//Producer

#include<sys/ipc.h>

#define NULL 0

#include<sys/shm.h>

#include<sys/types.h>

#include<unistd.h>

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<sys/wait.h>

#include<ctype.h>

#include<fcntl.h>

#include <semaphore.h>

#include <pthread.h>

#include <sys/sem.h>

#include <sys/wait.h>

#include <sys/errno.h>

#include<time.h>

extern int errno;

#define SIZE 20 /\* size of the shared buffer\*/ #define SHMPERM 0666

int shmid; /\* id for shared memory bufer \*/

int empty\_id;

int full\_id;

int mutex\_id;

int \*buff;

sem\_t \*empty;

sem\_t \*full;

sem\_t \*mutex;

void produce()

{

int i=0,count=1,n;

int num[20];

printf("Enter the no. of random no. to generate:");

scanf("%d",&buff[0]);

n=buff[0];

printf("Generating random numbers...\n");

srand(time(0));

for(i=1;i<=buff[0];i++)

num[i]=(rand()%(50) +1);

while(1)

{

if(count>=n)

{

printf("\n Producer exited \n");

exit(1);

}

printf("\nProducer trying to aquire Semaphore Empty \n");

sem\_wait(empty);

printf("\nProducer successfully aquired Semaphore Empty

\n");

printf("\nProducer trying to aquire Semaphore Mutex \n"); sem\_wait(mutex);

printf("\nProducer successfully aquired Semaphore Mutex

\n");

buff[count]=num[count];

printf("\nProducer Produced Item [ %d ] \n",buff[count]);

printf("\nItems in Buffer %d \n",count); count++;

sem\_post(mutex);

printf("\nProducer released Semaphore Mutex \n"); sem\_post(full);

printf("\nProducer released Semaphore Full \n");

}

}

int main()

{

if((shmid = shmget (111,SIZE, IPC\_CREAT | IPC\_EXCL | SHMPERM ))<0) {

perror("unable to generate shared memory buffer\nexiting\n");

exit(-1);

}

if((empty\_id=shmget(112,sizeof(sem\_t),IPC\_CREAT|IPC\_EXCL|SHMPERM))<0

)

{

perror("unable to generate shared memory empty semaphore\nexiting\n\n");

exit(-1);

}

if((full\_id=shmget(113,sizeof(sem\_t),IPC\_CREAT|IPC\_EXCL|SHMPERM))<0)

{

perror("unable to generate shared memory full semaphore\nexiting\n\n");

exit(-1);

}

if((mutex\_id=shmget(114,sizeof(sem\_t),IPC\_CREAT|IPC\_EXCL|SHMPERM))<0

)

{

perror("unable to generate shared memory mutex semaphore\nexiting\n\n");

exit(-1);

}

buff = shmat(shmid,NULL, 0 );

empty = shmat(empty\_id,(char \*)0,0);

full = shmat(full\_id,(char \*)0,0);

mutex = shmat(mutex\_id,(char \*)0,0);

sem\_init(empty,1,SIZE);

sem\_init(full,1,0);

sem\_init(mutex,1,1);

printf("producing\n");

produce();

shmdt(buff);

shmdt(empty);

shmdt(full);

shmdt(mutex);

shmctl(shmid, IPC\_RMID, NULL);

semctl( empty\_id, 0, IPC\_RMID, NULL);

semctl( full\_id, 0, IPC\_RMID, NULL);

semctl( mutex\_id, 0, IPC\_RMID, NULL);

sem\_destroy(empty);

sem\_destroy(full);

sem\_destroy(mutex);

printf("\n Server process exited \n\n"); return(0);

}

//Consumer

#include<sys/ipc.h>

#define NULL 0

#include<sys/shm.h>

#include<sys/types.h>

#include<unistd.h>

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<sys/wait.h>

#include<ctype.h>

#include<fcntl.h>

#include <semaphore.h>

#include <pthread.h>

#include <sys/sem.h>

#include <sys/wait.h>

#include <sys/errno.h>

#include<time.h>

extern int errno;

#define SIZE 20 /\* size of the shared buffer\*/

#define SHMPERM 0666

int shmid; /\* id for shared memory bufer \*/

int empty\_id;

int full\_id;

int mutex\_id;

int \*buff;

sem\_t \*empty;

sem\_t \*full;

sem\_t \*mutex;

void consume()

{

int i=1,n;

n=buff[0];

while (1)

{

if(i>n-1)

{

printf("\n Consumer exited \n");

exit(1);

}

printf("\nConsumer trying to aquire Semaphore Full \n"); sem\_wait(full);

printf("\nConsumer successfully aquired Semaphore Full

\n");

printf("\nConsumer trying to aquire Semaphore Mutex \n"); sem\_wait(mutex);

printf("\nConsumer successfully aquired Semaphore

Mutex\n");

printf("\nConsumer Consumed Item [ %d ] \n",buff[i]); i++;

printf("\nItems in Buffer %d\n",n);

sem\_post(mutex);

printf("\nConsumer released Semaphore Mutex \n"); sem\_post(empty);

printf("\nConsumer released Semaphore Empty \n");

}

}

int main()

{

shmid = shmget (111,SIZE,0);

empty\_id=shmget(112,sizeof(sem\_t),0);

full\_id=shmget(113,sizeof(sem\_t),0);

mutex\_id=shmget(114,sizeof(sem\_t),0);

buff = shmat(shmid,NULL,0);

empty = shmat(empty\_id,(char \*)0,0);

full = shmat(full\_id,(char \*)0,0);

mutex = shmat(mutex\_id,(char \*)0,0);

printf("Consuming\n");

consume();

shmdt(buff);

shmdt(empty);

shmdt(full);

shmdt(mutex);

shmctl(shmid, IPC\_RMID, NULL);

semctl( empty\_id, 0, IPC\_RMID, NULL);

semctl( full\_id, 0, IPC\_RMID, NULL);

semctl( mutex\_id, 0, IPC\_RMID, NULL);

sem\_destroy(empty);

sem\_destroy(full);

sem\_destroy(mutex);

printf("\n Server process exited \n\n"); return(0);

}

//ProducerConsumer

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <semaphore.h>

#include <pthread.h> // for semaphore operations sem\_init,sem\_wait,sem\_post #include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/sem.h>

#include <sys/wait.h>

#include <sys/errno.h>

#include <sys/types.h>

extern int errno;

#define SIZE 10 /\* size of the shared buffer\*/ #define VARSIZE 1 /\* size of shared variable=1byte\*/ #define INPUTSIZE 20

#define SHMPERM 0666 /\* shared memory permissions \*/ int segid; /\* id for shared memory bufer \*/ int empty\_id;

int full\_id;

int mutex\_id;

char \* buff;

char \* input\_string;

sem\_t \*empty;

sem\_t \*full;

sem\_t \*mutex;

int p=0,c=0;

//

* Producer function

void produce()

{

int i=0; while (1)

{

if(i>=strlen(input\_string))

{

printf("\n Producer %d exited \n",getpid());

wait(NULL);

exit(1);

}

printf("\nProducer %d trying to aquire Semaphore Empty

\n",getpid());

sem\_wait(empty);

printf("\nProducer %d successfully aquired Semaphore Empty \n",getpid());

printf("\nProducer %d trying to aquire Semaphore Mutex

\n",getpid());

sem\_wait(mutex);

printf("\nProducer %d successfully aquired Semaphore Mutex \n",getpid());

buff[p]=input\_string[i]; printf("\nProducer %d Produced Item [ %c ]

\n",getpid(),input\_string[i]);

i++;

p++;

printf("\nItems in Buffer %d \n",p);

sem\_post(mutex);

printf("\nProducer %d released Semaphore Mutex

\n",getpid());

sem\_post(full);

printf("\nProducer %d released Semaphore Full

\n",getpid());

sleep(2/random());

} //while } //producer fn

//

// Consumer function

//

void consume()

{

int i=0; while (1)

{

if(i>=strlen(input\_string))

{

printf("\n Consumer %d exited \n",getpid()); exit(1);

}

printf("\nConsumer %d trying to aquire Semaphore Full

\n",getpid());

sem\_wait(full);

printf("\nConsumer %d successfully aquired Semaphore Full

\n",getpid());

printf("\nConsumer %d trying to aquire Semaphore Mutex

\n",getpid());

sem\_wait(mutex);

printf("\nConsumer %d successfully aquired Semaphore Mutex\n",getpid());

printf("\nConsumer %d Consumed Item [ %c ] \n",getpid(),buff[c]);

buff[c]=' ';

c++;

printf("\nItems in Buffer %d \n",strlen(input\_string)-c); i++;

sem\_post(mutex);

printf("\nConsumer %d released Semaphore Mutex

\n",getpid());

sem\_post(empty);

printf("\nConsumer %d released Semaphore Empty

\n",getpid());

sleep(1);

} //while } //consumer fn

//--------------------------------------------------------------

//Main function

//---------------------------------------------------------------

int main()

{

int i=0;

pid\_t temp\_pid;

segid = shmget (IPC\_PRIVATE, SIZE, IPC\_CREAT | IPC\_EXCL | SHMPERM );

empty\_id=shmget(IPC\_PRIVATE,sizeof(sem\_t),IPC\_CREAT|IPC\_EXCL|

SHMPERM);

full\_id=shmget(IPC\_PRIVATE,sizeof(sem\_t),IPC\_CREAT|IPC\_EXCL|

SHMPERM);

mutex\_id=shmget(IPC\_PRIVATE,sizeof(sem\_t),IPC\_CREAT|IPC\_EXCL|

SHMPERM);

buff = shmat( segid, (char \*)0, 0 );

empty = shmat(empty\_id,(char \*)0,0);

full = shmat(full\_id,(char \*)0,0);

mutex = shmat(mutex\_id,(char \*)0,0);

* Initializing Semaphores Empty , Full & Mutex sem\_init(empty,1,SIZE); sem\_init(full,1,0);

sem\_init(mutex,1,1);

printf("\n Main Process Started \n");

printf("\n Enter the input string (20 characters MAX) : "); input\_string=(char \*)malloc(20); scanf("%s",input\_string);

printf("Entered string : %s",input\_string); temp\_pid=fork();

if(temp\_pid>0) //parent

{

produce();

}

else //child

{

consume();

}

shmdt(buff);

shmdt(empty);

shmdt(full);

shmdt(mutex);

shmctl(segid, IPC\_RMID, NULL); semctl( empty\_id, 0, IPC\_RMID, NULL); semctl( full\_id, 0, IPC\_RMID, NULL); semctl( mutex\_id, 0, IPC\_RMID, NULL); sem\_destroy(empty); sem\_destroy(full); sem\_destroy(mutex);

printf("\n Main process exited \n\n");

return(0);

}