WEEK - THREADS.

Aim - Implementation of Threads

Program

```
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
void *td(void * arg);
int i,j;
void main()
{
   pthread_t t1;
   pthread_create(&t1, NULL, td, NULL);
   printf("in main now\n");
   for(i=65; i<=70; i++)
   {
      printf("main = %c\n", i);
   }
}
void *td(void *arg)
{
   for(j=1; j<=5; j++)
   printf("thread = %d\n", j);
}
```

main = E
main = F

NAME OF THE LABORATORY : _____

Name _____ Roll No. _____ Page No. _____

(2) Aim - Implementation of 2 threads in a program

Program -

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
void *print_message_function (void *ptr);
main()
{
    pthread_t thread1, thread2;
    char *message1 = "Thread 1";
    char *message2 = "Thread 2";
    int iret1, iret2;
    iret1 = pthread_create (&thread1, NULL, print_message_function,
                            (void*) message1);
    iret2 = pthread_create (&thread2, NULL, print_message_function,
                            (void*) message2);
    pthread_join (thread1, NULL);
    pthread_join (thread2, NULL);
    printf ("Thread 1 returns : %d\n", iret1);
    printf ("Thread 2 returns : %d\n", iret2);
    exit(0);
}
void *print_message_function (void *ptr)
{
    char *message;
    message = (char*) ptr;
    printf ("%s\n", message);
}
```

DEPARTMENT OF : _____

NAME OF THE LABORATORY : _____

Name _____ Roll No. _____ Page No. _____

WEEK - SEMAPHORES

(1) Aim - Implementation of a shared process

```
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#include <semaphore.h>
void *inc(void *arg);
void *dec(void *arg);
int shared = 0;
sem_t mutex;
void main()
{
    sem_init(&mutex, 0, 1);
    pthread_t t1, t2;
    pthread_create(&t1, NULL, inc, NULL);
    pthread_create(&t2, NULL, dec, NULL);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    printf("\n final value of shared = %d", shared);
}
void *inc(void *arg)
{
    int x;
```

Output

$t_1 = 1$

$t_2 = 0$

final value of shared = 0

```
    sem_wait(&mutex);
    n = shared;
    printf("\nt1 = %d", ++n);
    sleep(1);
    shared = n;
    sem_post(&mutex);
}
void *dec(void *arg)
{
    int n;
    sem_wait(&mutex);
    n = shared;
    printf("\nt2 = %d", --n);
    sleep(1);
    shared = n;
    sem_post(&mutex);
}
```

(2) Aim - Implementation of Producer Consumer Problem using semaphores.

Program -

```
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#include <semaphore.h>
#define BUFFSIZE 5.
int buf[BUFFSIZE], f=-1, r=-1;
sem_t mutex, full, empty;
void *produce(void *arg)
{ int i;
  while(1)
  { sem_wait(&empty);
    printf("\n Producer got empty semaphore\n");
    sem_wait(&mutex);
    printf("\n Producer got mutex semaphore\n");
    r = (++r)%5;
    buf[r] = i;
    printf(" Produced item is %d, f:%d, r:%d\n", i, f, r);
    sleep(1);
    sem_post(&mutex);
    printf("Producer signals mutex semaphore\n");
```

```
    sem_post (&full);
    printf ("Producer signals full semaphore \n");
  }
}
void * consume (void *arg)
{
  int i, item;
  while (1)
  {
    sem_wait (&full);
    printf ("\n Consumer got full semaphore \n");
    sem_wait (&mutex);
    printf (" Consumer got mutex semaphore \n");
    f = (++f) % 5;
    item = buf [f];
    printf ("Consumer item is %d, f: %d, r = %d", item, f, r);
    sleep(1);
    sem_post (&mutex);
    printf (" Consumer signals mutex semaphore \n");
    sem_post (&empty);
    printf (" Consumer signals empty semaphore \n");
  }
}
```

Output

Producer got empty semaphore
Consumer signals mutex semaphore
Consumer signals empty semaphore
Consumer got mutex semaphore
Consumer item is 0, f:2, r:5.

Producer got empty semaphore
Consumer signals mutex semaphore
Consumer signals empty semaphore
Consumer got mutex semaphore
Consumer item is 0, f:3, r:4.

.
.
.

NAME OF THE LABORATORY :

Name _____ Roll No. _____ Page No. ____

```
int main ()
{
    pthread_t tid1[4], tid2[4];
    int i;
    sem_init (&mutex, 0, 1);
    sem_init (&full, 0, 0);
    sem_init (&empty, 0, 5);
    for (i=0; i<4; i++)
        pthread_create (&tid1[i], NULL, produce, NULL);
    for (i=0; i<4; i++)
        pthread_create (&tid2[i], NULL, consume, NULL);
    for (i=0; i<4; i++)
        pthread_~~create (&t~~ join (tid1[i], NULL);
    for (i=0; i<4; i++)
        pthread_join (tid2[i], NULL);
}
```

(3)

Aim - Implementation of Reader-writer problem using semaphores.

Program -

```
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
pthread_mutex_t wr, mutex;
int a = 10, rc = 0;
void * reader1 (void * arg)
{ long int num;
  num = (long int) arg;
  pthread_mutex_lock (&mutex);
  rc++;
  pthread_mutex_unlock (&mutex);
  if (rc == 1)
      pthread_mutex_lock (&wr);
  pthread ("READ_IN_CS : reading data by %d by %ld \n", a, num);
  sleep(2);
  pthread_mutex_lock (&mutex);
  rc--;
  pthread_mutex_unlock (&mutex);
  if (rc == 0)
  pthread_mutex_unlock (&wr);
  printf ("\n Reader left CS %ld \n", num);
}
```

Output

WRITER_IN_CS : written data 16 by 8
written left cs 8
WRITER_IN_CS : written data 17 by 1
written left cs 1
WRITER_IN_CS : written data 18 by 9
written left cs 9
WRITER_IN_CS : written data 19 by 4
written left cs 4
WRITER_IN_CS : written data 20 by 7
written left cs 7

READ_IN_CS : reading data 20 by 0
READ_IN_CS : reading data 20 by 1
READ_IN_CS : reading data 20 by 2
READ_IN_CS : reading data 20 by 4
READ_IN_CS : reading data 20 by 3
READ_IN_CS : reading data 20 by 5

Reader left cs 0
Reader left cs 1
Reader left cs 4
Reader left cs 2
Reader left cs 3
Reader left cs 5.

```
void *writer (void *arg)
{
    long int num;
    num = (long int) arg;
    pthread_mutex_lock (&wr);
    printf ("WRITER_IN_CS : written data %d by %ld \n", ++a, num);
    pthread_mutex_unlock (&wr);
    printf ("\n writer left cs %ld \n", num);
}
int main ()
{
    pthread_t r[10], w[10];
    long int now=10, ner=6, i, j;
    pthread_mutex_init (&wr, NULL);
    pthread_mutex_init (&mutex, NULL);
    for (j=0; j<now; j++)
        pthread_create (&w[j], NULL, writer1, (void *)j);
    for (i=0; i<ner; i++)
        pthread_create (&r[i], NULL, reader1, (void *)i);
    for (j=0; j<now; j++)
        pthread_join (w[j], NULL);
    for (i=0; i<ner; i++)
        pthread_join (r[i], NULL);
}
```

(4)

Aim - Implementation of dining philosophers problem using semaphore

Program.

```
#include <stdio.h>
#include <semaphore.h>
#include <pthread.h>
#include <unistd.h>
sem_t chop[5];
sem_t mutex;
char st[5] = {'t','t','t','t','t'};
void printst()
{
    int i;
    printf("\n");
    for(i=0; i<5; i++)
        printf(" %c ", st[i]);
}
void *philo(void *arg)
{
    int i;
    long int num = (long int)arg;
    st[num] = 'h';
    sem_wait(&mutex);
    sem_wait(&chop[num]);
    sem_wait(&chop[(num+1)%5]);
```

Output

e h h t t

t e h h h

t t e h h

t t t e h

t t t t e

```
    st[num] = 'e';
    printst();
    sleep(1);
    sem_post(&chop[(num+1)%5]);
    sem_post(&chop[num]);
    sem_post(&mutex);
    st[num] = 't';
    pthread_exit(NULL);
}
int main()
{
    pthread_t phil[5];
    long int i, j;
    for(i=0; i<5; i++)
        sem_init(&chop[i], 0, 1);
    sem_init(&mutex, 0, 1);
    for(i=0; i<5; i++)
        pthread_create(&phil[i], NULL, philo, (void*)i);
    for(i=0; i<5; i++)
        pthread_join(phil[i], NULL);
}
```

(5)

Aim - Implementation of shared memory.

Program -

FILE: SHARE2.C

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/shm.h>
#include <string.h>
int main()
{
    int i;
    void *shared_memory;
    char buff[100];
    int shmid;
    shmid = shmget((key_t)2345, 1024, 0666);
    printf("Key of shared memory is %d\n", shmid);
    shared_memory = shmat(shmid, NULL, 0);
    printf("Process attached at %p\n", shared_memory);
    printf("Data read from shared memory is: %s\n", (char*)
                                                shared_memory);

    write(1, "enter the data", 14);
    read(0, buff, 100);
    strcpy(shared_memory, buff);
    sleep(2);
}
```

(Affiliated to Osmania University)
Hyderabad - 500 031.

DEPARTMENT OF : _____

NAME OF THE LABORATORY : _____

Name _____ Roll No. _____ Page No. _____

FILE SHARE1.C

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/shm.h>
#include <string.h>
int main()
{
    int i;
    void *shared_memory;
    char buff[100];
    int shmid;
    shmid = shmget((key_t)2345, 1024, 0666|IPC_CREAT);
    printf("Key of shared memory is %d\n", shmid);
    shared_memory = shmat(shmid, NULL, 0);
    printf("Process attached at %p\n", shared_memory);
    printf("Enter some data to write to shared memory\n");
    read(0, buff, 100);
    strcpy(shared_memory, buff);
    sleep(10)
    printf("Received: %s\n", (char*)shared_memory);
}
```