Operating Systems 2016/17
Assignment 6

Prof. Dr. Frank Bellosa
Dipl.-Inform. Marc Rittinghaus

Submission Deadline: Monday, December 05th, 2016 – 23:59

A new assignment will be published every week, right after the last one was due. It must be completed before its submission deadline.

The assignments must be filled out online in ILIAS. Handwritten solutions are no longer accepted. You will find the online version for each assignment in your tutorial's directory. **P-Questions** are programming assignments. Download the provided template from ILIAS. Do not fiddle with the compiler flags. Submission instructions can be found on the first assignment.

In this assignment you will get familiar with interprocess communication.

T-Question 6.1: Interprocess Communication

a. What are the two fundamental models of interprocess communication? Give a short explanation for each. **2 T-pt**

b. Why is IPC via shared memory often more difficult to use for an application developer? **1 T-pt**

c. Explain the concept of mailboxes for IPC. How are mailboxes uniquely identified in Linux? **2 T-pt**

d. You have been asked to write a server application using message-based IPC. You can choose between two request processing models: **3 T-pt**

Forking For each incoming request the server forks, creating a new worker that is responsible for processing the request. The worker exits afterwards.

Worker Pool At program start a fixed number of worker threads are created. An incoming request is directed to an idle worker thread of the pool or queued if all threads are busy.

Briefly compare the two models regarding implementation cost, resources usage (CPU, memory, etc.), and complexity of data sharing between workers.

P-Question 6.1: Pipes

Download the template **p1** for this assignment from ILIAS. You may only modify and upload the file `pipe.c`.

In assignment 2 you wrote a simple program starter that used the `fork`, `exec` and `waitpid` system calls to start a program and wait for its exit. Your solution had to return the exit status of the new process or the special value 127 to indicate error conditions in your own program. However, the solution could not properly detect if the `fork`ed child failed to `exec` or if the started program normally exited with status code 127.

In this question you will extend the program starter to receive proper error information.

a. Complete the function `run_program` so that the `fork`ed child transmits the error number `errno` to the parent (the program starter), if the call to `exec` fails. Your function should fulfill the following requirements: **4 P-pt**

- Returns -1 on any error; the exit status of the child process otherwise.
- Allows the caller to read the correct error number from `errno`.
- Uses the `pipe2` system call to create a new pipe and sets the necessary flags to avoid leakage of the pipe's file descriptors into the `exec`ed program.
- Uses the `write`/`read` system calls to write to and read from the pipe, transmitting – on error – the error number `errno` of `exec` from the child to the parent.
- Closes unnecessary pipe endings in the parent and child respectively as soon as possible and fully closes the pipe before returning to the caller. **Do not leak file descriptors!**

Hints: Assume that the `write` and `read` system calls do not fail and ignore errors from the `close` system call. The template only marks the most important locations that need to be extended, you need to add further code to fulfill all requirements.

```
int run_program(char *file_path, char *argv[]);
```

P-Question 6.2: Message Queues

Download the template **p2** for this assignment from ILIAS. You may only modify and upload the file `message_queue.c`.

In this question you will write a simple client-/server application using two processes and a mailbox (known as message queue in Linux) for communication. The server accepts commands from the client and performs corresponding actions. The message format is defined in the template by the `Message` structure, the available commands are defined in the `Command` enumeration.

a. Implement the server in the `runServer` function. The server should adhere to the following criteria: **3 P-pt**

- Returns -1 on any error, 0 otherwise.
- Creates and initializes a new message queue, which takes `Message` structures and provides space for 10 messages, using the `mq_open` call and appropriate flags for read-only access and `QUEUE_NAME` as name.
- Fails if the message queue already exists.
- Receives messages from the client and processes them until an exit condition is met.
- Implements the following commands:

  CmdExit Exits the server

  CmdAdd Adds the two parameters supplied in the message and prints the result with the `FORMAT_STRING_ADD` format string.

  CmdSubtract Subtracts the second message parameter from the first one and prints the result with the `FORMAT_STRING_SUBTRACT` format string.
- Exists on error or on reception of an unknown command.
- Closes the message queue on exit and unlinks it

```
int runServer();
```

b. Implement the client functions. Assume the message queue to be already created by the server and ready to be opened for write access by the client. **2 P-pt**

```
mqd_t startClient();
int sendExitTask(mqd_t client);
int sendAddTask(mqd_t client, int operand1, int operand2);
int sendSubtractTask(mqd_t client, int operand1, int operand2);
int stopClient(mqd_t client);
```

Total:
8 T-pt
9 P-pt