Exercise sheet 9 – Process communication 2

Goals:

- Message queues
- Shared memory

Exercise 9.1: Message queue log server

(a) Update the `OS_exercises` repository with `git pull`.

Proposal for solution: `git pull`

(b) Change into the `OS_exercises/sheet_09_process_comm2/message_queue` directory.

Proposal for solution:

```
cd sheet_09_process_comm2/message_queue
```

(c) Inspect the `log_server_mq.c`.

(d) Inspect the `log_client_mq.c`.

(e) Complete `log_client_mq.c`.

Proposal for solution:

```c
#include <stdio.h>      //printf, perror
#include <stdlib.h>     //EXIT_FAILURE, EXIT_SUCCESS
#include <stdbool.h>    //bool
#include <string.h>     //strcmp
#include <sys/msg.h>    //msgget, msgrcv, msgsnd, msgctl

key_t MESSAGE_QUEUE_KEY   = 0x4242; //key of the message queue
#define MAX_MESSAGE_LEN     (1024)  //max length of messages

//structure for messages
typedef struct {
    long priority;
    char message[MAX_MESSAGE_LEN];
} message_t;

int main() {
    //open the message queue
    int message_queue_id = msgget(MESSAGE_QUEUE_KEY, IPC_PRIVATE);
    if(message_queue_id < 0) {
        printf("Error: can't open message queue!\n");
        exit(EXIT_FAILURE);
    }

    //client endless loop
    while(true) {
        //fetch user input from console (stdin)
```

```
27          char buffer[MAX_MESSAGE_LEN];
28          fgets(buffer, MAX_MESSAGE_LEN-1, stdin);
29
30          if(strcmp("\\quit\n", buffer) == 0) { //quit if user types: \quit
31              break;
32          }
33
34          //prepare message
35          message_t message;
36          message.priority = 1;
37          strcpy(message.message, buffer);
38
39          //send message to message queue
40          int size = msgsnd(message_queue_id, &message, MAX_MESSAGE_LEN-1, 0);
41          if(size < 0){
42              printf("Error: can't send message;");
43              exit(EXIT_FAILURE);
44          }
45      }
46
47      return EXIT_SUCCESS;
48  }
```

(f) Compile your program into `log_client_mq`. Use the prepared `Makefile` with the target `log_client_mq` for this!

Proposal for solution: `make log_client_mq`

(g) Start the provided `log_server_mq`.

Proposal for solution: `./log_server_mq`

(h) Start a separate console to continuously see the log in `log_server_mq.log`.

Proposal for solution: `tail -f log_server_mq.log`

(i) Start your `log_client_mq` and send some messages to the `log_server_mq`.

Proposal for solution: `./log_client_mq`

Exercise 9.2: Shared memory log server

(a) Change into the `OS_exercises/sheet_09_process_comm2/shared_mem` directory.

Proposal for solution:

```
cd sheet_09_process_comm2/shared_mem
```

(b) Inspect the `log_server_sm.c`.

(c) Inspect the `log_client_sm.c`.

(d) Complete `log_client_sm.c`.

Proposal for solution:

```
1  #include <stdio.h>      //printf, perror
2  #include <stdlib.h>     //EXIT_FAILURE, EXIT_SUCCESS
3  #include <stdbool.h>    //bool
```

```c
#include <string.h>    //strcmp, strcpy
#include <sys/shm.h>   //shm*
#include <fcntl.h>     //flags: O_CREAT, O_EXCL
#include <semaphore.h> //sem_open, sem_wait, sem_post, sem_close

const key_t SHM_KEY  = 0x424242;   //Key of the shared memory segment
#define MAX_SHM_LEN        (1024)    // length of shared memory

/*
 * The log server consists of two semaphores
 * - ready_to_write_sem - Initialized with 1
 * - ready_to_read_sem  - Initialized with 0
 */
#define SEMAPHORE_READY_TO_WRITE_NAME "/log_rw" //name of semaphore
#define SEMAPHORE_READY_TO_READ_NAME  "/log_rr" //name of semaphore
sem_t* ready_to_write_sem = NULL;
sem_t* ready_to_read_sem  = NULL;
const int PERM            = 0600;    //Permission to the semaphore and shared memory

void fetch_semaphore() {
    ready_to_write_sem = sem_open(SEMAPHORE_READY_TO_WRITE_NAME, O_EXCL);
    if(ready_to_write_sem == SEM_FAILED){
        perror("Error when opening the ready_to_write_sem ...\n");
        exit(EXIT_FAILURE);
    }

    ready_to_read_sem = sem_open(SEMAPHORE_READY_TO_READ_NAME, O_EXCL);
    if(ready_to_read_sem == SEM_FAILED){
        perror("Error when opening the ready_to_read_sem ...\n");
        exit(EXIT_FAILURE);
    }
}

void close_semaphore() {
    if(sem_close(ready_to_write_sem) == -1){
        perror("Error can't close ready_to_write_sem ...\n");
        exit(EXIT_FAILURE);
    }

    if(sem_close(ready_to_read_sem) == -1){
        perror("Error can't close ready_to_read_sem ...\n");
        exit(EXIT_FAILURE);
    }
}

int main() {
    //fetch the semaphores
    fetch_semaphore();

    //get existing shared memory
    int shared_mem_id = shmget(SHM_KEY, 0, IPC_PRIVATE);
    if(shared_mem_id < 0) {
        perror("Error: can't get shared memory!\n");
        exit(EXIT_FAILURE);
    }

    //attach the shared memory
    void* shared_mem_address = shmat(shared_mem_id, NULL, 0);
    if(shared_mem_address == (void*)-1) {
```

```
63          perror("Error: can't attach shared memory!\n");
64          exit(EXIT_FAILURE);
65      }
66
67      //let buffer point the shared mem address as a char pointer
68      char* buffer = (char*)shared_mem_address;
69
70      //client endless loop
71      while(true) {
72          char message[MAX_SHM_LEN];
73          printf("enter message: ");
74          fgets(message, MAX_SHM_LEN, stdin);
75
76          if(strcmp("\\quit\n", message) == 0) { //quit if user types: \quit
77              break;
78          }
79
80          //wait until the shared memory is free
81          sem_wait(ready_to_write_sem);
82
83              //write the message into the shared memory
84              strcpy(buffer, message);
85
86          //signal the server that it can now read the shared memory
87          sem_post(ready_to_read_sem);
88      }
89
90      //detach shared memory
91      shmdt(shared_mem_address);
92      buffer = NULL;
93      shared_mem_address = NULL;
94
95      //close semaphore
96      close_semaphore();
97
98      return EXIT_SUCCESS;
99  }
```

(e) Compile your program into `log_client_sm`. Use the prepared `Makefile` with the target `log_client_sm` for this!

Proposal for solution: `make log_client_mq`

(f) Start the provided `log_server_sm`.

Proposal for solution: `./log_server_sm`

(g) Start a separate console to continuously see the log in `log_server_sm.log`.

Proposal for solution: `tail -f log_server_sm.log`

(h) Start your `log_client_sm` and send some messages to the `log_server_sm`.

Proposal for solution: `./log_client_sm`