COMPSCI 3SH3 Fall, 2022
By: Nathan Agbomedarho
Student ID: 400081762
Date: November 20 2022

Lab 4 Report

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

#define MAX_PHILOSOPHER 5
#define MAX_SLEEP_TIME 3

enum{THINKING, HUNGRY, EATING};
int state[MAX_PHILOSOPHER];
int philosopher_id[MAX_PHILOSOPHER];

pthread_cond_t condition_variables[MAX_PHILOSOPHER];
pthread_mutex_t mutex;
pthread_t thread_id[MAX_PHILOSOPHER];

void init()
{

    for (int i = 0; i < MAX_PHILOSOPHER; i++)
    {
        state[i] = THINKING;
        philosopher_id[i] = i;
        pthread_cond_init(&condition_variables[i], NULL);
    }
    pthread_mutex_init(&mutex, NULL);
}

void eat(int eating_time, int philosopherNumber)
{
    printf("Philosopher %d EATING \n", philosopherNumber);
    sleep(eating_time);
}

void think(int thinking_time, int philosopherNumber)

{
    printf("Philosopher %d is THINKING \n", philosopherNumber);
```

```
    sleep(thinking_time);
}
void *philosopherStart(void *id)
{
    int *x = (int *)id;
    int philosopherNumber = *x;
    int sleep_time;
    srandom((unsigned)time(NULL));
    while (z < 5)
    {
        sleep_time = (int)((random() % MAX_SLEEP_TIME + 1));
        think(sleep_time, philosopherNumber);
        pickup_forks(philosopherNumber);
        sleep_time = (int)((random() % MAX_SLEEP_TIME + 1));
        eat(sleep_time, philosopherNumber);
        return_forks(philosopherNumber);
        z+=1;
    }
}

int left_neighbor(int n)
{
    if (n == 0)
    {
        return MAX_PHILOSOPHER - 1;
    }
    else
```

```
    {
        return n - 1;
    }
}

int right_neighbor(int n)
{
    if (n == MAX_PHILOSOPHER - 1)
    {
        return 0;
    }
    else
    {
        return n + 1;
    }
}

void check_philosopher(int i)
{
    if (state[left_neighbor(i)] != EATING && state[right_neighbor(i)] != EATING && state[i]

    {
        state[i] = EATING;
        pthread_cond_signal(&condition_variables[i]);
    }
}

void pickup_forks(int i)
{

    pthread_mutex_lock(&mutex);

    state[i] = HUNGRY;

    check_philosopher(i);

    while (state[i] != EATING)

    {

        pthread_cond_wait(&condition_variables[i], &mutex);

    }

    pthread_mutex_unlock(&mutex);
```

```
}

void return_forks(int i)

{
    pthread_mutex_lock(&mutex);

    state[i] = THINKING;

    check_philosopher(left_neighbor(i));

    check_philosopher(right_neighbor(i));

    pthread_mutex_unlock(&mutex);

}


int main()

{
    init();

    for (int i = 0; i < MAX_PHILOSOPHER; i++)

    {

        pthread_create(&thread_id[i], 0, philosopherStart, (void *)&philosopher_id[i]);

    }

    for (int i = 0; i < MAX_PHILOSOPHER; i++)

    {

        pthread_join(thread_id[i], NULL);

    }

}
```

This program simulates the dining philosopher problem. The init function initializes the synchronization of the data structures, and sets the thinking state of the philosophers. The eat and think function, operate as a simulation of their respective actions, for a certain amount of time. The philosopher start function then initializes the threads for each philosopher. Pickup forks and return forks,

call the check philosopher function, which checks to the left and right of each philosopher, and then execute the action if the check is successful, then releases the mutex lock. Once the simulation is complete the threads are joined and the program exits.