CEN354 Operating Systems - Assignment #3:

The Dining Philosophers Problem

The Dining Philosopher Problem states that K philosophers seated around a circular table with one chopstick between each pair of philosophers. There is one chopstick between each philosopher. A philosopher may eat if he can pickup the two chopsticks adjacent to him. One chopstick may be picked up by any one of its adjacent followers but not both.

#include <stdio.h> → For printing
#include <unistd.h> → For sleep() function
#include <pthread.h> → For threads and mutexes
#include <string.h> → For strcmp()
#include <stdlib.h> → For rand()

`pthread_mutex_t lock;` → Mutual exclusion lock for locking the shared resources.

`pthread_cond_t cond[5];` → Declare condition variables of the philosophers.

`enum philosopher_state { THINKING, HUNGRY, EATING };` → Philosophers alternate between Thinking and Eating. Also adding Hungry as a middle state is useful for condition checks.

```
void *chow_line(void *nphilosopher) {

while(TRUE) {
    int *no_philosopher = nphilosopher;
    sleep( 1 + (rand() % 2));

    printf("Philosopher %d is THINKING.\n", *no_philosopher + 1);
    if(passed_argc > 1){
if(strcmp(passed_argv[1], "--with-time") == 0)
          sleep(1+(rand()%2));
    }
    state[*no_philosopher] = HUNGRY;
    printf("Philosopher %d is HUNGRY.\n", *no_philosopher + 1);
    pickup_forks(*no_philosopher);
    putdown_forks(*no_philosopher);
  }

}
```

→In this part, Before performing operations, make sleep every philosopher for a random time for making output more readable. If the main thread is called with --with-time argument, make philosopher thinking for a random number of time between 1 and 3. Thinking operation is finished, try to pick up forks/chopsticks. Eating operation is finished, release the forks/chopsticks.

```
void pickup_forks(int no_philosopher) {
  pthread_mutex_lock(&lock);
  control(no_philosopher);       while (state[no_philosopher] != EATING)
pthread_cond_wait(&cond[no_philosopher], &lock);
  pthread_mutex_unlock(&lock);
}
```

→In pickup_forks function, philosophers are in HUNGRY state and takes their chances to eat. Lock the pickup_forks function in order to pretend mutual exclusion. If left and right neighbors are not eating, philosopher begins eating. If they are eating, waits them for releasing the forks/chopsticks.If philosopher couldn't started eating, wait until the lock is unlocked. And Release the pickup_forks function.

```
void putdown_forks(int no_philosopher) {
  pthread_mutex_lock(&lock);
  state[no_philosopher] = THINKING;
  control((no_philosopher + 4) % 5);
control((no_philosopher + 1) % 5);
  pthread_mutex_unlock(&lock);
}
```

→ Lock the putdown_forks function in order to pretend mutual exclusion. Philosopher finished eating and started thinking again. Make left neighbor eating if she is hungry and Make right neighbor eating if she is hungry.After than Release the putdown_forks function.

```
void control(int no_philosopher) {
  if (state[no_philosopher] == HUNGRY   && state[(no_philosopher + 4) % 5] != EATING
&& state[(no_philosopher + 1) % 5] != EATING)
{
      state[no_philosopher] = EATING;
      printf("Philosopher %d is EATING.\n", no_philosopher + 1);
      if(passed_argc > 1){
if(strcmp(passed_argv[1], "--with-time") == 0)
        sleep(1+(rand()%2));
      }
      pthread_cond_signal(&cond[no_philosopher]);
```

→Purpose of the this part,checking current philosopher wants eating.

ULAŞ RENAS ORDU - 2016556047
2nd Education