COE18B001
Adhesh R

Operating Systems (COM301T & P)
End Assessment

PART-B

Q.1.

Given,

Speed up gained due to new CPU = 10

Fraction enhanced, fe = 40% = 0.4

[∵ Only the computation can be parallelized]

By Amdahl's Law,

$$\text{Overall speed up} = \frac{1}{(1 - Fe) + \left(\frac{Fe}{\text{Speed enhanced}}\right)}$$

$$= \frac{1}{(1-0.4) + \left(\frac{0.4}{10}\right)}$$

$$= \frac{1}{0.6 + 0.04} = \frac{1}{0.64}$$

$$= 1.5625$$

Q.2

According to the question

$$R = [3 \quad 2 \quad 3 \quad 2]$$

At time $t = 0$

P_1, P_2, P_3 will acquire 2units of R_2, 2units of R_3 and 1 unit of R_4 respectively without any problem.

So Available resources become

$$A = [3 \quad 2 \quad 3 \quad 2] - [0 \quad 2 \quad 2 \quad 1]$$

$$= [3 \quad 0 \quad 1 \quad 1]$$

At $t = 1$

Only P_1 requests 1 unit of R_3 which is available hence it acquires it

$$A = [3 \quad 0 \quad 1 \quad 1] - [0 \quad 0 \quad 1 \quad 0]$$

$$= [3 \quad 0 \quad 0 \quad 1]$$

At $t = 2$

P_2 requests for 1 unit of R_4

P_3 requests for 2 units of R_1 both of which are available.

Hence

$$A = [3 \quad 0 \quad 0 \quad 1] - [2 \quad 0 \quad 0 \quad 1]$$

$$= [1 \quad 0 \quad 0 \quad 0]$$

At $t = 3$:

P_1 requests 2 units of R_1 but we only have 1 unit of R_1 so this request cant be satisfied completely and P_1 will have to wait and it will blocked until 2 units of R_1 becomes available.

$$A = [1\ 0\ 0\ 0] \quad \text{(no allocation)}$$

At $t = 4$

P_2 requests 1 unit of R_1 which is available, hence P_2 acquires R_1.

$$A = [1\ 0\ 0\ 0] - [1\ 0\ 0\ 0] = [0\ 0\ 0\ 0]$$

At $t = 5$

P_3 releases 2 units of R_1.

$$A = [0\ 0\ 0\ 0] + [2\ 0\ 0\ 0] = [2\ 0\ 0\ 0]$$

The system will assign these two units of R_1 to P_1 and wake it up. So

$$A = [2\ 0\ 0\ 0] - [2\ 0\ 0\ 0] = [0\ 0\ 0\ 0]$$

P_1 has gotten its two units, so it will resume operation and execute next instruction at $t+2$ time.

At $t = 6$

P_2 releases one unit of R_3

$$A = [0\ 0\ 0\ 0] + [0\ 0\ 1\ 0]$$
$$= [0\ 0\ 1\ 0]$$

At $t = 7$

P_1 (running delayed) will release one unit of R_2 and 1 unit of R_1

$$A = [0\ 0\ 1\ 0] + [1\ 1\ 0\ 0]$$
$$= [1\ 1\ 1\ 0]$$

Also, P_3 requests 1 unit of R_2 (here even if P_3 executes before P_1, the block delay is negligible).

$$A = [1\ 1\ 1\ 0] - [0\ 1\ 0\ 0]$$
$$= [1\ 0\ 1\ 0]$$

At $t = 8$,

P_3 requests 1 unit of R_3. It is acquired.

P_2 will release all its resources. So

$$A = [1\ 0\ 1\ 0] - [0\ 0\ 1\ 0] + [1\ 0\ 1\ 1]$$
$$= [2\ 0\ 1\ 1]$$

At t = 9

P_3 will finish, and will release all its resources.

P_1 will release 1 unit of R_3. So

$$A = [2\ 0\ 1\ 1] + [0\ 1\ 1\ 1] + [0\ 0\ 1\ 0]$$

$$= [2\ 1\ 3\ 2]$$

At t = 10.

P_1 will acquire 2units of R_4 without problem

$$A = [2\ 1\ 3\ 2] - [0\ 0\ 0\ 2]$$

$$= [2\ 1\ 3\ 0]$$

At t = 12

P_1 will finish and release all resources

$$A = [2\ 1\ 3\ 0] + [1\ 1\ 0\ 2]$$

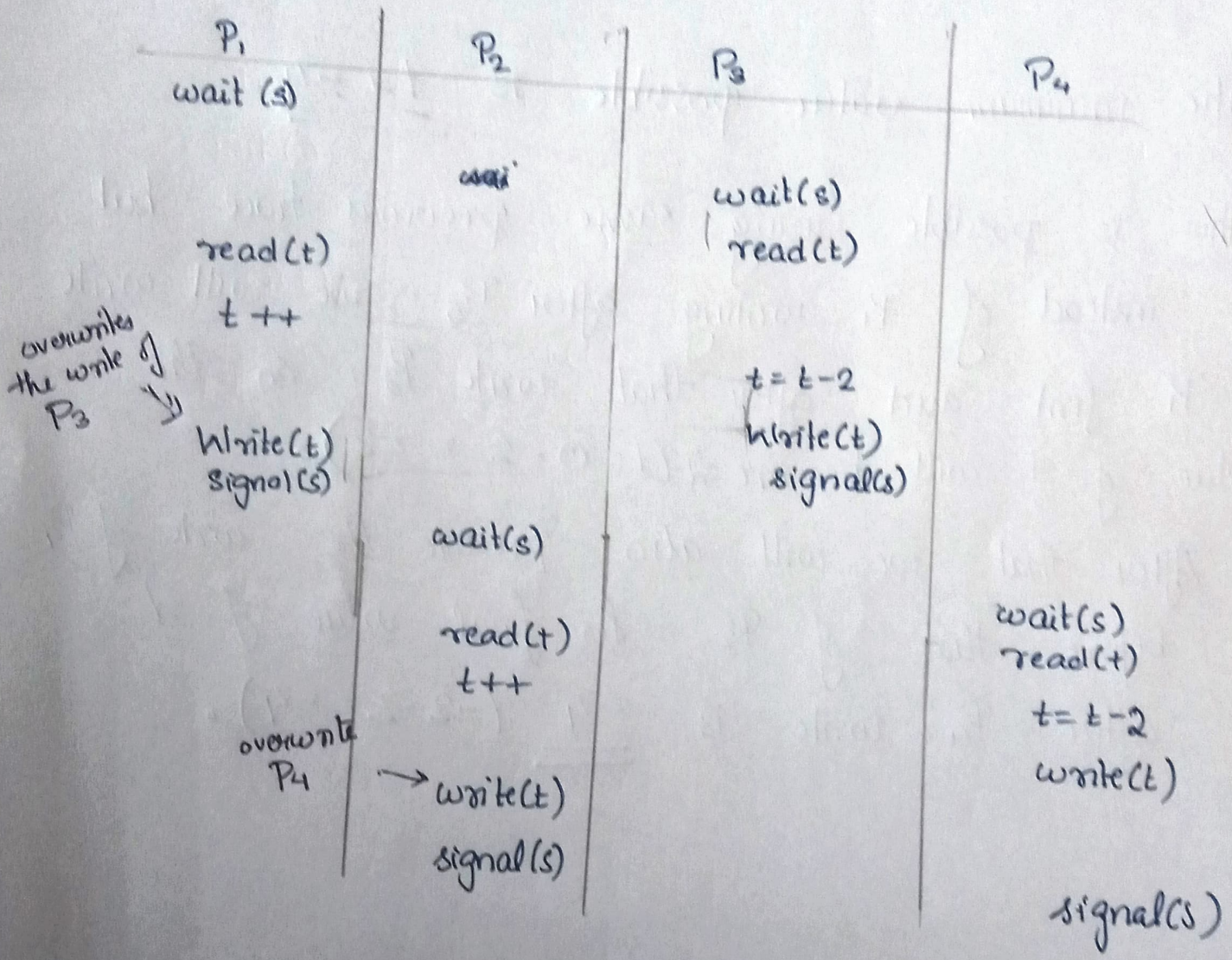
$$= [3\ 2\ 3\ 2]$$

Hence all process will execute/finish without any deadlock.

Q.3 * Assuming initial value of semaphore is 2. and t = 0.

The maximum possible value = 2

Trace to get max value is:



If the code executes as above, P_1 and P_3 will read the same value of t (t=0) and P_1 will increment it (t = t+1 = 1) and P_3 will decrement it (t = 0-2 = -2) and P_3 will write first and followed by P_1 which overwrites on the value written by P_3. So curr value of t is 1. Similarly P_2 and P_4 will now enter the CSP and read t=1.

P_2 will increment ($t = 1+1 = 2$) and P_4 will decrement ($t = 1-2 = -1$). P_4 will write first and P_2 after that. So final value becomes $\underline{t = 2}$.

The minimum value possible is $\underline{t = -4}$

This is possible using same previous trace but instead of P_1 writing after P_3, we will write P_1 first and after that write P_3. So the value of t will become -2 ($t = 0 - 2 = -2$).

After that we will also precede the write of P_4 before that of P_2. So final value of t after P_4's write is $\underline{-4}$ ($-2-2 = -4$).

Q.4

Dijkstra's Algorithm for Synchronization is:
(for n process setup)

Shared variables: b, c[1:N]; (initialized to true)
integer k;

Note: $1 \le k \le N$. b[i] and c[i] are set by P_i only whereas all other process can read them. Here, i, contains the process number and N total no of processes.

Local Variables: j (integer)

Protocol for Process P_i $(1 \le i \le N)$ is

```
Li0:  b[i] := false
Li1:  if k ≠ i then:
Li2:  begin  c[i] := true;
Li3:         if b[k] then k := i;
               goto Li1;
      end
Li4:  else begin:
            c[i] := false;
            for j := 1 step 1 until N do
                if (j ≠ i) and (not c[j]) then goto Li1;
      end;
      < C S >
       c[i] := true;
       b[i] := true;
      < Remainder >
       goto Li0;
```

The algorithm enters its C.S only when it finds all other c's true after having set its own c to false. Mutual exclusion is achived by this.

Let us assume, on contrary, two processes P_i and P_j are in their critical section simultaneously. To enter enter critical section, P_i must set c[i] to false and find c[j] to true. On other hand, P_j must have found c[i] to be true after setting c[j] to false. This leads to contradiction and hence assumption is wrong and mutual exclusion ensured.

The solution also avoids deadlock.

If the process P_k is not trying to enter the critical section, b[k] will be true and all other processes trying to enter C.S will find ($k \neq i$) true. As a result, several processes may several processes may execute the assignment stm $k = i$. After first assignment, no new process can assign a new value to k as they all will find b[k] false. Since k is shared, it will contain number of last processes, say i, to have had carried out k=i and will not change until b[i] becomes true. Now P_i will wait (in L4) until all

P.T.O

other processes set their c true, and then Pi will enter its critical section. Therefore, when none of the processes is in C.S, one process will only be able to do so. Hence no deadlock

This algorith however doesn't ensure ~~the same~~ fair ~~process~~ entry to Critical section. A process may get starved in this algorithm.

Part-C

Q.5

Contention scope → It refers to the scope in which threads compete for use of physical CPUs.

i) Process Contention Scope :→

→ In systems implementing many-to-one and many-to-many models.

→ Scheduling competition is within the process.

ii) System-Contention Scope :→

→ In one-one model

→ Involves system scheduler scheduling kernel threads to run on one or more CPUs.

→ Scheduling competition among all threads in system.

Q5.

Logic Used

In this we have to run three binaries in 3 different threads. The threads have the following attributes:

- Contention Scope: Process Contention Scope
- Scheduling policy: Round Robin

We can set these attributes to the thread using pthread commands:

To set the contention scope:

```
pthread_attr_setscope(&attr,PTHREAD_SCOPE_PROCESS);
```

To set the scheduling policy:

```
pthread_attr_setschedpolicy(&attr,SCHED_RR) !=0
```

Once the attributes are set, we can execute each of the binaries in the separate threads using the system() command.

Code

```
/*****************************************
* AUTHOR : AdheshR *
* Problem Description: Q5: Execute three binaries in threads using RR and PCS*
*****************************************/
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <pthread.h>
#define MAX 100
#define NUM_THREADS 3
```

```c
// Define the binaries to be executed
char *binaries[] = {"./binsearch","./matmul","./primegen"};

void *bin_runner(void *arg);

int main()
{
    int i,scope,policy;

    // define the threads
    pthread_t tid[NUM_THREADS];
    pthread_attr_t attr;

    // Get default attributes
    pthread_attr_init(&attr);

    // Inquire the scope
    if(pthread_attr_getscope(&attr, &scope) != 0)
       fprintf(stderr, "Unable to get scheduling scope.\n");
    else
    {
       if(scope == PTHREAD_SCOPE_PROCESS)
             printf("PTHREAD_SCOPE_PROCESS\n");
       else if(scope == PTHREAD_SCOPE_SYSTEM)
       {
             printf("PTHREAD_SCOPE_SYSTEM\n");
       }
       else
             fprintf(stderr, "Illegal scope value\n");
    }

    // Set scope to PCS if not already
    if(scope != PTHREAD_SCOPE_PROCESS)
       pthread_attr_setscope(&attr,PTHREAD_SCOPE_PROCESS);
    printf("scheduling scope set to PTHREAD_SCOPE_PROCESS\n");

    // Inquire the current scheduling policy
    if(pthread_attr_getschedpolicy(&attr,&policy) != 0)
       fprintf(stderr, "Unable to get the scheduling policy\n");
    else
    {
       if(policy == SCHED_OTHER)
             printf("SCHED_OTHER\n");
```

```
        else if(policy == SCHED_RR)
                printf("SCHED_RR\n");
        else if(policy == SCHED_FIFO)
                printf("SCHED_FIFO\n");
    }

    // Set scheduling policy to RR if not already
    if(policy != SCHED_RR)
    {
        // Set to RR
        if(pthread_attr_setschedpolicy(&attr,SCHED_RR) !=0)
                fprintf(stderr, "Unable to set scheduling policy\n");
        else
                printf("scheduling policy set to SCHED_RR\n");
    }

    // Create and call the threads
    for(i=0;i<NUM_THREADS;++i)
    {
        int *indx = malloc(sizeof(int));
        *indx = i;
        pthread_create(&tid[i],&attr, bin_runner,indx);
    }

    // Join all the threads after exit
    for(i=0;i<NUM_THREADS;++i)
        pthread_join(tid[i],NULL);

    return 0;
}

// Define the thread runners
void *bin_runner(void *arg)
{
    // Get the argument in appropriate type
    int indx = *((int *)arg);
    printf("%s\n",binaries[indx]);

    // Execute the appropriate binary using execlp call
    system(binaries[indx]);

    pthread_exit(NULL);
}
```

Code for the Binaries are:

Binary Search

```c
/****************************************
* AUTHOR : AdheshR *
* Problem Description: Binary Search*
****************************************/
#include <stdio.h>
#include <stdlib.h>
#define MAX 1000

int binarySearch(int arr[], int l, int r, int x) ;
int main(int argc, char *argv[])
{
    // Define the array and length and key x
    int arr[] = {1,2,3,4,5,6};
    int n = 6;
    int x = 6;
    int pos = binarySearch(arr,0,n-1,x);
    if(pos >= 0)
       printf("Position of %x in the array is %d\n",x,pos);
    else
       printf("%d not found in given array\n",x);
    return 0;
}

// Define the binary search function
int binarySearch(int arr[], int l, int r, int x)
{
      if (r >= l) {
      int mid = l + (r - l) / 2;

      if (arr[mid] == x)
            return mid;

      if (arr[mid] > x)
            return binarySearch(arr, l, mid - 1, x);

      return binarySearch(arr, mid + 1, r, x);
      }
      return -1;
}
```

Prime Generation upto n

```c
/*****************************************
* AUTHOR : AdheshR *
* Problem Description: Prime Number generation upto n.*
*****************************************/
#include <stdio.h>
#include <stdlib.h>
#define MAX 1000

void generatePrime(int n);
int isPrime(int num);

int main(int argc, char *argv[])
{
    // Define range
    int n = 13;

    // call function to generate prime upto n
    generatePrime(n);

    return 0;
}

// Define primeGenerate() function
void generatePrime(int n)
{
    int i=1;
    for(;i<=n;++i)
    {
       if(isPrime(i) == 1)
             printf("Prime: %d\n",i);
    }
}

// Utiltity function to check if the given number is prime or not
int isPrime(int num)
{
    if(num == 1)
       return 0;
    if(num == 2 || num == 3)
       return 1;
```

```
    int i = 2,flag = 0;
    for(i=2; i< num/2 +1;++i)
    {
       if(num%i == 0)
       {
             ++flag;
             break;
       }
    }

    if(flag == 0)
       return 1;
    else
       return 0;
}
```

Matrix Multiplication

```
/*****************************************
* AUTHOR : AdheshR *
* Problem Description: Matrix Multiplication*
*****************************************/
#include <stdio.h>
#include <stdlib.h>
#define MAX 1000
#define N 4

void matrixMul(int matA[][N], int matB[][N], int matC[][N]);

int main(int argc, char *argv[])
{
    // Define two matrices of N*N
    int matA[N][N] = { { 1, 1, 1, 1 },
                       { 2, 2, 2, 2 },
                       { 3, 3, 3, 3 },
                       { 4, 4, 4, 4 } };

      int matB[N][N] = { { 1, 1, 1, 1 },
                       { 2, 2, 2, 2 },
                       { 3, 3, 3, 3 },
                       { 4, 4, 4, 4 } };
```

```
      // Define result matrix
      int product[N][N];
      matrixMul(matA, matB, product);

      // Print the product matrix
      int i,j;
      for(i=0;i<N;++i)
      {
       for(j=0;j<N;++j)
             printf("%d ",product[i][j]);
       puts("");
      }

    return 0;
}

// Define matrix multiplication function
void matrixMul(int matA[][N], int matB[][N], int product[][N])
{
    int i,j,k;
    for (i = 0; i < N; i++) {
      for (j = 0; j < N; j++) {
            product[i][j] = 0;
            for (k = 0; k < N; k++)
                  product[i][j] += matA[i][k] * matB[k][j];
      }
      }
}
```

Output of main.c

```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Endsem/Q_5$ ./out
PTHREAD_SCOPE_SYSTEM
scheduling scope set to PTHREAD_SCOPE_PROCESS
SCHED_OTHER
scheduling policy set to SCHED_RR
./matmul
./binsearch
./primegen
Position of 6 in the array is 5
10 10 10 10
Prime: 2
20 20 20 20
Prime: 3
30 30 30 30
Prime: 5
40 40 40 40
Prime: 7
Prime: 11
Prime: 13
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Endsem/Q_5$
```

```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Endsem/Q_5$ ./out
PTHREAD_SCOPE_SYSTEM
scheduling scope set to PTHREAD_SCOPE_PROCESS
SCHED_OTHER
scheduling policy set to SCHED_RR
./binsearch
./matmul
./primegen
Prime: 2
Prime: 3
Prime: 5
Prime: 7
Prime: 11
Prime: 13
Position of 6 in the array is 5
10 10 10 10
20 20 20 20
30 30 30 30
40 40 40 40
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Endsem/Q_5$
```

Q.6

Priority Inversion Problem :→

When a high priority process gets blocked ~~is~~ waiting for a resource that is currently held by a low-priority process.

Example:

Consider three processes L, M and H.

The priority of the processes are

$$L < M < H$$

Let us consider the situation when L holds the mutex and working in the critical section. Process H is waiting for this mutex held by L.

In the meanwhile M arrives and pre-empts L.

The waiting time of H is prolonged by process M which has lower priority causing priority inversion.

Real-word example : Mars Path finder. In this case the processes were

L : ~~meteorological~~ meteorological data gathering

M : long-running communicatn tasks

H : bus mgmt tasks

→ It was noticed that bus task (H) did not execute for sometime causing total system reset. It was later found out to be case of priority inversion.

Solution to Priority Inversion Problem :→

Priority Inheritance

Acc
In this solution, the low priority process holding a resource for which a high-priority process is waiting will temporarily inherit the high priority from waiting process.

This prevents the medium-priority processes from preempting the low-priority process until it releases the resource blocking the priority inversion problem.

In terms of L, M, H

→ L runs with priority of H
→ M comes and H is waiting
→ H would get resource ahead of M maintaing the priority.

Q6.

Part I

Solution to Priority Inversion Problem - Implementation of a version of priority inheritance

Logic Used

In this we are supposed to implement a solution to PIP. For this I have considered three thread processes L, M H having priorities 1, 2 and 3 respectively. For this I have used a modified version of the Bounded Wait TSL program for n processes. Each thread enters the critical section in the same way but the difference lies in the bounded wait exit portion. A thread checks if there are other higher priority threads waiting and if there is, it sets the waiting[j] = 0, thereby allowing that process to enter the CS next. If no higher priority threads are waiting, then it releases the lock and proceeds to the remainder section.

Code

```
/*****************************************
* AUTHOR : AdheshR *
* Problem Description: Priority Inheritance solution implemented along with Bounded Wait TSL solution for n process - Here in the bounded wait condition check any process first unlocks
                                                highest priority process if it wants to enter the CS. Else it unlocks any other waiting process. If no process is waiting it just releases the lock*
*****************************************/
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <inttypes.h>
#include <fcntl.h>
#define MAX 100
#define NUM_THREADS 3
#define L_PRIORITY 1
#define M_PRIORITY 2
#define H_PRIORITY 3
```

```c
// Define the priority for three processes
int priority[] = {L_PRIORITY,M_PRIORITY,H_PRIORITY};

// Define the shared variable to access in CS
int data = 0;
int key = 0;
int lock = 0;
int waiting[] = {0,0,0};

int TestNSet(int *target);
void *runner(void *arg);

int main()
{
    int i;

    // Define three threads - L, M, H
    pthread_t tid[NUM_THREADS];

    // create threads and call runners
    for(i=0;i<NUM_THREADS;++i)
    {
       int *indx = malloc(sizeof(int));
       *indx = i;
       pthread_create(&tid[i],NULL,runner,indx);
    }

    // Join all threads
    for(i=0;i<NUM_THREADS;++i)
       pthread_join(tid[i],NULL);

    return 0;
}

// Define the thread to access CS
void *runner(void *arg)
{
    // Get the argument in appropriate type
    int indx = *((int *)arg);
    int j;

    while(1)
    {
```

```
	printf("\nProcess:%d is WAITING...\n",indx+1);

	/* -------- Entry Section ----------- */
	waiting[indx] = 1;
	key = 1;
	while(waiting[indx] && key)
		key = TestNSet(&lock);
	waiting[indx] = 0;

	/* ---------- Critical section -------------- */
	printf("\n%d\n",priority[indx]);
	printf("Process:%d is CRITICAL SECTION...\n",indx+1);
	data++;
	printf("Updated Data: %d\n",data);

	/* ---------- Exit section - Is based on allowing higher priority wait
run first-------------- */
	j = (indx + 1)%NUM_THREADS;

	if(indx == 0)
	{
		if(waiting[2] == 1)
			waiting[2] = 0;
		else if(waiting[1] == 1)
			waiting[1] = 0;
		else
			lock = 0;
	}
	else if(indx == 1)
	{
		if(waiting[2] == 1)
			waiting[2] = 0;
		else if(waiting[0] == 1)
			waiting[0] = 0;
		else
			lock = 0;
	}
	else if(indx == 2)
	{
		if(waiting[1] == 1)
			waiting[1] = 0;
		else if(waiting[0] == 1)
			waiting[0] = 0;
```

```
            else
                  lock = 0;
      }

      // Remainder section
      sleep(4);
    }

}

// Test N Set
int TestNSet(int *target)
{
    int rv = *target;
    *target = 1;
    return rv;
}
```

Output

```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Endsem/Q_6$ gcc -o out priorityInheritance.c -lpthread
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Endsem/Q_6$ ./out

Process:1 is WAITING...

1
Process:1 is CRITICAL SECTION...
Updated Data: 1

Process:2 is WAITING...

2
Process:2 is CRITICAL SECTION...
Updated Data: 2

Process:3 is WAITING...

3
Process:3 is CRITICAL SECTION...
Updated Data: 3

Process:1 is WAITING...

1
Process:1 is CRITICAL SECTION...
Updated Data: 4

Process:2 is WAITING...

2
Process:2 is CRITICAL SECTION...
Updated Data: 5

Process:3 is WAITING...

3
Process:3 is CRITICAL SECTION...
Updated Data: 6

Process:1 is WAITING...

1
Process:1 is CRITICAL SECTION...
Updated Data: 7

Process:2 is WAITING...

2
Process:2 is CRITICAL SECTION...
Updated Data: 8

Process:3 is WAITING...

3
Process:3 is CRITICAL SECTION...
Updated Data: 9

Process:1 is WAITING...

1
Process:1 is CRITICAL SECTION...
Updated Data: 10
```

```
2
Process:2 is CRITICAL SECTION...
Updated Data: 8

Process:3 is WAITING...

3
Process:3 is CRITICAL SECTION...
Updated Data: 9

Process:1 is WAITING...

1
Process:1 is CRITICAL SECTION...
Updated Data: 10

Process:2 is WAITING...

2
Process:2 is CRITICAL SECTION...
Updated Data: 11

Process:3 is WAITING...

3
Process:3 is CRITICAL SECTION...
Updated Data: 12

Process:1 is WAITING...

1
Process:1 is CRITICAL SECTION...

Process:3 is WAITING...
Updated Data: 13

3
Process:3 is CRITICAL SECTION...
Updated Data: 14

Process:2 is WAITING...

2
Process:2 is CRITICAL SECTION...
Updated Data: 15

Process:1 is WAITING...

Process:2 is WAITING...

1
Process:1 is CRITICAL SECTION...
Updated Data: 16

2
Process:2 is CRITICAL SECTION...
Updated Data: 17

Process:3 is WAITING...

3
Process:3 is CRITICAL SECTION...
Updated Data: 18
^C
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Endsem/Q_6$
```

Part II - Include Priority Inversion Solution in Dining Philosopher

Logic Used

We can include it in the Test() function where a philosopher tests to see if the neighbors are not eating but are hungry and have higher priority then forfeit the chance to eat and give way to the other philosopher.

Code

```
/*****************************************
* AUTHOR : AdheshR *
* Problem Description: Implement the solution for the dining philosopher problem using semaphores. Include Priority Inheritance in Dining Philosopher Problem. We can include it in the Test() function
                              where a philosopher tests to see if the neighbors are not eating but hungry and have higher priority - forefit chance to give way to higher phil*
*****************************************/
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <inttypes.h>
#include <pthread.h>
#include <semaphore.h>
#define MAX 100

// Define problem macros
#define N 5
#define THINKING 0
#define HUNGRY 1
#define EATING 2

#define LEFT (phnum + 4)%N
#define RIGHT (phnum + 1)%N

// Define sempahores
sem_t mutex;
sem_t S[N];

// Define shared variables
```

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

// Define the priority of each philosopher
int priority[] = {1,2,3,4,5};

void *philosopher(void *num);
void take_fork(int phnum);
void put_fork(int phnum);
void test(int phnum);

int main()
{
    int i;

    pthread_t thread_id[N];

    // Init the semaphore
    sem_init(&mutex, 0,1);

    for(i=0;i<N;++i)
       sem_init(&S[i],0,0);

    // create and call thread runners
    for(i=0;i<N;++i)
    {
       pthread_create(&thread_id[i], NULL, philosopher, &phil_num[i]);
       printf("Philosopher %d is thinking.\n",i+1);
    }

    for(i=0;i<N;++i)
       pthread_join(thread_id[i],NULL);
}

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

```
    }
}

// Take chopsticks up
void take_fork(int phnum)
{
    sem_wait(&mutex);

    // move to HUNGRY state
    state[phnum] = HUNGRY;
    printf("Philosopher %d is hungry.\n",phnum);

    // Eait if neighbors arent eating
    test(phnum);

    sem_post(&mutex);

    // If unable to eat wait
    sem_wait(&S[phnum]);

    sleep(1);
}

// Put down chopstick
void put_fork(int phnum)
{
    sem_wait(&mutex);

    // move to state of THINKING
    state[phnum] = THINKING;
    printf("Philosopher %d putting fork %d and %d down\n",phnum + 1, LEFT + 1,
phnum + 1);
      printf("Philosopher %d is thinking\n", phnum + 1);

      test(LEFT);
      test(RIGHT);

      sem_post(&mutex);
}

void test(int phnum)
{
      if (state[phnum] == HUNGRY && state[LEFT] != EATING && state[RIGHT] !=
```

```
EATING)
      {
       // Additional priority check - check if the left neighbor or the right
neighbor is HUNGRY and has a higher priority - then forfeit turn to EAT
       if((priority[LEFT] > priority[phnum - 1] && state[LEFT] == HUNGRY) ||
(priority[RIGHT] > priority[phnum - 1] && state[RIGHT] == HUNGRY))
             sem_post(&S[phnum]);
       else
       {
             // state that eating
            state[phnum] = EATING;

            sleep(2);

            printf("Philosopher %d takes fork %d and %d\n", phnum + 1, LEFT + 1,
phnum + 1);
            printf("Philosopher %d is Eating\n", phnum + 1);

            sem_post(&S[phnum]);
       }
      }
}
```

Output

```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Endsem/Q_6$ gcc -o out diningPhil.c -lpthread
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Endsem/Q_6$ ./out
Philosopher 1 is thinking.
Philosopher 2 is thinking.
Philosopher 3 is thinking.
Philosopher 4 is thinking.
Philosopher 5 is thinking.
Philosopher 0 is hungry.
Philosopher 1 takes fork 5 and 1
Philosopher 1 is Eating
Philosopher 1 is hungry.
Philosopher 2 is hungry.
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
Philosopher 3 is hungry.
Philosopher 4 is hungry.
Philosopher 1 putting fork 5 and 1 down
Philosopher 1 is thinking
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
Philosopher 2 takes fork 1 and 2
Philosopher 2 is Eating
Philosopher 0 is hungry.
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 2 is hungry.
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
Philosopher 1 takes fork 5 and 1
Philosopher 1 is Eating
Philosopher 4 is hungry.
Philosopher 4 putting fork 3 and 4 down
Philosopher 4 is thinking
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
Philosopher 1 is hungry.
Philosopher 1 putting fork 5 and 1 down
Philosopher 1 is thinking
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 3 is hungry.
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
Philosopher 2 takes fork 1 and 2
Philosopher 2 is Eating
Philosopher 0 is hungry.
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 2 is hungry.
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
Philosopher 1 takes fork 5 and 1
Philosopher 1 is Eating
Philosopher 4 is hungry.
Philosopher 4 putting fork 3 and 4 down
Philosopher 4 is thinking
^C
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Endsem/Q_6$
```

Note: The solution at the moment does not specifically allow only neighbor with higher priority but rather current philosopher forfeits if it sees any higher prior philosopher in the wait queue.