VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CSE 2005: OPERATING SYSTEMS [LAB]

Digital Assignment: Synchronization Problems

Anushka Dixit

[19BCE0577]

1) Producer Consumer Problem:

Program:

Select anushka_os@DESKTOP-96L9A8G: ~

GNU nano 4.8

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

int mutex=1,full=0,empty=3,x=0;

int main()
{
        int n;
        void producer();
        void consumer();
        int wait(int);
        int signal(int);
        printf("\n1.Producer\n2.Consumer\n3.Exit");
        while(1)
        {
                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;

                }
        }

        return 0;
}

int wait(int s)
{
        return (--s);
}

int signal(int s)
{
        return(++s);
}
```

```
void producer()
{
        mutex=wait(mutex);
        full=signal(full);
        empty=wait(empty);
        x++;
        printf("\nProducer produces the item %d",x);
        mutex=signal(mutex);
}

void consumer()
{
        mutex=wait(mutex);
        full=wait(full);
        empty=signal(empty);
        printf("\nConsumer consumes item %d",x);
        x--;
        mutex=signal(mutex);
}
```

Output:

```
anushka_os@DESKTOP-96L9A8G:~$ ./producer

1.Producer
2.Consumer
3.Exit
Enter your choice:1

Producer produces the item 1
Enter your choice:2

Consumer consumes item 1
Enter your choice:1

Producer produces the item 1
Enter your choice:1

Producer produces the item 2
Enter your choice:1

Producer produces the item 3
Enter your choice:1
Buffer is full!!
Enter your choice:2

Consumer consumes item 3
Enter your choice:
2

Consumer consumes item 2
Enter your choice:3
anushka_os@DESKTOP-96L9A8G:~$
```

2) Readers Writers Problem

Program:

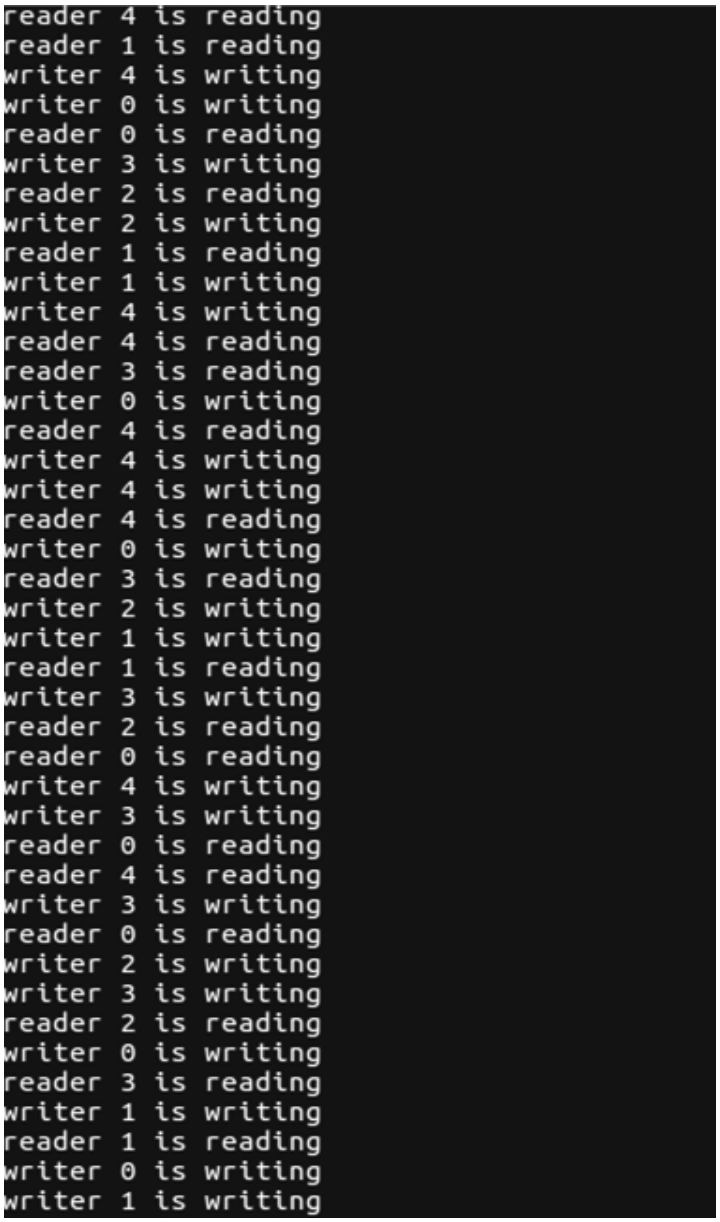
anushka_os@DESKTOP-96L9A8G: ~

GNU nano 4.8

```
#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);
    }

}
```

Output:


reader 4 is reading
reader 1 is reading
writer 4 is writing
writer 0 is writing
reader 0 is reading
writer 3 is writing
reader 2 is reading
writer 2 is writing
reader 1 is reading
writer 1 is writing
writer 4 is writing
reader 4 is reading
reader 3 is reading
writer 0 is writing
reader 4 is reading
writer 4 is writing
writer 4 is writing
reader 4 is reading
writer 0 is writing
reader 3 is reading
writer 2 is writing
writer 1 is writing
reader 1 is reading
writer 3 is writing
reader 2 is reading
reader 0 is reading
writer 4 is writing
writer 3 is writing
reader 0 is reading
reader 4 is reading
writer 3 is writing
reader 0 is reading
writer 2 is writing
writer 3 is writing
reader 2 is reading
writer 0 is writing
reader 3 is reading
writer 1 is writing
reader 1 is reading
writer 0 is writing
writer 1 is writing

3)Dining Philosopher's Problem

USING MONITORS

Code:

```
#include<stdio.h>

#define n 4

int compltedPhilo = 0,i;

struct fork{
        int taken;
}ForkAvil[n];

struct philosp{
        int left;
        int right;
}Philostatus[n];

void goForDinner(int philID){ //same like threads concept here cases implemented
        if(Philostatus[philID].left==10 && Philostatus[philID].right==10)
        printf("Philosopher %d completed his dinner\n",philID+1);
        //if already completed dinner
        else if(Philostatus[philID].left==1 && Philostatus[philID].right==1){
            //if just taken two forks
            printf("Philosopher %d completed his dinner\n",philID+1);

            Philostatus[philID].left = Philostatus[philID].right = 10; //remembering that he completed dinner by assigning value 10
            int otherFork = philID-1;

            if(otherFork== -1)
                otherFork=(n-1);

            ForkAvil[philID].taken = ForkAvil[otherFork].taken = 0; //releasing forks
            printf("Philosopher %d released fork %d and fork %d\n",philID+1,philID+1,otherFork+1);
            compltedPhilo++;
        }
        else if(Philostatus[philID].left==1 && Philostatus[philID].right==0){ //left already taken, trying for right fork
                if(philID==(n-1)){
                    if(ForkAvil[philID].taken==0){ //KEY POINT OF THIS PROBLEM, THAT LAST PHILOSOPHER TRYING IN reverse DIRECTION
                        ForkAvil[philID].taken = Philostatus[philID].right = 1;
                        printf("Fork %d taken by philosopher %d\n",philID+1,philID+1);
                    }else{
                        printf("Philosopher %d is waiting for fork %d\n",philID+1,philID+1);
                    }
                }else{ //except last philosopher case
                    int dupphilID = philID;
                    philID-=1;

                    if(philID== -1)
                        philID=(n-1);

                                                                    [ Wrote 91 lines ]
```

```
                    if(ForkAvil[philID].taken == 0){
                        ForkAvil[philID].taken = Philostatus[dupphilID].right = 1;
                        printf("Fork %d taken by Philosopher %d\n",philID+1,dupphilID+1);
                    }else{
                        printf("Philosopher %d is waiting for Fork %d\n",dupphilID+1,philID+1);
                    }
                }
            }
            else if(Philostatus[philID].left==0){ //nothing taken yet
                    if(philID==(n-1)){
                        if(ForkAvil[philID-1].taken==0){ //KEY POINT OF THIS PROBLEM, THAT LAST PHILOSOPHER TRYING IN reverse DIRECTION
                            ForkAvil[philID-1].taken = Philostatus[philID].left = 1;
                            printf("Fork %d taken by philosopher %d\n",philID,philID+1);
                        }else{
                            printf("Philosopher %d is waiting for fork %d\n",philID+1,philID);
                        }
                    }else{ //except last philosopher case
                        if(ForkAvil[philID].taken == 0){
                            ForkAvil[philID].taken = Philostatus[philID].left = 1;
                            printf("Fork %d taken by Philosopher %d\n",philID+1,philID+1);
                        }else{
                            printf("Philosopher %d is waiting for Fork %d\n",philID+1,philID+1);
                        }
                    }
        }else{}
}

int main(){
        for(i=0;i<n;i++)
        ForkAvil[i].taken=Philostatus[i].left=Philostatus[i].right=0;

        while(compltedPhilo<n){
                /* Observe here carefully, while loop will run until all philosophers complete dinner
                Actually problem of deadlock occur only thy try to take at same time
                This for loop will say that they are trying at same time. And remaining status will print by go for dinner function
                */
                for(i=0;i<n;i++)
            goForDinner(i);
                printf("\nTill now num of philosophers completed dinner are %d\n\n",compltedPhilo);
        }

        return 0;
}
```

Output:

```
anushka_os@DESKTOP-96L9A8G:~$ gcc philo.c -o philo
anushka_os@DESKTOP-96L9A8G:~$ ./philo
Fork 1 taken by Philosopher 1
Fork 2 taken by Philosopher 2
Fork 3 taken by Philosopher 3
Philosopher 4 is waiting for fork 3

Till now num of philosophers completed dinner are 0

Fork 4 taken by Philosopher 1
Philosopher 2 is waiting for Fork 1
Philosopher 3 is waiting for Fork 2
Philosopher 4 is waiting for fork 3

Till now num of philosophers completed dinner are 0

Philosopher 1 completed his dinner
Philosopher 1 released fork 1 and fork 4
Fork 1 taken by Philosopher 2
Philosopher 3 is waiting for Fork 2
Philosopher 4 is waiting for fork 3

Till now num of philosophers completed dinner are 1

Philosopher 1 completed his dinner
Philosopher 2 completed his dinner
Philosopher 2 released fork 2 and fork 1
Fork 2 taken by Philosopher 3
Philosopher 4 is waiting for fork 3

Till now num of philosophers completed dinner are 2

Philosopher 1 completed his dinner
Philosopher 2 completed his dinner
Philosopher 3 completed his dinner
Philosopher 3 released fork 3 and fork 2
Fork 3 taken by philosopher 4

Till now num of philosophers completed dinner are 3

Philosopher 1 completed his dinner
Philosopher 2 completed his dinner
Philosopher 3 completed his dinner
Fork 4 taken by philosopher 4

Till now num of philosophers completed dinner are 3

Philosopher 1 completed his dinner
Philosopher 2 completed his dinner
Philosopher 3 completed his dinner
Philosopher 4 completed his dinner
Philosopher 4 released fork 4 and fork 3
```

```
Philosopher 1 completed his dinner
Philosopher 2 completed his dinner
Philosopher 3 completed his dinner
Fork 4 taken by philosopher 4

Till now num of philosophers completed dinner are 3

Philosopher 1 completed his dinner
Philosopher 2 completed his dinner
Philosopher 3 completed his dinner
Philosopher 4 completed his dinner
Philosopher 4 released fork 4 and fork 3

Till now num of philosophers completed dinner are 4
```

USING SEMAPHORES

Code:

```c
#include<stdio.h>
#include<semaphore.h>
#include<pthread.h>

#define N 5
#define THINKING 0
#define HUNGRY 1
#define EATING 2
#define LEFT (ph_num+4)%N
#define RIGHT (ph_num+1)%N

sem_t mutex;
sem_t S[N];

void * philospher(void *num);
void take_fork(int);
void put_fork(int);
void test(int);

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

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,philospher,&phil_num[i]);
        printf("Philosopher %d is thinkingn",i+1);
    }
    for(i=0;i<N;i++)
        pthread_join(thread_id[i],NULL);
}

void *philospher(void *num)
{
    while(1)
    {
        int *i = num;
        sleep(1);
        take_fork(*i);
        sleep(0);
        put_fork(*i);
    }
```

```
}

void take_fork(int ph_num)
{
    sem_wait(&mutex);
    state[ph_num] = HUNGRY;
    printf("Philosopher %d is Hungryn",ph_num+1);
    test(ph_num);
    sem_post(&mutex);
    sem_wait(&S[ph_num]);
    sleep(1);
}

void test(int ph_num)
{
    if (state[ph_num] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING)
    {
        state[ph_num] = EATING;
        sleep(2);
        printf("Philosopher %d takes fork %d and %dn",ph_num+1,LEFT+1,ph_num+1);
        printf("Philosopher %d is Eatingn",ph_num+1);
        sem_post(&S[ph_num]);
    }
}

void put_fork(int ph_num)
{
    sem_wait(&mutex);
    state[ph_num] = THINKING;
    printf("Philosopher %d putting fork %d and %d downn",ph_num+1,LEFT+1,ph_num+1);
    printf("Philosopher %d is thinkingn",ph_num+1);
    test(LEFT);
    test(RIGHT);
    sem_post(&mutex);
}
```

Output:

```
Philosopher 1 is thinking
Philosopher 1 is eating
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 3 is eating
Philosopher 4 is thinking
Philosopher 5 is thinking
Philosopher 1 Finished eating
Philosopher 3 Finished eating
Philosopher 4 is eating
Philosopher 5 is eating
Philosopher 2 is eating
Philosopher 4 Finished eating
Philosopher 5 Finished eating
Philosopher 2 Finished eating
```

Deadlock Avoidance

Code:

```
#include <stdio.h>

int current[5][5], maximum_claim[5][5], available[5];
int allocation[5] = {0, 0, 0, 0, 0};
int maxres[5], running[5], safe = 0;
int counter = 0, i, j, exec, resources, processes, k = 1;

int main()
{
        printf("\nEnter number of processes: ");
  scanf("%d", &processes);

    for (i = 0; i < processes; i++)
        {
        running[i] = 1;
        counter++;
    }

    printf("\nEnter number of resources: ");
    scanf("%d", &resources);

    printf("\nEnter Claim Vector:");
    for (i = 0; i < resources; i++)
        {
                scanf("%d", &maxres[i]);
    }

   printf("\nEnter Allocated Resource Table:\n");
    for (i = 0; i < processes; i++)
        {
                for(j = 0; j < resources; j++)
                {
  scanf("%d", &current[i][j]);
        }
    }

    printf("\nEnter Maximum Claim Table:\n");
    for (i = 0; i < processes; i++)
        {
        for(j = 0; j < resources; j++)
                {
            scanf("%d", &maximum_claim[i][j]);
        }
    }

        printf("\nThe Claim Vector is: ");
    for (i = 0; i < resources; i++)
        {
```

```
    for (i = 0; i < resources; i++)
        {
                printf("\t%d", maxres[i]);
        }

    printf("\nThe Allocated Resource Table:\n");
    for (i = 0; i < processes; i++)
        {
                for (j = 0; j < resources; j++)
                {
            printf("\t%d", current[i][j]);
        }
                printf("\n");
    }

    printf("\nThe Maximum Claim Table:\n");
    for (i = 0; i < processes; i++)
        {
        for (j = 0; j < resources; j++)
                {
                        printf("\t%d", maximum_claim[i][j]);
        }
        printf("\n");
    }
```

```
for (i = 0; i < processes; i++)
    {
    for (j = 0; j < resources; j++)
            {
        allocation[j] += current[i][j];
    }
}

printf("\nAllocated resources:");
for (i = 0; i < resources; i++)
    {
    printf("\t%d", allocation[i]);
}

for (i = 0; i < resources; i++)
    {
            available[i] = maxres[i] - allocation[i];
    }

printf("\nAvailable resources:");
for (i = 0; i < resources; i++)
    {
    printf("\t%d", available[i]);
```

```
printf("\n");

while (counter != 0)
    {
    safe = 0;
    for (i = 0; i < processes; i++)
            {
        if (running[i])
                {
            exec = 1;
            for (j = 0; j < resources; j++)
                    {
                if (maximum_claim[i][j] - current[i][j] > available[j])
                            {
                    exec = 0;
                    break;
                }
            }
            if (exec)
                    {
                printf("\nProcess%d is executing\n", i + 1);
                running[i] = 0;
                counter--;
                safe = 1;

                for (j = 0; j < resources; j++)
                            {
                    available[j] += current[i][j];
                }
                            break;
            }
        }
    }
    if (!safe)
            {
        printf("\nThe processes are in unsafe state.\n");
        break;
    }
            else
            {
        printf("\nThe process is in safe state");
        printf("\nAvailable vector:");

        for (i = 0; i < resources; i++)
                {
```

```
            for (i = 0; i < resources; i++)
                        {
                    printf("\t%d", available[i]);
            }

                        printf("\n");
        }
    }
    return 0;
}
```

Output:

```
anushka_os@DESKTOP-96L9A8G:~$ ./banker

Enter number of processes: 3

Enter number of resources: 3

Enter Claim Vector:8 4 3

Enter Allocated Resource Table:
0 0 1
3 2 0
2 1 1

Enter Maximum Claim Table:
8 4 3
6 2 0
3 3 3

The Claim Vector is:    8       4       3
The Allocated Resource Table:
        0       0       1
        3       2       0
        2       1       1

The Maximum Claim Table:
        8       4       3
        6       2       0
        3       3       3

Allocated resources:    5       3       2
Available resources:    3       1       1

Process2 is executing

The process is in safe state
Available vector:       6       3       1

The processes are in unsafe state.
anushka_os@DESKTOP-96L9A8G:~$
```