Assignment 5

Futures: Part 2

In this assignment, we are adding additional flags for the modes of operation for the future. The three flags used are FUTURE_EXCLUSIVE, FUTURE_SHARED and FUTURE_QUEUE. There are three states also which include FUTURE_EMPTY, FUTURE_WAITING and FUTURE_VALID.

FUTURE_EXCLUSIVE:

In this mode the future that we would create would have a one-to-one relationship between the threads calling future_get and future_set. Future is initialized in the FUTURE_EMPTY state. The process would be blocked if it would make a call to future_get in the FUTURE_EMPTY state and the process id is stored in the pid field of the future, then we would change the state of the future to FUTURE_WAITING. Further calls to future_get would result in a system error. If we call the future_set in the FUTURE_EMPTY state, then the value returned by the function is stored in the value field and state is changed to FUTURE_VALID.

FUTURE_SHARED:

In this mode there is a one-to-many relationship between the threads calling future_get and future_set. An error is returned in case multiple threads call future_set at the same time. There is only one thread which can access this function and set the value of the future at a given point. Multiple threads are able to call future_get and fetch the value of the future. All the threads which are waiting to get the value would be put in the get_queue and when the value of the future is set then all the waiting threads would be resumed. I use the resume and suspend system calls to resume the threads and to put the threads in the waiting state respectively.

FUTURE_QUEUE:

In this mode there is a many-to-may relationship between the threads that call future_get and future_set. Two queues namely get_queue and set_queue were used to hold the waiting threads. Multiple threads call future_get and future_set function. In the case when one thread called future_get and the other threads are waiting to obtain that value in the get_queue, then first the value of the future is set and then the threads from the queue are resumed one by one on the FIFO basis and provided that value. In case get_queue is empty then I enqueue the thread which calls future_set into the set_queue.

In another case when there is only one thread which is calling future_get and multiple threads which are present in the set_queue then based on the FIFO order, the first thread would be removed from the queue and the thread calling get obtains the value set by this thread. Similarly, when there is no thread waiting to set the value then the thread which needs value would be enqueued in the get_queue.

In addition to the get and set functionality, there is also a future_free call which would free the memory occupied by the future. There is also a future_alloc call which would allocate the memory for the future when it if first created.

These functions are then used to create futures, implement the functionalities and also free the memory when needed. One such functionality is finding the Nth number in the Fibonacci series. In the Fibonacci function, I am first allocating the space for the fibfut array and then creating and resuming the process. Then I am calling the future_get function by passing the fibfut array along with the result variable whose value is initialized to 0 and then the value keeps changing and in the end we get the final result for the Nth Fibonacci number.