Contribution

1.	future.h	Pratik Patel
2.	xsh_prodcons.c	Pratik Patel
3.	future_cons.c	Anand Nahar
4.	future_prod.c	Anand Nahar
5.	future_alloc.c	Anand Nahar
6.	future_get.c	Anand Nahar
7.	future_set.c	Pratik Patel
8.	future_free.c	Pratik Patel

Functions

future.h

Contains the declaration for MACROS, struct future and other functions.

Code

```
#ifndef _FUTURE_H_
#define _FUTURE_H_

#include <xinu.h>

/* define states */
#define FUTURE_EMPTY      0
#define FUTURE_WAITING    1
#define FUTURE_VALID      2

/* modes of operation for future*/
#define FUTURE_EXCLUSIVE  1

typedef struct futent
{
  int *value;
  int flag;
  int state;
  pid32 pid;
} future;

/* Interface for system call */
future* future_alloc(int future_flags);
syscall future_free(future*);
syscall future_get(future*, int*);
syscall future_set(future*, int*);

int future_cons(future *fut);
int future_prod(future *fut);
```

```
#endif /* _FUTURE_H_ */
```

xsh_prodcons.c

Contains the declaration of future variable and thread creation logic.

Code

```
#include <prodcons.h>
#include <future.h>

int n=0;                //Definition for global variable 'n'
/*Now global variable n will be on Heap so it is accessible all the processes i.e. consume and
produce*/

sid32 produced,consumed;
future *f1,*f2,*f3;

shellcmd xsh_prodcons(int nargs, char *args[])
{
  //Argument verifications and validations

  int count;            //local varible to hold count
  int flag_sem=1;
  n=0;

  if (nargs == 2 && strncmp(args[1], "--help", 7) == 0)
  {
    printf("Usage: %s\n\n", args[0]);
    printf("Description:\n");
    printf("\tProducer consumer problem\n");
    printf("\tPass a number, if number is not passed default value is 2000\n");
    printf("Options (one per invocation):\n");
    printf("\t--help\tdisplay this help and exit\n");
    return 0;
  }

  if(nargs>2)
  {
    fprintf(stderr,"\n%s: many Arguments...!!!",args[0]);
    fprintf(stderr,"\nUsage prodcons [number]");
    return 1;
  }
  else if(nargs==2)
  {
    if(strncmp(args[1],"-f",2)==0)
      flag_sem=0;
    else    //check args[1] if present assign value to count
    {
      count=atoi(args[1]);
      if(count<=0)
      {
        printf("\nPlease enter a valid value.",count);
        return 1;
      }
    }
  }
  else
    count=2000;

  if(flag_sem)
  {
    produced = semcreate(0);
    consumed = semcreate(1);

    //create the process producer and consumer and put them in ready queue.
    //Look at the definations of function create and resume in exinu/system folder for reference.
    resume( create(producer, 1024, 20, "producer", 1, count) );
    resume( create(consumer, 1024, 20, "consumer", 1, count) );
  }
  else
  {
    f1 = future_alloc(FUTURE_EXCLUSIVE);
    f2 = future_alloc(FUTURE_EXCLUSIVE);
    f3 = future_alloc(FUTURE_EXCLUSIVE);

    if(f1)
    {
      resume( create(future_cons, 1024, 20, "fcons1", 1, f1) );
      resume( create(future_prod, 1024, 20, "fprod1", 1, f1) );
    }
    else
      printf("\nError creating future f1");

    if(f2)
    {
      resume( create(future_cons, 1024, 20, "fcons2", 1, f2) );
      resume( create(future_prod, 1024, 20, "fprod2", 1, f2) );
    }
    else
      printf("\nError creating future f2");

    if(f3)
    {
     resume( create(future_cons, 1024, 20, "fcons3", 1, f3) );
     resume( create(future_prod, 1024, 20, "fprod3", 1, f3) );
    }
    else
     printf("\nError creating future f3");
   }
   return 0;
  }
```

future_cons.c

Consumes the values produced by the producer and also free's future.

Code

```
#include <future.h>

int future_cons(future *fut)
{
 int i, status;
 status = future_get(fut, &i);
 if (status < 1)
 {
  printf("future_get failed\n");
  return -1;
 }
 printf("\nConsumer consumed %d", i);

 if(!(future_free(fut)))
  return SYSERR;

 return OK;
}
```

future_prod.c

Responsible for producing the value that would be consumed by the consumer

Code

```
#include <future.h>

int future_prod(future *fut)
{
 int i,j,status;
 j = (int)fut;

 for (i=0; i<1000; i++)
 {
   j += i;
 }

 status=future_set(fut, &j);
 if (status < 1)
 {
   printf("future_set failed\n");
   return -1;
 }

 printf("\nProducer produced %d",j);
 return OK;
}
```

future_alloc.c

Allocates memory to future variable and also to value variable inside the future.

Code

```
#include <future.h>

future* future_alloc(int future_flag)
{
  future *f;
  f=(future *)getmem(sizeof(future));   //allocating memory to new future

  if(f==NULL)
  {
    printf("\nError allocating memory for future variable");
    return NULL;
  }

  f->value=(int *)getmem(sizeof(int));  //allocating to member of struct future

  if(f->value==NULL)
  {
    printf("\nError allocating memory for value in future variable");
    return NULL;
  }

  f->flag=FUTURE_EXCLUSIVE;     //initializing flag for EXCLUSIVE mode
  f->state=FUTURE_EMPTY;        //initializing state of the variable
  f->pid=-1;    //initializing pid
  *(f->value)=0;
  return f;
}
```

future_free.c

Free the memory allocated for the the value and future variables

Code

```
#include <future.h>

syscall future_free(future* f)
{
  return ((freemem(f->value,sizeof(int))) && (freemem(f,sizeof(future))));
}
```

future_get.c

Consumer calls future_get in order to fetch the value present in future variable

Code

```
#include <future.h>

syscall future_get(future *f, int *value)
{

  if(f->state!=FUTURE_EMPTY)
    return SYSERR;

  if(f->state==FUTURE_EMPTY)
  {
    f->pid=getpid();
    f->state=FUTURE_WAITING;
  }

  while(f->state==FUTURE_WAITING){
    printf("");
  }

  f->state=FUTURE_EMPTY;
  *value=*(f->value);

  return OK;
}
```

future_set.c

Producer calls future_set in order to set the value present in the future variable.

Code

```
#include <future.h>
```

```
syscall future_set(future *f, int *value)
{
  if(f->state==FUTURE_EMPTY || f->state==FUTURE_WAITING)
  {
    f->state=FUTURE_VALID;
    *(f->value)=*value;
    return OK;
  }
  return SYSERR;
}
```