Documentation for Assignment ‘Interprocess Comunication‘

Niklas Stylianou 1284037 & Maurice Wimmer 1250175

December 3, 2019

1 Specification of Message Datastructures

Request Message:

- hash (uint128_t) - hash to be matched.
- first_letter (char) - starting character of plaintext password.
- alphabet_size (int) - alphabet ranges from ’a’ to ’a + alphabet_size - 1’.
- quit_flag (bool) - if true, exit worker process.
- ID (int) - identifies which hash corresponds to which plaintext password.

Response Message:

- finished (bool) - if true, worker process found matching plaintext password.
- password (char[MAX_MESSAGE_LENGTH + 1]) - plaintext password if found.
- ID (int) - identifies which plaintext password corresponds to which hash.

2 Interprocess Communication

The farmer starts off by initializing a Request MQ and Response MQ. Then the farmer process dispatches worker processes via *fork()*. Next the Request MQ receives jobs until it is full. Once a worker process reads a message from the Request MQ it sends a Response with the *finish* flag set to false. This indicates to the farmer that a message was pulled from the Request MQ and that a new Request should be pushed.
Within a while loop the farmer reads from the Response MQ via *mq_receive()* which blocks execution to prevent busy waiting.
Upon receival of a Response the farmer checks if the *finished* flag was set to true, which indicates that a password has been found. In that case the farmer stores the password within an array. In any case a new message is pushed into the Request MQ if there are still jobs to be done.
If the amount of received passwords equals the size of the given array of hashes, the farmer prints out all plaintext passwords, closes the message queues, exists the children processes and finally returns. Closing the worker processes is realized by filling up the Request MQ with as many messages as there are workers. The *quit_flag* is raised in these messages to indicate to the workers to exit. Each worker consumes at most one such message.

3 Process creation

Process creation is done by the farmer via two methods. *create_worker(pid_t *processID)* creates a single worker by calling the *fork()* method provided by the POSIX API. The child process (i.e. if *fork()* returns 0) executes the worker binary while passing on the names of the MQ’s. The parent process merely returns from the method. *create_workers(pid_t workers_processID[ ])* simply loops over *create_worker(pid_t *processID)* until *NROF_WORKERS* are created.