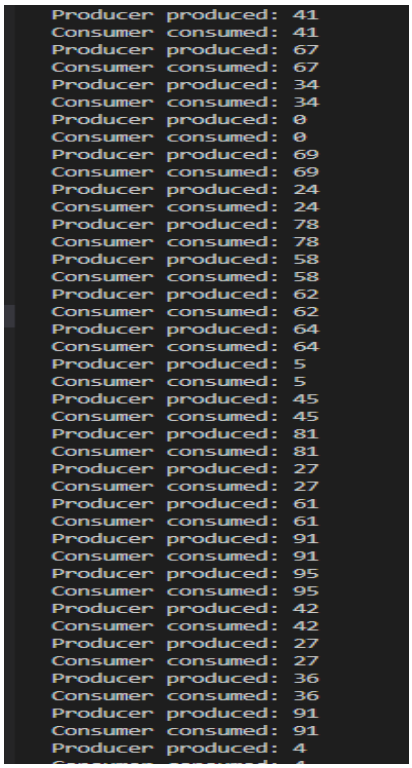
OS LAB4 – 17/04/2025

1. **Write a C program to stimulate Producer-Consumer problem using semaphores**

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

#define SIZE 5

int buffer[SIZE];
int in = 0, out = 0;

sem_t empty, full;
pthread_mutex_t mutex;

void* producer(void* arg) {
   int item;
   while (1) {
      item = rand() % 100;
      sem_wait(&empty);
      pthread_mutex_lock(&mutex);

      buffer[in] = item;
      printf("Producer produced: %d\n", item);
      in = (in + 1) % SIZE;

      pthread_mutex_unlock(&mutex);
      sem_post(&full);

      sleep(1);
   }
}

void* consumer(void* arg) {
   int item;
   while (1) {
      sem_wait(&full);
      pthread_mutex_lock(&mutex);

      item = buffer[out];
      printf("Consumer consumed: %d\n", item);
      out = (out + 1) % SIZE;

      pthread_mutex_unlock(&mutex);
      sem_post(&empty);
```

```
        sleep(1);
    }
}

int main() {
    pthread_t prod, cons;

    sem_init(&empty, 0, SIZE);
    sem_init(&full, 0, 0);
    pthread_mutex_init(&mutex, NULL);

    pthread_create(&prod, NULL, producer, NULL);
    pthread_create(&cons, NULL, consumer, NULL);

    pthread_join(prod, NULL);
    pthread_join(cons, NULL);

    sem_destroy(&empty);
    sem_destroy(&full);
    pthread_mutex_destroy(&mutex);

    return 0;
}
```

OUTPUT:



2. Write a C program to stimulate Dining-Philosopher's problem

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

#define N 5

pthread_mutex_t chopstick[N];

void* philosopher(void* num) {
   int id = *(int*)num;

   while (1) {
      printf("Philosopher %d is thinking...\n", id);
      sleep(1);

      printf("Philosopher %d is hungry.\n", id);

      pthread_mutex_lock(&chopstick[id]);
      pthread_mutex_lock(&chopstick[(id + 1) % N]);

      printf("Philosopher %d is eating...\n", id);
      sleep(2);

      pthread_mutex_unlock(&chopstick[id]);
      pthread_mutex_unlock(&chopstick[(id + 1) % N]);

      printf("Philosopher %d has finished eating and puts down chopsticks.\n", id);
      sleep(1);
   }

   return NULL;
}

int main() {
   pthread_t threads[N];
   int ids[N];

   for (int i = 0; i < N; i++) {
      pthread_mutex_init(&chopstick[i], NULL);
   }

   for (int i = 0; i < N; i++) {
      ids[i] = i;
      pthread_create(&threads[i], NULL, philosopher, &ids[i]);
```

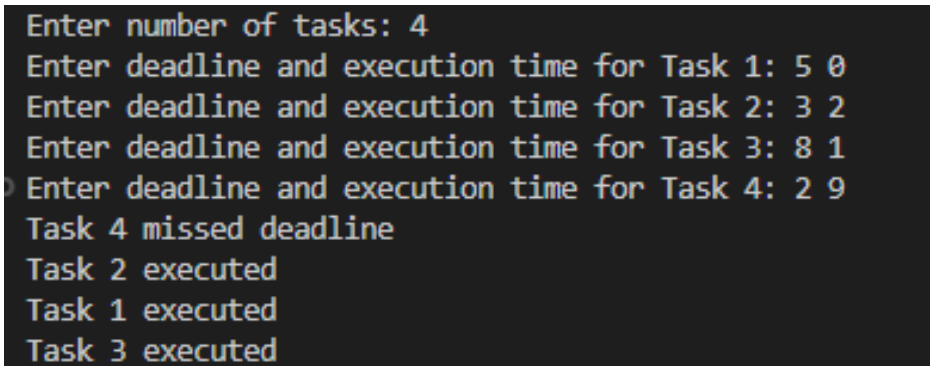
```
    }

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

    for (int i = 0; i < N; i++) {
        pthread_mutex_destroy(&chopstick[i]);
    }

    return 0;
}
```

OUTPUT:

3. **Write a C program to stimulate Real-time CPU Scheduling algorithems for Earliest-deadline First**

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

typedef struct {
   int id, deadline, exec_time;
} Task;

int cmp(const void *a, const void *b) {
   return ((Task *)a)->deadline - ((Task *)b)->deadline;
}

void edf(Task tasks[], int n) {
   qsort(tasks, n, sizeof(Task), cmp); // Sort by deadline
   int time = 0;
   for (int i = 0; i < n; i++) {
      if (time + tasks[i].exec_time <= tasks[i].deadline) {
         time += tasks[i].exec_time;
         printf("Task %d executed\n", tasks[i].id);
      } else {
         printf("Task %d missed deadline\n", tasks[i].id);
      }
   }
}

int main() {
   int n;
   printf("Enter number of tasks: ");
   scanf("%d", &n);

   Task tasks[n];
   for (int i = 0; i < n; i++) {
      printf("Enter deadline and execution time for Task %d: ", i + 1);
      tasks[i].id = i + 1;
      scanf("%d %d", &tasks[i].deadline, &tasks[i].exec_time);
   }

   edf(tasks, n);
   return 0;
}
```

OUTPUT:


Enter number of tasks: 4
Enter deadline and execution time for Task 1: 5 0
Enter deadline and execution time for Task 2: 3 2
Enter deadline and execution time for Task 3: 8 1
Enter deadline and execution time for Task 4: 2 9
Task 4 missed deadline
Task 2 executed
Task 1 executed
Task 3 executed