Design Document

Extension of Futures

This project is an extension of the earlier implementation of futures in C which contained functionality to synchronize producer and consumer threads. Two new flags have been introduced and the implementation contains new functions to set the value of a function into the future and retrieve it later. The code was run successfully with XINU OS deployed on RPI node 21.

Changes to the earlier implementation of Futures:

The futures structure has been modified to include two more flags: FT_SHARED and FT_QUEUE. Also, new functions asynch() and cont() have been added to set value into the future and retrieve it later. A queue has been added to the structure futures to enqueue the values if set() has been called on a full future.

Implementation:

1. struct future:
   In the structure of the future, there will be changes to add FT_QUEUE, FT_SHARED and pointer to a queue.

2. futures* future_alloc(int future_flags):
   In this function, the object of the 'future' struct is initialized. In this project, there will be changes in this function to incorporate two new flags and a queue.
   Depending on the value of the parameter future_flags, the values of FT_QUEUE and FT_SHARED are set respectively.

future_flag	Binary	FT_QUEUE	FT_SHARED
0	0 0	False	False
1	0 1	False	True
2	1 0	True	False

   Note: It is assumed that FT_SHARED and FT_QUEUE are not both true

   FT_SHARED: If this is true, then multiple processes can get the value of a future. This means that the value of the future persists even after its value has been consumed
   FT_QUEUE: If this is true, then all the values set on a full future are enqueued into a queue structure.
   Queue : If FT_QUEUE is set, then this queue is initialized.

3. Syscall asynch(future*, void * funcPointer) :
   This function is invoked by the user program in which the user wants to run a thread for the function pointed by the pointer passed and retrieve the value returned by the execution of the function. The value of the function will be set in the future and the user can later retrieve it using future_get() or cont() function. This is implemented as follows:
   i. The user creates a new thread for the function pointer passed. The execution continues asynchronously with the main program.
   ii. If the flag FT_QUEUE is True and the future is full(the semcount > 0) , then, the value

returned by the function is added to the queue.

iii. If FT_QUEUE is True and the future is empty, then the value returned by the function is set into the future.

iv. If FT_QUEUE is False, then the existing handling of futures will be invoked, i.e. the if the future is full then, an error is returned, otherwise, the value is set into the future.

v. In all the above cases, signal() is called to wake up all the functions from waitList. This is done because, if multiple functions had called get() on a future having no value set, then all of them will be put to wait. As soon as a set() is done, these processes are woken up and they receive the current set value.

Note: checking FT_SHARED flag in this case is not necessary because when FT_SHARED is false, only one process is waiting at most.

Also, it is assumed that if the future is empty, then the queue is also empty because as soon as get() fetches a value from the future, the first element of the queue is removed and set into the future.

4. syscall cont(future*, void *funcPointer)

This function is used to retrieve the value of the future and pass this value as a parameter to the function pointed by funcPointer. The value is retrieved as follows:

i. A new thread is created . So, the retrieval continues asynchronously with the main program.

ii. If FT_SHARED is True and the future is not set(not initialised) then the thread is blocked and its added to the waitQueue of the semaphore of the future. These threads are woken up as soon as a value is set into the future by the Asynch function.

iii. If FT_SHARED is False, then the existing handling continues, I.e, the value is returned if the thread is full, otherwise the process blocks and it is added to the waitQueue of the semaphore for the future. Subsequent get() calls return error.

In this case, we check if FT_QUEUE is true. If yes, then after returning the value, since the future is empty now, we check if there are any elements present inside the queue. If yes, then the first element of the queue is set into the future.

iv. If FT_SHARED is True, and the future is set, then the value is returned. For all subsequent get()/cont() calls, the same value is returned. This is because the set value is shared by all the processes.

Note: *when FT_shared is false, semaphore count 1 implies Full future and 0 implies empty future*

5. The changes made in this project allow the futures implementation to be used as an interface by any user program to run a function aysnchronously and retrieve the value returned by the function by using futures. Also, multiple processes can now get the semaphore value if a given flag is set. Functionality is also provided for queuing the values to be set in the future.