Assignment 5 - Report

Implementation of Queue -

A linked-list implementation of queue is designed to accommodate FUTURE_QUEUE and FUTURE_SHARED.

The following is the structure of the Queue –

1. Queue List-Node –
   a. Process ID
   b. Next - Pointer to the next queue List-node
2. Queue
   a. Head – Pointer to the head of queue List-Node
   b. Tail – Pointer to the tail of queue List-Node

The struct of Future has *get_queue* and *set_queue*, each of type Queue.

Implementation details of Queue:

1. The head of the queue always points a dummy node which points to the first process to get itself enqueued
2. The tail of the queue points points to the last element of the queue
3. When tail and head, point to the same dummy node, then queue list empty

Functions for queue

1. *enqueue* –
   a. enqueues a new element to the tail of the queue and tail pointer now points to this element
   b. If tail and head point to the dummy element, then first element is added. Next of head pointer is modified and so is tail pointer
2. *dequeue* –
   a. dequeues the first element pointed by the next of head and modifies the pointer next of the head
   b. If there's only one node in the queue, head and tail points to the dummy element
3. *isEmpty* –
   a. Checks if the queue is empty

Implementation of FUTURE_SHARED -

1. This implementation uses the get_queue for the consumers waiting for a producer
2. A single semaphore is used to synchronize the entry
3. Since, no two process (producer or consumer) are allowed to enter together at the same time, enqueue and dequeue operations are mutually exclusive.
4. When producer comes –
   a. Case 1 – Consumer/consumers are waiting [FUTURE_WAITING]
      i. It sets the value of the future.
      ii. dequeues all consumers, till get_queue is empty
      iii. resumes consumer process from process id
      iv. Consumers resumed get the value from the value set by the future.
   b. Case 2 – No Consumer is waiting [FUTURE_EMPTY]
      i. It sets the value of the future
      ii. Change state of future to FUTURE_VALID
      iii. Ends
      iv. Whenever the first consumer comes, it gets the value set in the future
5. When consumer comes –

   a. Case 1 – Consumers are waiting [FUTURE_WAITING] or No Consumer waiting [FUTURE_EMPTY]
      i. Change state of future to FUTURE_WAITING
      ii. Enqueue itself in get_queue
      iii. Suspends itself
      iv. Resumed by the first producer, after which it gets the value and ends
   b. Case 2 – If FUTURE_VALID state, value set by producer
      i. Get the value from the future
      ii. Ends

Implementation of FUTURE_QUEUE -

1. Implementation uses *get_queue* and *set_queue* for waiting producers and consumers
2. A single semaphore is used to synchronize the entry
3. Since, no two process (producer or consumer) are allowed to enter together at the same time, enqueue and dequeue operations are mutually exclusive.
4. When producer comes –
   a. Case 1 – No producers and consumers waiting [FUTURE_EMPTY]
      i. Set the value of the future
      ii. Change the state to FUTURE_VALID
      iii. Enqueue itself to set_queue
      iv. Suspend itself
      v. After resumed by consumer, check the if other producers in *set_queue*, resume the first available producer and let it set a value, which would be available for the next consumer [ASYNCHRONOUS nature of Future is achieved!]
      vi. If no other producers, signal on the semaphore to release for the next waiting process on the semaphore
   b. Case 2 – Producer waiting [FUTURE_VALID]
      i. Enqueue itself to set_queue
      ii. Suspend itself
      iii. Resumed by previous producer (refer Case 1.v), set the value of the future for the next incoming or waiting consumer
      iv. Keeps the state as FUTURE_VALID, as value of future is set by producer
      v. Suspend itself again!
      vi. Resumed by the consumer who got the value of previously set (refer Case 2.iii) and execute Case 1.v, to set the value of the future for the next consumer in the list
   c. Case 3 – Consumer waiting & no value set, as all produced values consumed [FUTURE_WAITING]
      i. Set the value
      ii. In set check if consumer waiting – keep state as FUTURE_WAITING, else set as FUTURE_VALID
      iii. Dequeue the consumer, and resume the consumer. Ends
      iv. Consumer gets the value and ends
5. When consumer comes –
   a. Case 1 – No consumer and producer [FUTURE_EMPTY]
      i. Enqueue itself
      ii. Set state to FUTURE_WAITING
      iii. Suspend itself
      iv. Resumed by the first incoming producer, as it already as set the value in the future,
      v. get the value from the future
      vi. Ends

b. Case 2 – Consumer waiting [FUTURE_WAITING]
    i. Same as Case 1, just doesn't change the state. Remains in FUTURE_WAITING
c. Case 3 – Previously available Producer has produced the value [FUTURE_VALID]
    i. Dequeue the producer and resume with the first producer
    ii. The first producer, resumes the next available producer to set the value as in point 4. Case 1.v and Case 2.vi

Implementation of future_alloc and future_free –

- In alloc function, a new memory for future is allocated and based on the future flag passed the respective queues' are created with head and tail pointed to a dummy queue node which will the permeant head of the queue.
- In free function, based on the future flag passed the respective queues are DE queued' and the processes which are waiting on set and get queue are killed and finally deleting the queues and at last the future.

Prodcons command,

Syntax: Prodcons argument, Argument must be a valid integer value or be left blank. If blank the default value will be 2000

Options: --help – will display the help information about command.

Usage: Invokes two independent processes, Producer and consumer, which will work in synchronous manner to produce and consume values, communicated via a global variable and synchronization handled with the help of two semaphores.

Misc: All the validations and verifications have been handled such as argument validity, number of arguments and correctness of argument.

Producer,

The producer function will run the main loop for specified number of times given by the argument passed to prodcons command else will be run 2000 (default) number of times. The function will work in synch with consumer function by making use of a global variable 'n' and two semaphores, produced and consumed as described at the beginning of the document.

Consumer,

The consumer function will also run in the same fashions as that of producer, only difference will be that, it will consume values (decrement global variable 'n').

Miscellaneous,

All the procedure involved in developing a command are taken care of, such as creation and updation of prodcons.h, shprototype.c etc. Also the make file has been modified to accommodate the apps folder.

Efforts,

The functionality and logic was developed collaboratively.

Anuj,

- Modifying the xsh_prodcons.c

- Send and receive between prodcons and child
- Future alloc and future free
- Designing the queue
- Implemented fut_queue.c functionalities

Chitesh,

- Future prod and future cons
- Future get and future set
- Report
- Designing the queue
- Implemented fut_queue.c