Operating Systems Module: CO-562
Constructor University Date: 2026-11-06
Dr. Jürgen Schönwälder Due: 2026-11-13

Problem Sheet #10

Problem 10.1: *word guessing game (blocking, forking, threads)* (1+2+2+2+2+1 = 10 points)

The word guessing game invites clients to guess words missing in dynamically created (almost random) phrases. The game is implemented in C as a standalone server. After starting up and binding to a port, optionally specified on the command line, the server accepts incoming connections from clients and it greets them by sending generic greeting messages (prefixed by `M:`). Afterwards, the server creates a word guess challenge and it sends the challenge to the client as a challenge message (prefixed by `C:`). The client then sends a response message (prefixed by `R:` in an attempt to guess the word. If the client guessed the word correctly, the server sends an OK message (prefixed by `O:`). If the guessed word was wrong, the server sends a fail message (prefixed by `F:`). If the client does not comply to the protocol, the server may send generic messages (prefixed by `M:`). The client can at any time send a quit message (prefixed by `Q:`) to end the game.

An example exchange is shown below. The messages send by the client are the messages starting with `R:` and `Q:` , all other messages are generated by the server.

```
$ nc localhost 1234
M: Guess the missing ____!
M: Send your guess in the form 'R: word\r\n'.
C: Is this ______ happening?
R: really
O: Congratulation - challenge passed!
C: You ___ standing on my toes.
R: are
O: Congratulation - challenge passed!
C: Are you _ turtle?
R: a
O: Congratulation - challenge passed!
C: You'll __ sorry...
R: be
O: Congratulation - challenge passed!
C: You are standing on __ toes.
R: my
O: Congratulation - challenge passed!
C: You should go ____.
R: party
F: Wrong guess 'party' - expected 'home'
C: There __ a fly on your nose.
Q:
M: You mastered 5/6 challenges. Good bye!
```

The server obtains the phrases by running `fortune -s` as a child process and reading the output produced by the child process from a pipe. The server than selects a random word, replaces it with underscores, and sends the challenge to the client. The server has to accept the following command line options:

```
gwgd [-t] [-f] [-p port]
```

The `-t` option enables thread mode where every client is handled by a separate thread. The `-f` option enables forking mode, where every client is handled by a separate child process. Without the `-t` and `-f` options, the server will only handle one client at a time and block any additional clients until the current client has finished.

The assignment can be broken down into the following steps:

a) Write a main function implementing option parsing and performing any necessary initialization work.

b) Write code to create listening sockets (for both IPv4 and IPv6) and implement a select loop that picks up incoming connection requests arriving on the listening sockets.

c) Implement the game play in blocking mode, i.e., the server handles only one client at a time.

d) Implement the game play in fork mode, i.e., every client is handled by a separate child process of the main server. Make sure that child processes do not become zombie processes.

e) Implement the game play in threaded mode, i.e., every client is handled by a separate thread of the main server.

f) Cleanup your code, make it clear and easy to read. Minimize code duplication, ideally the game logic should be the same code regardless whether the server is running in blocking, threaded or forking mode.

You are allowed (and even encouraged) to reuse the TCP functions documented in the lecture notes. To separate the game logic from the server logic, you can implement a player object handling the game play and a challenge object handling the fetching and preparation of game challenges.

```c
/*
 * chlng.h --
 */

#ifndef CHLNG_H
#define CHLNG_H

typedef struct {
    char *text;                 /* text with a missing word */
    char *word;                 /* the missing word */
} chlng_t;

/* Allocate a new challenge. */
extern chlng_t* chlng_new(void);

/* Reset the internal state of the challenge. */
extern void chlng_reset(chlng_t*);

/* Delete a challenge and all its resources. */
extern void chlng_del(chlng_t*);

/* Fetch new text from an invocation of 'fortune'. */
extern int chlng_fetch_text(chlng_t *c);

/* Select and hide a word in the text. */
extern int chlng_hide_word(chlng_t *c);

#endif
```

```c
/*
 * player.h --
 */

#ifndef PLAYER_H
#define PLAYER_H

#include <stdbool.h>
```

```c
#include "chlng.h"

typedef struct player {
    int solved;                 /* correctly solved challenges */
    int total;                  /* total number of challenges */
    bool finished;              /* true if we are done */
    chlng_t *chlng;             /* current challenge */
} player_t;

/* Allocate a new player. */
extern player_t* player_new(void);

/* Reset the internal state of the player. */
extern void player_reset(player_t*);

/* Delete a player and all its resources. */
extern void player_del(player_t*);

/* Allocate a new challenge for the player. */
extern int player_fetch_chlng(player_t*);

/* Retrieve a greeting message. */
extern int player_get_greeting(player_t*, char **);

/* Retrieve the challenge message. */
extern int player_get_challenge(player_t*, char **);

/* Post a message to the player and retrieve the response message. */
extern int player_post_challenge(player_t*, char *, char **);

#endif
```

With these interfaces, a simple standalone program to play the game can look like this:

```c
/*
 * gwg.c --
 */

#define _POSIX_C_SOURCE 200809L

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

#include "player.h"

int main(void)
{
    player_t *p;
    char *msg;
    int rc;

    p = player_new();
    if (! p) {
        return EXIT_FAILURE;
    }

    rc = player_get_greeting(p, &msg);
    if (rc > 0) {
        (void) fputs(msg, stdout);
        (void) free(msg);
    }
```

```
    while (! (p->finished)) {
        char *line = NULL;
        size_t linecap = 0;

        rc = player_get_challenge(p, &msg);
        if (rc > 0) {
            (void) fputs(msg, stdout);
            (void) free(msg);
        }

        if (getline(&line, &linecap, stdin) <= 0) {
            rc = player_post_challenge(p, "Q:", &msg);
            if (rc > 0) {
                (void) fputs(msg, stdout);
                (void) free(msg);
            }
            break;
        }
        rc = player_post_challenge(p, line, &msg);
        if (rc > 0) {
            (void) fputs(msg, stdout);
            (void) free(msg);
        }
        (void) free(line);
    }
    player_del(p);

    return EXIT_SUCCESS;
}
```