EXPERIMENT 8
DINING PHILOSOPHER PROBLEM

AIM
Implement Dining philosophers problem program in C.

ALGORITHM
STEP 0: START
STEP 1: Include all the necessary header files required for program
STEP 2: Define all constants -
N - No of philosophers and forks
LEFT & RIGHT - Represent the index of left & right neighbours of a philosopher
THINKING, HUNGRY & EATING - constants to represent the states of a philosopher
STEP 3: Declare the global variables
state[N] - Array to store state of each philosopher
pthread_t t[N] - To store ID of each philosopher.
sem_t s[N] - semaphore representing forks
sem_t mutex - semaphore to access the critical section.
STEP 4: END

ALGORITHM FOR THINK()
STEP 0: START
STEP 1: Print which philosopher is thinking for 1 second
STEP 2: END

ALGORITHM FOR EAT()
STEP 0: START
STEP 1: Print which philosopher is eating for 1 second
STEP 2: Print which philosopher has finished eating
STEP 3: END

ALGORITHM FOR TAKEFORK()
STEP 0: START
STEP 1: sem_wait(&mutex) is initialized to lock mutex to enter critical section
STEP 2: set state[i] = HUNGRY
STEP 3: If state[i] == HUNGRY & state[LEFT] != EATING & state[RIGHT] != EATING
i) set state[i] = EATING
ii) sem_wait(&s[LEFT]) & sem_wait(&s[RIGHT])
iii) unlock mutex to exit critical section by initializing sem_post(&mutex)
STEP 4: END

ALGORITHM FOR PUTFORKS()
STEP 0: START
STEP 1: Set state[i] = THINKING
STEP 2: sem-post(&s[LEFT]) &
sem-post(&s[RIGHT]) to release both left & right forks after eating.
STEP 3: END

ALGORITHM FOR *PHILO()
STEP 0: START
STEP 1: Set ~~THINK~~ (n)
STEP 2: Call TAKEFORK(n)
STEP 3: If state[n] == EATING, then
i) set EAT(n)
ii) PUTFORKS(n)
STEP 4: END

ALGORITHM FOR MAIN
STEP 0: START
STEP 1: Using a for loop, initialize semaphores for forks & mutex semaphore
STEP 2: Create threads for philosophers
STEP 3: Infinite loop to handle deadlock & prevent circular wait.