            break;
        }
        if (num < quiz->number) {
            printf("Too small, ");
        } else {
            printf("Too large, ");
        }
        if (points > 1) {
            printf("try again.\n");
        } else {
            printf("the correct answer was %d.\n", quiz->number);
        }
    }

    printf("Your current score is %d/%d points.\n", quiz->score, quiz->max);
}

int play(quiz_t *quiz)
{
    char *json;

    assert(quiz);

    json = fetch(URL);
    if (! json) {
        fprintf(stderr, "failed to fetch question\n");
        return -1;
    }
    if (parse(json, quiz) == -1) {
        return -1;
    }
    interact(quiz);
    free(quiz->text);
    return 0;
}


/*
 * quiz/quiz.c --
 */

#include <stdio.h>
```

```c
#include <stdlib.h>
#include <signal.h>

#include "quiz.h"

static quiz_t quiz;

static void bye(quiz_t *quiz)
{
    printf("\nThanks for playing today.\nYour final score is %d/%d points.\n",
           quiz->score, quiz->max);
}

static void handler(int signum)
{
    if (signum == SIGINT) {
        bye(&quiz);
        (void) signal(SIGINT, SIG_DFL);
        raise(SIGINT);
    }
}

int main(void)
{
    struct sigaction sa;

    sa.sa_handler = handler;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = SA_RESTART | SA_RESETHAND;
    if (sigaction(SIGINT, &sa, NULL) == -1) {
        perror("sigaction");
        exit(EXIT_FAILURE);
    }

    sa.sa_handler = SIG_DFL;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = SA_NOCLDWAIT;
    if (sigaction(SIGCHLD, &sa, NULL) == -1) {
        perror("sigaction");
        exit(EXIT_FAILURE);
    }

    puts("Answer questions with numbers in the range [1..100].\n"
         "You score points for each correctly answered question.\n"
         "If you need multiple attempts to answer a question, the\n"
         "points you score for a correct answer go down.");

    while (! feof(stdin)) {
        if (play(&quiz) == -1) {
            return EXIT_FAILURE;
        }
    }

    bye(&quiz);
    return EXIT_SUCCESS;
}
```

Marking:

a)
- *1pt for creating a pipe to curl running as a child process*
- *1pt for redirecting the curl output to the pipe and reading it*
- *1pt for properly allocating memory for the result read from the pipe*

b) - *1pt for extracting the text field (reasonably robust)*
- *1pt for extracting the number field (reasonably robust)*

c) - *1pt for parsing user input properly*
- *1pt for the correct play logic*
- *1pt for handling EOF situations properly*

d) - *1pt for ending the game properly upon receiving a SIGINT*
- *1pt for an overall correct main loop*