2INC0 Operating Systems

Zachary Kohnen (1655221)

Inter-process Communication

In the design of any inter-process communication (IPC) there must be a common structure of the messages passed between each process so that each can interpret the messages in the way that the sender intended. A common method, and the method used in my implementation, to encode and decode data is to send the raw bytes of a C data structure. This can in some cases cause problems since this requires reading the raw bytes of a data structure from memory and sending those bytes over an IPC channel in the hopes that the receiver has been compiled with the same version of the C structure. This is helped by the fact that C data structures have a very well established representation in memory. Unlike other languages like Rust, C will store the contents of structs as a contiguous block of memory in the same order they are defined so any C program compiled with the same struct will represent it in memory exactly the same which can be used to my advantage, simplifying the message structure.

Data Structures

The task requires that a farmer process send messages through a (SPMC) message queue to many worker processes which will each take messages from the queue to process, returning their results down a separate (MPSC) message queue which is read from the farmer process and stored.

To facilitate this data transfer, two data structures are needed. One to represent the job sent to the workers and one to represent the response sent back from the workers. The data structures are shown below with comments to explain the importance of each field.

```c
typedef struct {
    char starting_char; // The first char of the message to search for
    char alphabet_start; // The min char value that could be in the message
    char alphabet_stop; // The max char value that could be in the message
    uint128_t hash; // The hash to compare against
    int hash_id; // A unique ID of the hash to identify responses
} job_t;

typedef struct {
    char match[MAX_MESSAGE_LENGTH + 1]; // The message that matches the hash
    int hash_id; // The ID of the hash that was matched against
} response_t;
```

The "`hash_id`" in the data structures is the index of the hash in the array stored in the farmer. This id is needed in order to preserve the output ordering of the hashes. If this id was not used, it is very likely that the workers would finish the hash jobs in a different order than they were dispatched since they jobs are processed in parallel.

Stopping Workers

When all jobs are finished, the workers need a way to detect that there are no more jobs to process and for them to exit. My implementation makes use of sending a job with the starting char as the NULL char since it is impossible for this to be part of the message given how the C language stores strings as arrays of non-NULL characters followed by a NULL character to signify the end of the string. The farmer process, upon completion of all jobs, will send out one of these messages to the queue for each of the workers so that each will consume the message, following by an immediate exit.