# AOS Futures Assignment - Report

Murali K

Futures are implemented in the 3 modes: FUTURE_EXCLUSIVE, FUTURE_SHARED, FUTURE_QUEUE and observed to work well with the cases I tested.

When a future_t* variable is created, each with a different mode, we allocate memory by passing mode to the future_alloc function and initialize the future variables.

➻ *FUTURE_EXCLUSIVE implementation* : In the exclusive mode, 2 threads (say producer and consumer) are created in  xsh_prodcons.c file in shell folder.

- If producer is called first, the value of the future is assigned in the value field and prints the value. Later when consumer is called, the value is fetched from the future and displayed.
- If consumer is called prior to producer, we block the consumer. After producer is called (future_cons function), it sets the value and resumes the suspended consumer. The consumer now fetches the value and prints it.

➻ *FUTURE_SHARED implementation* : In shared mode only one producer is allowed to call future_set(). If there are more than 1 producer, I have prevented them from calling future_set() by including a simple constraint that checks if process ID of the first process matches with the current process' PID. Since it does not match the extra producers cannot access future_set() thus achieving one-to-many relationship where multiple consumers can access the same the value set by the first producer.

- If producer is called first, the value of the future is assigned in the value field and prints the value. Later when multiple consumers are called, the value is fetched from the future and displayed for each consumer.
- If some consumer thread(s) are called prior to producer, they are suspended and put in get_queue of the future. As soon as the producer executes and sets the value, all the suspended consumers present in queue are resumed and value is printed. The consumers that execute after producer get value directly from value field set in future and print it, which avoids suspension.

➻ *FUTURE_QUEUE implementation* : In this mode, we maintain separate 2 queues one for producer and one for consumer. Each consumer waits for a unique producer to set value and to display it.

- In a single pair of producer and consumer, if producer runs prior to consumer, then the producer places itself in the producer's queue and suspends. It only resumes after the respective consumer tries to fetch value.
- If consumer executes prior to a producer, it places itself in consumer's queue and suspends itself. The PID is stored in the list and when respective producer runs and sets the value, the consumer is dequeued and resumes it's execution.

To free up the memory allocated, a call is made to future_free() by passing the future_t* variable as parameter. In the function call, freemem function is used to free the memory.