***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.