NAME : K DHANUSH

REGISTER NUMBER : 192311094

COURSE : OPERATING SYSTEMS FRO MOBILE APPLICATIONS

COURSE CODE : CSA0497

1.ROUND ROBIN SCHEDULING

```
#include <stdio.h> #define
MAX 10
void findWai ngTime(int processes[], int n, int bt[], int wt[], int quantum) {
int rem_bt[MAX];
   for (int i = 0; i < n; i++)
rem_bt[i] = bt[i];     int t = 0;
while (1) {        int done = 1;
for (int i = 0; i < n; i++) {            if
(rem_bt[i] > 0) {               done =
0;               if (rem_bt[i] >
quantum) {                  t +=
quantum;                  rem_bt[i] -=
quantum;
            } else {
t = t + rem_bt[i];
wt[i] = t - bt[i];
rem_bt[i] = 0;
```

```
                }
            }
        }
        if (done == 1)
break;
    }
}
void findTurnAroundTime(int processes[], int n, int bt[], int wt[], int tat[]) {
    for (int i = 0; i < n; i++)
tat[i] = bt[i] + wt[i];
}
void findavgTime(int processes[], int n, int bt[], int quantum) {
int wt[MAX], tat[MAX];     findWaitingTime(processes, n, bt,
wt, quantum);     findTurnAroundTime(processes, n, bt, wt,
tat);

    float total_wt = 0, total_tat = 0;
    for (int i = 0; i < n; i++) {
total_wt += wt[i];         total_tat
+= tat[i];
    }
    printf("Average waiting time: %.2f\n", total_wt / n);     printf
("Average turnaround time: %.2f\n", total_tat / n);
}
```

```
int main() {     int processes[] = { 0, 1, 2, 3 };
int n = sizeof(processes) / sizeof(processes[0]);
int burst_time[] = { 10, 5, 8, 12 };     int quantum
= 4;     findavgTime(processes, n, burst_time,
quantum);
   return 0;
}
```

OUTPUT :

```
Average waiting time: 19.25
Average turnaround time: 28.00
```

2.INTER-PROCESS COMMUNICATION

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define MAX_TEXT 512
struct message {     long
msg_type;     char
text[MAX_TEXT];
};
```

```
int main() {     struct
message msg;
   int msgid;
   key_t key = 1234;     msgid =
msgget(key, 0666 | IPC_CREAT);
   if (msgid == -1) {
perror("msgget failed");
exit(EXIT_FAILURE);
   }
   msg.msg_type = 1;     snprintf(msg.text, sizeof(msg.text), "Hello from
process %d", getpid());

   if (msgsnd(msgid, &msg, sizeof(msg.text), 0) == -1)
{        perror("msgsnd failed");
exit(EXIT_FAILURE);
   }
   printf("Message sent: %s\n", msg.text);
   return 0;
}
```

OUTPUT :

```
Message sent: Hello from process 16920
```

3.DINING-PHILOSOPHERS PROBLEM

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

```
#include <semaphore.h>

#include <unistd.h>

#define NUM_PHILOSOPHERS 5


sem_t mutex; sem_t

forks[NUM_PHILOSOPHERS];


void* philosopher(void* num) {

    int id = *(int*)num;


    while (1) {

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

        sleep(1);


        sem_wait(&mutex);

sem_wait(&forks[id]);

        sem_wait(&forks[(id + 1) %
NUM_PHILOSOPHERS]);


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

        sleep(1);


        sem_post(&forks[id]);

        sem_post(&forks[(id + 1) %

NUM_PHILOSOPHERS]);        sem_post(&mutex);
```

```
    }
}

int main() {
    pthread_t philosophers[NUM_PHILOSOPHERS];
int philosopher_ids[NUM_PHILOSOPHERS];

    sem_init(&mutex, 0, 1);     for (int i = 0; i <
NUM_PHILOSOPHERS; i++) {
        sem_init(&forks[i], 0, 1);
philosopher_ids[i] = i;
        pthread_create(&philosophers[i], NULL, philosopher,
&philosopher_ids[i]);
    }

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

    for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
sem_destroy(&forks[i]);
    }
    sem_destroy(&mutex);

    return 0;
}
```

OUTPUT :

```
Philosopher 0 is thinking.

Philosopher 2 is thinking.

Philosopher 1 is thinking.

Philosopher 3 is thinking.

Philosopher 4 is thinking.

Philosopher 3 is eating.

Philosopher 3 is thinking.

Philosopher 4 is eating.

Philosopher 3 is eating.

Philosopher 4 is thinking.
```

4. BANKER’S ALGORITHEM

```
#include <stdio.h>


#define MAX 10

#define RESOURCES 3


int main() {

   int max[MAX][RESOURCES], allot[MAX][RESOURCES], need[MAX][RESOURCES];

   int available[RESOURCES], finish[MAX], safeSeq[MAX];

   int n, m, i, j, k, count = 0;
```

```
    printf("Enter number of processes: ");
scanf("%d", &n);     printf("Enter number
of resources: ");     scanf("%d", &m);


    printf("Enter maximum resource matrix:\n");
    for (i = 0; i < n; i++) {
for (j = 0; j < m; j++) {
scanf("%d", &max[i][j]);
        }
    }


    printf("Enter allocation matrix:\n");
    for (i = 0; i < n; i++) {
for (j = 0; j < m; j++) {
scanf("%d", &allot[i][j]);
        }
    }


    printf("Enter available resources:\n");
    for (i = 0; i < m; i++) {
scanf("%d", &available[i]);
    }
    for (i = 0; i < n; i++) {         for (j =
0; j < m; j++) {             need[i][j] =
max[i][j] - allot[i][j];
        }
    }

    for (i = 0; i < n; i++) {
finish[i] = 0;
    }

    while (count < n) {         int found =
0;         for (i = 0; i < n; i++) {
if (finish[i] == 0) {                 for (j = 0;
j < m; j++) {                     if (need[i][j]
> available[j]) {
                        break;
                    }
}
                if (j == m) {                     for
(k = 0; k < m; k++) {
available[k] += allot[i][k];
                    }
                    safeSeq[count++] = i;
                    finish[i] = 1;
found = 1;
```

```
                }
            }
        }
        if (found == 0) {
            printf("System is not in a safe state\n");
            return 0;
        }
    }

    printf("System is in a safe state.\nSafe sequence is: ");
    for (i = 0; i < n; i++) {
        printf("%d ", safeSeq[i]);
    }
    printf("\n");

    return 0;
}
```

OUTPUT :

```
Enter allocation matrix:
Enter available resources: System
is not in a safe state
Enter number of processes:
```

5.PRODUCER CONSUMER PROBLEM

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

```
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#define BUFFER_SIZE 5

int buffer[BUFFER_SIZE];
int in = 0;
int out = 0;

sem_t empty;
sem_t full;
pthread_mutex_t mutex;

void* producer(void* arg) {
    for (int i = 0; i < 10; i++) {
        sem_wait(&empty);
pthread_mutex_lock(&mutex);

        buffer[in] = i;
        printf("Produced: %d\n", buffer[in]);
in = (in + 1) % BUFFER_SIZE;

        pthread_mutex_unlock(&mutex);
        sem_post(&full);
    }
    return NULL;
}
void* consumer(void* arg) {
    for (int i = 0; i < 10; i++) {
sem_wait(&full);
        pthread_mutex_lock(&mutex);

        int item = buffer[out];
        printf("Consumed: %d\n", item);
out = (out + 1) % BUFFER_SIZE;

        pthread_mutex_unlock(&mutex);
sem_post(&empty);
    }
    return NULL;
}

int main() {     pthread_t
prod, cons;

    sem_init(&empty, 0, BUFFER_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 :

```
Produced: 0
Produced: 1
Produced: 2
Produced: 3
Produced: 4
Consumed: 0
Consumed: 1
Consumed: 2
Consumed: 3
Consumed: 4
```