AOS ASSIGNMENT 4 FUTURES (PART-1)

"future.h"

This file contains the reference to all the system calls and external and global variables.

```
#include<xinu.h>
#ifndef _FUTURE_H_
#define _FUTURE_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;
   volatile int state;
   pid32 pid;
}future;
//extern int s;
/* Interface for system call */
future* future_alloc(int future_flags);
syscall future_free(future*);
syscall future_get(future*, int*);
syscall future_set(future*, int*);
#endif /* _FUTURE_H_ */
```

"xsh_prodcons.c"

Creates the threads and the is the main controlling program. Also it is the driver program.

```
#include<prodcons.h>
int n;
int s;
int flag;
sid32 produced, consumed;
//sid32 m;
shellcmd xsh_prodcons(int nargs, char *args[])
{
n=0;
s=0;
flag=1;
int count=0;
     if(nargs==2 && strncmp(args[1], "--help", 7) == 0)
     {
```

```
        printf("Usage: %s\n\n", args[0]);
        printf("***** Description: Producer Consumer Problem
*****\n\n");
        printf("*Enter 'prodcons' command followed by valid
positive integer value.\n");
        printf("*Give only one argument\n");
        printf("*Don't give more than one argument\n");
        printf("*If no value is entered, program will run for
value = 2000.\n");
        printf("*This program is also used to implement 'Future'
concept.\n");
        printf("*To use future, use '-f' next to prodcons
command.\n\n\n");
        return 0;
    }
    if(nargs>2)
    {
        printf("Too many arguments.\n");
        return 0;
    }
    if(nargs==2 && strncmp(args[1], "-f", 2) == 0)
    {
        flag=0;
    }
    if(nargs==2)
    {
        count=atoi(args[1]);
        if(count<0)
        {
            printf("Please enter valid positive integer
value!\n",count);
        }
    }
    else
        count=2000;
    if(flag)
    {
        consumed = semcreate(1);
        produced = semcreate(0);
        //m=semcreate(1);
        resume(create(producer, 1024, 20, "producer", 1, count));
        resume(create(consumer, 1024, 20, "consumer", 1, count));
    }
    else
    {
        future *f1, *f2, *f3;
        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("Error creating the file");
          }
          if(f2)
          {
               resume( create(future_cons, 1024, 20, "fcons2", 1,
f2) );
               resume( create(future_prod, 1024, 20, "fprod2", 1,
f2) );
          }
          else
          {
               printf("Error creating the file");
          }
          if(f3)
          {
               resume( create(future_cons, 1024, 20, "fcons3", 1,
f3) );
               resume( create(future_prod, 1024, 20, "fprod3", 1,
f3) );
          }
          else
          {
               printf("Error creating the file");
          }
     }
     return 0;
}
```

"future_cons.c"

Consumes the values produced by the future_prods.

```
#include <prodcons.h>
uint32 future_cons(future *fut)
{
     int i, status;
     status = future_get(fut, &i);
```

```
    if (status != OK)
    {
        printf("future_get failed\n");
        return -1;
    }
    printf("Consumed Value is:  %d\n\n", i);
    if(!future_free(fut))
         return SYSERR;
    return OK;
}
```

"future_prod.c"

Produces the values that can be consumed by the consumer in future.

```
#include <prodcons.h>
uint32 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("failed\n");
        return(-1);
    }
    printf("Produced Value is: %d \n",j);
    return OK;
}
```

"future_alloc.c"

Allocates the memory to the future variables.

```
#include<prodcons.h>
future * future_alloc(int future_flags){
    int * val;
    future *f =(future *)getmem(sizeof(future));
    if(f==NULL)
    {
        printf("Not able to allocate memory to future");
        return NULL;
    }
    f->value =(int*)getmem(sizeof(int));
    if(f->value==NULL)
```

```
     {
          printf("Not able to allocate memory ");
          return NULL;
     }
     f->flag=future_flags;
     (*f).state=FUTURE_EMPTY;
     *(f->value)=0;
         (*f).pid=NULLPROC;
         return f;
}
```

"future_get.c"

Returns the value of future variables.

```
#include<prodcons.h>
syscall future_get(future *f, int *value)
{
     int i;
         if(f->state!=0)
{
return SYSERR;
}
     if (f->state == 0)
     {
          f->pid=getpid();
          f->state=1;
     }
     while(f->state != 2)
     {}
     *value = *(f->value);
         f->value=NULL;
     f->pid=NULLPROC;
         f->state=FUTURE_EMPTY;
         return OK;
}
```

"future_set.c"

Sets the value for future variables.

```
#include<prodcons.h>
syscall future_set(future *f,  int *value) {
     if(f->state==1 || f->state==0)
     {
          f->value = (int*)getmem(sizeof(int));
          *(f->value) = *value;
          f->state=2;
```

```
            return OK;
        }
        return SYSERR;
}
```

"future_free.c"

Frees the allocated memory

```
#include<prodcons.h>
syscall future_free(future *f)
{
        return ((freemem(f,sizeof(future))) && (freemem(f-
>value,sizeof(int))));
}
```

CONTRIBUTION:

xsh_prodcons.c: Kushal
future.h: Rudrani
future_free.c: Rudrani
future_alloc.c: Kushal
future_get.c: Rudrani
future_set.c: Kushal
future_prod.c: Kushal, Rudrani
future_cons.c: Kushal, Rudrani
Report: Kushal Rudrani