Design Document

Implementation of futures and using it for synchronization:

This project is an implementation of the structure called future in C with the underlying operating system XINU and then using it to synchronize producer and consumer threads. The code was run successfully with XINU OS deployed on RPI node 21.

Introduction to Futures:

Future is a structure used for implementing synchronization in various programming languages. Future objects are expected to generate values in the future. They offer functions get() and set(). The difference between a thread and a future is that we can return values in futures. The get function is used to receive values from the thread.

Implementation of futures

1. Structure of a Future:

   Futures have been implemented using semaphores. It is a struct which consists of a variable which contains the shared value between the producer and consumer. The other variable flag will contain the ID of the semaphore created for the future.

2. Initialization of Future: *future_alloc()*

   In the initialization, space is allocated to the object of the 'future' struct. It creates a semaphore with count 0 (it means that the future is empty).

3. get() function: *future_get()*

   This function is invoked by the consumer to return the value produced:
   i. If the consumer tries to consume on a full future, then it consumes the value and make it empty.
   ii. If the consumer tries to consume on an empty future, then it blocks and waits for a signal to continue.
   iii. If the consumer tries to consume on an empty future on which another consumer thread is waiting, it returns an error.

   This function checks if the semaphore has a count less than 0. If yes, it means that there is already a thread waiting to consume the value. Hence it returns an error.
   Then the function calls wait() function of the semaphore where Count 1 signifies full future and count 0 signifies empty future. So, wait() on count 1 will yield value whereas that on 0 will block for a signal.

4. Set() function: *future_set()*

   This function is invoked by the producer to produce a value:

   i. If the producer tries to produce on an empty future, then it produces the value and make it full.
   ii. If the producer tries to produce on a full future, then it returns an error.

   The function checks if the semaphore is greater than 0. If yes, then it means that there is already a value produced. So, another attempt to produce will return an error. If the value is 0, then it will produce a value. So, we are maintaining the future as 'full' or 'empty' based on the value of the semaphore count.

5. States of the future:

   The states of the future can be identified by the count of the semaphore:
   Value of the semaphore count:

   i. 1 means Full future
   ii. 0 means empty future
   iii. Less than 0 means there is a consumer thread waiting on the semaphore

6. The synchronization between producer-consumer is achieved by creating threads for producer and consumer and sending the reference of a common future to all the threads in which synchronization is needed. This future is created and initialized with *future_alloc()* with its semaphore set to count 0 and value initially empty. Then, the producer and consumer threads call future_set() and future_get() respectively and the future ensures that both are synchronized as per the rules mentioned in 3.i, 3.ii, 3.iii and 4.i, 4.ii.