FUTURES2 IMPLEMENTATION IN XINU

Introduction:

The design document provides an overview of the futures concept part of the new C++ standard. The futures variable is a placeholder for an eventual output generated by a function called asynchronously.

Design Considerations

- The program has an asynch() function implemented as a system call, which is used to call a function asynchronously in our case we call function int addInt() which adds two integers. The output of the asynchronous function is used to set future variable, asynch() uses future_set() within its implementation to set the value of the future variable.
- The program has a cont() function implemented as a system call, which uses future_get within its implementation and it, is used to query the future variable value.
- The future structure has two queue's implemented which are used for queuing successive future values as well to queue processes waiting on a full future in case of shared future.

  Implements the following future types:
- 1) **NORMAL FUTURE** - The future variable is set once and queried once and a successive attempt to set or query the future fails i.e. Project Futures1 implementation.
- 2) **SHARED FUTURE** - The future variable is set once and can be queried multiple times, any attempt to set the future for the second time results in failure.
- 3) **QUEUE FUTURE** - The future variable can be set and queried multiple times. The successive future variable values on a full future will be queued to produce a subsequent value of the future.

Implementation Details

- Header file <futures.h> contains function prototype and the structure corresponding to the future variable.

```
    #define FT_NORMAL 0
    #define FT_SHARED 1
    #define FT_QUEUE 2
    typedef struct futentries        /*future table entry*/
  {
    char state; /*state of the future which can be NORMAL FUTURE, SHARED FUTURE or QUEUE FUTURE*/
    int count;     /*keep track of no.of processes which either try to set or get the future */
    int vflag;     /* to check if the future is blocked */
    int flag;            /*flag to check if value is set for the future*/
    int pqueue[10]; /*process id waiting for the future*/
    int vqueue[10];  /*queued future values*/
    int front1, front2, rear1, rear2; /*flags to keep track of insertions and deletions in the queue*/
   }  futur;
```

- C file futalloc.c to allocate the future variable.

  - ➢ The function futalloc () allocates the memory to the future, sets the flag (type of future) and returns the address of the allocated future.
    Prototype: futur* future_alloc(int future_flags);

- **syscall future_set(futur *, int)**
    - ➢ The function is implemented as a system call and it is used to set the data of the future.
- **syscall future_get(futur *, int *)**
    - ➢ The function is implemented as a system call and it is used to get the data stored in future.
- **syscall futures_free(futur *)**
    - ➢ The function is implemented as a system call to free up the future variable.
- **syscall asynch(futur *f, int (*funptr)(int, int),int x, int y)** /* will run and set the future */
    - ➢ The function is implemented as a system call and it is used to call some function asynchronously in our case it is the function int addInt() (function to add two integers). The result produced from the asynchronously called function is used to set the future variable value.
- **syscall cont(futur *f, void (*funptr)(futur *))** /* will start the thread when the future is set. */
    - ➢ This function is implemented as a system call and it in turns implements the consumer function and is used to query the future variable value.

<u>Input snapshot</u>

```
futur *f1,*f2,*f3;


/* Normal Future*/

f1=future_alloc(0);

asynch(f1,addInt,5,6);

cont(f1,fconsumer);

cont(f1,fconsumer);                //process gets blocked


/*Shared future*/

f2=future_alloc(1);

asynch(f2,addInt,1,2);

cont(f2,fconsumer);                //prints 3

cont(f2,fconsumer);                //prints 3
```

```
/*Queued Future*/

f3=future_alloc(2);

cont(f3,fconsumer);                //prints 5

asynch(f3,addInt,2,3);

asynch(f3,addInt,20,30);

cont(f3,fconsumer);                //prints 50

cont(f3,fconsumer);                //prints 80

asynch(f3,addInt,50,30);
```

Output Snapshot

```
xsh$ fut

Trying to access blocked future: Process cons blocked

The sum is 11

The sum is 3

The sum is 3

The sum is 5

The sum is 50

The sum is 80
```

References

Wikipedia, CPP reference, Stack-Overflow, Xinu Approach-Linksys Edition Douglas Comer