AOS ASSIGNMENT 5: FUTURES 2

The aim is to implement the producer consumer problem using futures with new states like FUTURE_SHARED and FUTURE_EXCLUSIVE.

- **<u>Work done by Urmil</u>**

1. **<u>future_set :- syscall future_set(future* f, int* value)</u>**
    a. This file is used to set value in the future structure. This function is used in the future_prod.c file to produce values in the producer code. This function will set the value in a future if the state of the future is **FUTURE_EMPTY** or **FUTURE_WAITING**. After setting the value, the state of the future is changed to **FUTURE_VALID**. We have used new states FUTURE_SHARED and FUTURE_QUEUE to change the implementation of this function. In FUTURE_SHARED state, this function is used by only one future to set the value. In FUTURE_QUEUE state, future_set will check if there is any future in get queue, if yes then it dequeues that future and allows that future to take the set value, otherwise it pushes itself into set queue.

2. **<u>future_get :- syscall future_get(future* f, int* value)</u>**
    This file is used to get the value of the future structure. This function is used in the future_cons.c file to produce values in the consumer code. This function will get the value in a future if the state of the future is **FUTURE_VALID**. After setting the value, the state of the future is changed to **FUTURE_EMPTY**. We have used new states FUTURE_SHARED and FUTURE_QUEUE to change the implementation of this function. . In FUTURE_SHARED state, this function is used by multiple future structs to get the value. In FUTURE_QUEUE state, future_get will check if there is any future in set queue, if yes then it dequeues that future, the future sets the value after being dequeued and then the thread invoking future get uses the set value, otherwise it pushes itself into get queue.

3. **<u>prodcons.h</u>**
    a. This file defines the global variables and functions that define the structure of producer and consumer functions that will be synchronized by the future structure. Below is the structure of the file.

4. **<u>xsh_prodcons.c</u>**
    a. This file is actually used to test the future implementation code. This file includes the creation and execution of threads that is responsible to run the producer and consumer code.

5. **<u>future.h : typedef struct futent future</u>**
    a. This file serves as a source file with all the declarations required throughout the assignment. This file declares the future structure, define the future states, the future flag and contains the interface definition for all the system calls i.e. **future_alloc, future_free, future_set, future_get**.

- **<u>Work Done by Riddhi</u>**

1. **<u>future_alloc.c :- future* future_alloc(int future_flag)</u>**
    a. In this method, we allocate memory to a newly created future structure. Internally we use **getmem()** function of Xinu. Whenever a new future instance is created, it is initialized with FUTURE_EMPTY state, and the flag is always set to **FUTURE_EXCLUSIVE** for this assignment. The **getmem()** function uses the **sizeof()** function to dynamically determine the size of the future structure that is to be created and allocates this required memory size to the structure.

2. **<u>future_free.c :- syscall future_free(future* f)</u>**
    a. In this method, we deallocate the memory space given to an already allocated future structure.
    b. This method will take the future structure and the memory that was allocated to that pointer in bytes and free the allocated memory.

3. **<u>future_prod.c :</u>**
    a. This file is the producer part of the futures implementation. This file uses the function future_set() to set the value of the future. When the future state is waiting, the future state is changed to valid.

4. **<u>future_cons.c</u>**
    b. This file is the consumer part of the future implementation. This file used the **future_get()** method to check the value of the future and waits for a status. If the status is valid, it gets the value from the future and consumes the value.

5. **<u>fqueue.h</u>**
    a. This file includes the structure of queue. It includes the function interfaces and typedef of the queue structure.

6. **<u>fqueue.c</u>**
    c. This file includes the actual implementation of the queue along with the functions. The typical function used are enqueue, dequeue, peek, isempty etc.