**<u>Name</u>: Pratyush Kumar**

**<u>Reg.No.</u>: 19BCE0506**

**<u>Course</u>: CSE2005, Operating Systems**

**<u>Slot</u>: L41+L42**

# DIGITAL ASSIGNMENT – 3

1. We declare a semaphore as:

   **sem_t sem;**

2. **sem_init()** [#include<semaphore.h>]

   int sem_init(sem_t * sem, int pshared, unsigned int value);

   initializes the semaphore *sem.

- Initial value of the semaphore = value.
- If pshared is 0, the semaphore is shared among all threads of a process (and hence need to be visible to all of them such as a global variable).
- If pshared is not zero, the semaphore is shared but should be in shared memory.

<u>Notes</u>:

- On success, the return value is 0, and on failure, the return value is -1.
- An attempt to initialize a semaphore that has already been initialized results in undefined behavior.

3. **sem_post() => V() = post() of semaphore** (#include <semaphore.h>)

int sem_post(sem_t * sem);

implements the post function described above on the semaphore *sem.

<u>Note</u>: On success, the return value is 0, and on failure, the return value is -1 (and the value of the semaphore is unchanged).

4. **sem_wait()** => **P() = wait() of semaphore** (#include <semaphore.h>)

int sem_wait(sem_t * sem);

implements the wait function described above on the semaphore *sem.

Notes:

- Here sem_t is a typdef defined in the header file as (apparently) some variety of integer.
- On success, the return value is 0, and on failure, the return value is -1 (and the value of the semaphore is unchanged).
- There are related functions sem_trywait() and sem_timedwait().

## Dining Philosophers Problem using Semaphores

- **Aim**: To solve Dining Philosophers which is a classical process synchronization problem using Semaphores

- **Algorithm**:

Each philosopher is an endluss cycle of eating and thinking

```
procedure philosopher(i)
  {
    while TRUE do
     {
       THINKING;
       take_chopsticks(i);
       EATING;
       drop_chopsticks(i);
     }
  }
```

The **takechops** procedure involves checking the status of neighboring philosophers and then declaring one's own intention to eat.

This is a two-phase protocol; first declaring the status **HUNGRY**, then going on to **EAT**.

procedure takechops(i)

```
  {
      DOWN(me);               /* critical section */
      pflag[i] := HUNGRY;
      test[i];
      UP(me);                      /* end critical section */
      DOWN(s[i])                   /* Eat if enabled */
   }
```

```
void test(i)            /* Let phil[i] eat, if waiting */
  {
      if ( pflag[i] == HUNGRY
            && pflag[i-1] != EAT
            && pflag[i+1] != EAT)
            then
            {
                  pflag[i] := EAT;
                  UP(s[i])
            }
    }
```

Once a philosopher finishes eating, all that remains is to relinquish the resources---its two chopsticks---and thereby release waiting neighbors.

```
void putchops(int i)
  {
      DOWN(me);                /* critical section */
      test(i-1);                      /* Let phil. on left eat if possible */
      test(i+1);                      /* Let phil. on rght eat if possible */
      UP(me);                         /* up critical section */
   }
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

int n;

/*#define LEFT i>0?(i-1)%N:N-1
#define RIGHT (i+1)%N*/
#define THINKING 0
#define HUNGRY 1
#define EATING 2



int state[10];
int philosopher_num[10];
```

```
}

sem_t mutex;
sem_t sem_phil[10];

void * philosopher(void *);
void takechops(int);
void putchops(int);
void test(int);

int main() {
    int k, N;
    printf("Enter the number of philosophers");
    scanf("%d", &N);
    n = N;
    pthread_t tid[N];

    printf("Dining Phiosopher\n");

    for(k = 0; k < N; k++) {
        philosopher_num[k]=k;
    }

    sem_init(&mutex,0,1);

    for(k = 0; k < N; k++)
        sem_init(&sem_phil[k],0,0);
    for(k = 0;k < N; k++)
        pthread_create(&tid[k],NULL,philosopher,&philosopher_num[k]);
    for(k = 0; k < N; k++)
        pthread_join(tid[k],NULL);

    return 0;
}

    #define LEFT i>0?(i-1)%n:n-1
    #define RIGHT (i+1)%n
```

```c
void * philosopher(void *param) {
    int i = *((int *)param);
    int tt=1;
    int et=2;
    while(1) {
        sleep(tt);
        takechops(i);
        sleep(et);
        putchops(i);
    }
}

void takechops(int i) {
    sem_wait(&mutex);
    state[i] = HUNGRY;
    printf("Philosopher %d is hungry\n",i);
    test(i);
    sem_post(&mutex);
    sem_wait(&sem_phil[i]);
}
void putchops(int i) {
    sem_wait(&mutex);
    state[i] = THINKING;
    printf("Philosopher %d is thinking\n",i);
    test(LEFT);
    test(RIGHT);
    sem_post(&mutex);
}
void test(int i) {
    if(state[i] == HUNGRY && state[LEFT] != EATING && state[RIGHT]!= EATING){
        state[i]=2;
        printf("Philosopher %d is eating\n",i);
        sem_post(&sem_phil[i]);
    }
}
```

```
pratyush@pratyush-Inspiron-5570:~/Desktop/Pratyush$ gcc -pthread -o dining_sem dining_sem.c
pratyush@pratyush-Inspiron-5570:~/Desktop/Pratyush$ ./dining_sem
Enter the number of philosophers4
Dining Phiosopher
Philosopher 0 is hungry
Philosopher 0 is eating
Philosopher 2 is hungry
Philosopher 2 is eating
Philosopher 3 is hungry
Philosopher 1 is hungry
Philosopher 0 is thinking
Philosopher 2 is thinking
Philosopher 1 is eating
Philosopher 3 is eating
Philosopher 0 is hungry
Philosopher 2 is hungry
Philosopher 1 is thinking
Philosopher 3 is thinking
Philosopher 2 is eating
Philosopher 0 is eating
Philosopher 1 is hungry
Philosopher 3 is hungry
Philosopher 0 is thinking
Philosopher 2 is thinking
Philosopher 1 is eating
Philosopher 3 is eating
Philosopher 2 is hungry
Philosopher 0 is hungry
Philosopher 1 is thinking
Philosopher 3 is thinking
Philosopher 2 is eating
Philosopher 0 is eating
Philosopher 1 is hungry
Philosopher 3 is hungry
Philosopher 2 is thinking
Philosopher 0 is thinking
Philosopher 3 is eating
Philosopher 1 is eating
Philosopher 2 is hungry
Philosopher 0 is hungry
^C
```

## Readers-Writers Problem using Semaphores

- **Aim:** To solve Readers-Writers, a classical Process Synchronization problem using Semphores
- **Algorithm:**

  Here, readers have **higher priority** than writer. If a writer wants to write to the resource, it must wait until there are no readers currently accessing that resource.
  Here, we use :-

- one mutex **m** and a semaphore **w**.
- An integer variable **read_count** :- used to maintain the number of readers currently accessing the resource. The variable read_count is initialized to 0.
- A value of **1** is given initially to **m** and **w**.
  Instead of having the process to acquire lock on the shared resource, we use the mutex m to make the process to acquire and release lock whenever it is updating the **read_count** variable.

a. **Writer Process** :

1. Writer requests the entry to critical section.
2. If allowed i.e. wait() gives a true value, it enters and performs the write. If not allowed, it keeps on waiting.
3. It exits the critical section.

```
while(TRUE)
      {
            wait(w);

            /* perform the write operation */
            signal(w);
}
```

b. **Reader Process** :

1. Reader requests the entry to critical section.
2. If allowed:
   i. it increments the count of number of readers inside the critical section. If this reader is the first reader entering, it locks the **w** semaphore to restrict the entry of writers if any reader is inside.
   ii.It then, signals mutex as any other reader is allowed to enter while others are already reading.
   iii. After performing reading, it exits the critical section. When exiting, it checks if no more reader is inside, it signals the semaphore **w** as now, writer can enter the critical section.
3. If not allowed, it keeps on waiting.

```
while(TRUE)
{
//acquire lock
wait(m);
read_count++;
if(read_count == 1)
      wait(w);

//release lock
signal(m);

/* perform the reading operation */

// acquire lock
wait(m);
read_count--;
if(read_count == 0)
     signal(w);

// release lock
signal(m);
}
```

Thus, the semaphore **w** is queued on both readers and writers in a manner such that preference is given to readers if writers are also there. Thus, no reader is waiting simply because a writer has requested to enter the critical section.

```c
#include <semaphore.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
sem_t x,y;
pthread_t tid;
pthread_t writerthreads[100],readerthreads[100];
int readercount = 0;
void *reader(void* param)
{
    sem_wait(&x);
    readercount++;
    if(readercount==1)
        sem_wait(&y);
    sem_post(&x);
    printf("%d reader is inside\n",readercount);
    usleep(3);
    sem_wait(&x);
    readercount--;
    if(readercount==0)
    {
        sem_post(&y);
    }
    sem_post(&x);
    printf("%d Reader is leaving\n",readercount+1);
    return NULL;
}
void *writer(void* param)
{
    printf("Writer is trying to enter\n");
    sem_wait(&y);
    printf("Writer has entered\n");
    sem_post(&y);
    printf("Writer is leaving\n");
    return NULL;
```

```
}
int main()
{
    int n2,i;
    printf("Enter the number of readers:");
    scanf("%d",&n2);
    printf("\n");
    int n1[n2];
    sem_init(&x,0,1);
    sem_init(&y,0,1);
    for(i=0;i<n2;i++)
    {
        pthread_create(&writerthreads[i],NULL,reader,NULL);
        pthread_create(&readerthreads[i],NULL,writer,NULL);
    }
    for(i=0;i<n2;i++)
    {
        pthread_join(writerthreads[i],NULL);
        pthread_join(readerthreads[i],NULL);
    }
}
```

```
pratyush@pratyush-Inspiron-5570:~/Desktop/Pratyush$ gcc -pthread -o readwrite readwrite.c
pratyush@pratyush-Inspiron-5570:~/Desktop/Pratyush$ ./readwrite
Enter the number of readers:5

1 reader is inside
Writer is trying to enter
2 reader is inside
2 Reader is leaving
Writer is trying to enter
Writer has entered
Writer is leaving
Writer has entered
Writer is trying to enter
1 reader is inside
Writer is leaving
2 reader is inside
1 Reader is leaving
1 Reader is leaving
2 Reader is leaving
Writer is trying to enter
Writer is trying to enter
Writer has entered
Writer is leaving
1 reader is inside
1 Reader is leaving
Writer has entered
Writer is leaving
Writer has entered
Writer is leaving
```