VIT-AP UNIVERSITY

School: SCOPE | Semester: WIN 2020-21

Subject: Operating System Lab | Subject Code: CSE2008

Assignment 6

NAME:MAJJIGA JASWANTH

Registration No:20BCD7171

Question1:

Implement the following program of inter process communication:

IPC Using Semaphore – Producer and Consumer Problem

CODE:

```
#include <stdio.h>

#include <stdlib.h>



int mutex = 1;



int full = 0;



int empty = 10, x = 0;



void producer()

{
```

```
    printf("20BCD7171");

    --mutex;

    ++full;

    --empty;

    x++;
    printf("\nProducer produces"
          "item %d",
          x);

    ++mutex;
}

void consumer()
{
    printf("20BCD7171");

    --mutex;

    --full;
```

```
#include <stdio.h>
#include <stdlib.h>

int mutex = 1;

int full = 0;

int empty = 10, x = 0;

void producer()
{
   --mutex;

   ++full;

   --empty;

   x++;
   printf("\nProducer produces"
```

```
        "item %d",
        x);


    ++mutex;
}


void consumer()
{

    --mutex;


    --full;


    ++empty;
    printf("\nConsumer consumes "
        "item %d",
        x);
    x--;


    ++mutex;
}


int main()
```

```
{
   int n, i;
   printf("\n1. Press 1 for Producer"
        "\n2. Press 2 for Consumer"
        "\n3. Press 3 for Exit");


#pragma omp critical

   for (i = 1; i > 0; i++) {

      printf("\nEnter your choice:");
      scanf("%d", &n);


      switch (n) {
      case 1:


         if ((mutex == 1)
            && (empty != 0)) {
            producer();
         }


         else {
            printf("Buffer is full!");
         }
         break;
```

```
        case 2:


            if ((mutex == 1)
               && (full != 0)) {
               consumer();
            }


            else {
               printf("Buffer is empty!");
            }
            break;


        case 3:
            exit(0);
            break;
        }
    }
}


    ++empty;
    printf("\nConsumer consumes "
          "item %d",
          x);
    x--;
```

```
    ++mutex;
}

// Driver Code
int main()
{
    int n, i;
    printf("\n1. Press 1 for Producer"
           "\n2. Press 2 for Consumer"
           "\n3. Press 3 for Exit");



#pragma omp critical

    for (i = 1; i > 0; i++) {

        printf("\nEnter your choice:");
        scanf("%d", &n);



        switch (n) {
        case 1:



            if ((mutex == 1)
                && (empty != 0)) {
                producer();
            }
```

```
            else {
                printf("Buffer is full!");
            }
            break;

        case 2:


            if ((mutex == 1)
                && (full != 0)) {
                consumer();
            }


            else {
                printf("Buffer is empty!");
            }
            break;


        case 3:
            exit(0);
            break;
        }
    }
}
```

OUTPUT:

QUESTION-2:

Implement the following program of inter process communication:

IPC Using Semaphore – Readers and Writers Problem

CODE:

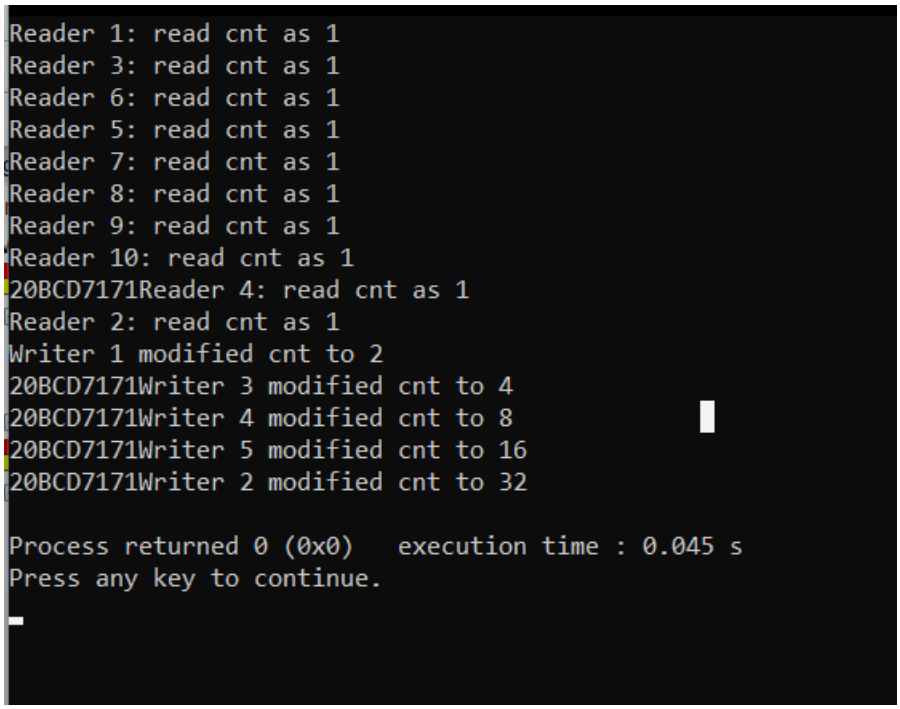
```
#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

sem_t wrt;

pthread_mutex_t mutex;

int cnt = 1;

int numreader = 0;


void *writer(void *wno)
```

```
{
   printf("20BCD7171");
   sem_wait(&wrt);
   cnt = cnt*2;
   printf("Writer %d modified cnt to %d\n",(*((int *)wno)),cnt);
   sem_post(&wrt);

}
void *reader(void *rno)
{

   pthread_mutex_lock(&mutex);
   numreader++;
   if(numreader == 1) {
      sem_wait(&wrt);
   }
   pthread_mutex_unlock(&mutex);

   printf("Reader %d: read cnt as %d\n",*((int *)rno),cnt);



   pthread_mutex_lock(&mutex);
   numreader--;
   if(numreader == 0) {
```

```
        sem_post(&wrt);
    }
    pthread_mutex_unlock(&mutex);
}

int main()
{

    pthread_t read[10],write[5];
    pthread_mutex_init(&mutex, NULL);
    sem_init(&wrt,0,1);

    int a[10] = {1,2,3,4,5,6,7,8,9,10};

    for(int i = 0; i < 10; i++) {
        pthread_create(&read[i], NULL, (void *)reader, (void *)&a[i]);
    }
    for(int i = 0; i < 5; i++) {
        pthread_create(&write[i], NULL, (void *)writer, (void *)&a[i]);
    }

    for(int i = 0; i < 10; i++) {
        pthread_join(read[i], NULL);
    }
```

```
    for(int i = 0; i < 5; i++) {
        pthread_join(write[i], NULL);
    }

    pthread_mutex_destroy(&mutex);
    sem_destroy(&wrt);

    return 0;
}
```

OUTPUT:



Implement the following program of inter process communication:

IPC Using Semaphore – Dining Philosopher Problem

CODE:

```
#include <pthread.h>

#include <semaphore.h>

#include <stdio.h>


#define N 5

#define THINKING 2

#define HUNGRY 1

#define EATING 0

#define LEFT (phnum + 4) % N

#define RIGHT (phnum + 1) % N


int state[N];

int phil[N] = { 0, 1, 2, 3, 4 };


sem_t mutex;

sem_t S[N];


void test(int phnum)

{

   if (state[phnum] == HUNGRY

       && state[LEFT] != EATING
```

```
        && state[RIGHT] != EATING) {

        // state that eating

        state[phnum] = EATING;


        sleep(2);


        printf("20BCD7171Philosopher %d takes fork %d and %d\n",

                    phnum + 1, LEFT + 1, phnum + 1);


        printf("20BCD7171Philosopher %d is Eating\n", phnum + 1);


        sem_post(&S[phnum]);

    }

}


void take_fork(int phnum)

{


    sem_wait(&mutex);




    state[phnum] = HUNGRY;


    printf("20BCD7171Philosopher %d is Hungry\n", phnum + 1);
```

```
    test(phnum);

    sem_post(&mutex);

    sem_wait(&S[phnum]);

    sleep(1);
}

void put_fork(int phnum)
{

    sem_wait(&mutex);


    state[phnum] = THINKING;

    printf("20BCD7171Philosopher %d putting fork %d and %d down\n",
          phnum + 1, LEFT + 1, phnum + 1);
    printf("20BCD7171Philosopher %d is thinking\n", phnum + 1);

    test(LEFT);
```

```
    test(RIGHT);

    sem_post(&mutex);
}

void* philosopher(void* num)
{

    while (1) {

        int* i = num;

        sleep(1);

        take_fork(*i);

        sleep(0);

        put_fork(*i);
    }
}

int main()
{
```

```
    int i;
    pthread_t thread_id[N];



    sem_init(&mutex, 0, 1);

    for (i = 0; i < N; i++)

        sem_init(&S[i], 0, 0);

    for (i = 0; i < N; i++) {

        pthread_create(&thread_id[i], NULL,
                philosopher, &phil[i]);

        printf("20BCD7171Philosopher %d is thinking\n", i + 1);
    }

    for (i = 0; i < N; i++)

        pthread_join(thread_id[i], NULL);
}
```

OUTPUT:

```
20BCD7171Philosopher 1 is thinking
20BCD7171Philosopher 2 is thinking
20BCD7171Philosopher 3 is thinking
20BCD7171Philosopher 4 is thinking
20BCD7171Philosopher 5 is thinking
20BCD7171Philosopher 2 is Hungry
20BCD7171Philosopher 1 is Hungry
20BCD7171Philosopher 3 is Hungry
20BCD7171Philosopher 4 is Hungry
20BCD7171Philosopher 5 is Hungry
20BCD7171Philosopher 5 takes fork 4 and 5
20BCD7171Philosopher 5 is Eating
20BCD7171Philosopher 5 putting fork 4 and 5 down
20BCD7171Philosopher 5 is thinking
20BCD7171Philosopher 4 takes fork 3 and 4
20BCD7171Philosopher 4 is Eating
20BCD7171Philosopher 1 takes fork 5 and 1
20BCD7171Philosopher 1 is Eating
20BCD7171Philosopher 4 putting fork 3 and 4 down
20BCD7171Philosopher 4 is thinking
20BCD7171Philosopher 3 takes fork 2 and 3
20BCD7171Philosopher 3 is Eating
20BCD7171Philosopher 5 is Hungry
20BCD7171Philosopher 1 putting fork 5 and 1 down
20BCD7171Philosopher 1 is thinking
20BCD7171Philosopher 5 takes fork 4 and 5
20BCD7171Philosopher 5 is Eating
20BCD7171Philosopher 4 is Hungry
20BCD7171Philosopher 3 putting fork 2 and 3 down
20BCD7171Philosopher 3 is thinking
20BCD7171Philosopher 2 takes fork 1 and 2
20BCD7171Philosopher 2 is Eating
20BCD7171Philosopher 5 putting fork 4 and 5 down
20BCD7171Philosopher 5 is thinking
20BCD7171Philosopher 4 takes fork 3 and 4
20BCD7171Philosopher 4 is Eating
20BCD7171Philosopher 1 is Hungry
20BCD7171Philosopher 3 is Hungry
20BCD7171Philosopher 2 putting fork 1 and 2 down
20BCD7171Philosopher 2 is thinking
20BCD7171Philosopher 1 takes fork 5 and 1
20BCD7171Philosopher 1 is Eating
20BCD7171Philosopher 5 is Hungry
20BCD7171Philosopher 4 putting fork 3 and 4 down
20BCD7171Philosopher 4 is thinking
```

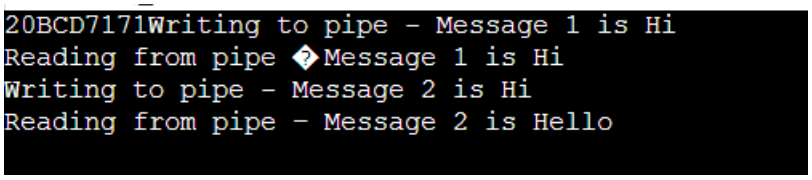
Implement the following program of inter process communication:

IPC Using Pipes

CODE:

```
#include<stdio.h>
#include<unistd.h>

int main() {
  int pipefds[2];
  int returnstatus;
  char writemessages[2][20]={"Hi", "Hello"};
  char readmessage[20];
  returnstatus = pipe(pipefds);

  if (returnstatus == -1) {
    printf("Unable to create pipe\n");
    return 1;
  }
  printf("20BCD7171");
  printf("Writing to pipe - Message 1 is %s\n", writemessages[0]);
  write(pipefds[1], writemessages[0], sizeof(writemessages[0]));
  read(pipefds[0], readmessage, sizeof(readmessage));
  printf("Reading from pipe – Message 1 is %s\n", readmessage);
  printf("Writing to pipe - Message 2 is %s\n", writemessages[0]);
  write(pipefds[1], writemessages[1], sizeof(writemessages[0]));
  read(pipefds[0], readmessage, sizeof(readmessage));
  printf("Reading from pipe – Message 2 is %s\n", readmessage);
  return 0;
}
```

OUTPUT:


20BCD7171Writing to pipe - Message 1 is Hi
Reading from pipe Message 1 is Hi
Writing to pipe - Message 2 is Hi
Reading from pipe - Message 2 is Hello