

AOS Futures Assignment - Report

Murali K

Futures are implemented in 3 modes: `FUTURE_EXCLUSIVE`, `FUTURE_SHARED`, `FUTURE_QUEUE` along with interrupt disabling and observed to work well with the cases I have 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 calling `getmem` function. In the `future_alloc` function I have initialized the variables of `future_t` structure.

» *`FUTURE_EXCLUSIVE` implementation* : In the exclusive mode, 2 threads (say producer and consumer) are created in `xsh_prodcons.c` file located in shell folder. Any further producers or/and consumers are not allowed execute and `YSERR` is returned.

- If producer is called first, the value of the future is assigned in the value field and producer prints the value. Later when consumer is called, the value is fetched from the future and displayed/printed.
- If consumer is called prior to producer, we block the consumer. After producer is called (which is through `future_cons` function), it sets the value field and resumes the suspended consumer. The consumer now fetches this 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 constraint that checks if process ID of the first process matches with the current process' PID. Since this match fails, the extra producers cannot access `future_set()` thus achieving one-to-many relationship where multiple consumers can access the value set by the first producer.

- In the scenario when producer is called first, the value of the future is assigned in the value field and producer 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 follow the procedure mentioned in above point, thereby avoiding suspension.

» *`FUTURE_QUEUE` implementation* : In this mode, we maintain 2 separate 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 itself. This producer is only resumed by the respective consumer, after producer's partner: the consumer, tries to fetch value. FIFO order is thus implemented.
- If consumer executes prior to a producer, it places itself in consumer's queue and suspends itself. When respective producer runs and sets the value, the consumer is dequeued and resumes its execution. Because of FIFO implementation, only the first producer or consumer are dequeued and resumed.

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.