COMPSCI 3SH3 Winter, 2021
Student name: Khizar Siddiqui
Student ID: 400109902
Date: 19-March-2021

Lab 4 Report

The Dining-Philosophers Problem

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

enum {THINKING, HUNGRY, EATING} state [5];
pthread_mutex_t forks;
pthread_cond_t condVar [ 5 ];
pthread_t philosopher [ 5 ];
int identity [ 5 ] = { 0, 1, 2, 3, 4 };

void *philo ( void *arg );
void *pickup_forks ( int philo_num );
void *return_forks ( int philo_num );
void *test ( int i );

int main () {
        pthread_mutex_init ( &forks, NULL );
        for ( int i = 0; i < 5; i++ ) {
                state [ i ] = THINKING;
                pthread_cond_init ( &condVar [ i ], NULL );
        }
        for ( int i = 0; i < 5; i++ ) {
                pthread_create ( &philosopher [ i ], NULL, philo, &identity [i]);
        }

        for ( int i = 0; i < 5; i++ ) {
                pthread_join ( philosopher [ i ], NULL );
                printf ( "Thread of Philosopher %d finished executing \n", i+1 );
```

```
        }
        return 0;
}

void *philo ( void *arg ) {
        int *tmp_ptr = ( int *)arg;
        int id = *tmp_ptr;
        printf ("Philosopher %d is THINKING \n",id + 1);

        while (1) {
                int time1 = ( rand())%3 + 1;
                sleep ( time1 );
                pickup_forks ( id );

                int time2 = ( rand())%3 + 1;
                sleep ( time2 );
                return_forks ( id );
        }
        return NULL;
}

void *pickup_forks (int id) {
        pthread_mutex_lock(&forks);
        state [id] = HUNGRY;
    printf ("Philosopher %d is HUNGRY \n", id + 1);
        pthread_mutex_unlock(&forks);
        test(id);
        pthread_mutex_lock(&forks);
        if (state [id] != EATING) {
                pthread_cond_wait(&condVar[id], &forks);
        }
        pthread_mutex_unlock(&forks);
        return NULL;
}

void *return_forks ( int id ) {
        pthread_mutex_lock(&forks);
        state [ id ] = THINKING;
        printf ("Philosopher %d returned forks \n", id + 1);
        pthread_mutex_unlock (&forks);
        test ((id + 4) % 5);
        test ((id + 1) % 5);
}

void *test ( int id ) {
        int left = (id + 4) % 5;
```

```
        int right = (id + 1) % 5;
        pthread_mutex_lock(&forks);
        if ((state[left] != EATING)
        && (state[right] != EATING)
        && (state[id] == HUNGRY)) {
                state[ id ] = EATING;
                printf("Philosopher %d takes forks \n", id + 1);
                pthread_cond_signal(&condVar[id]);
        }
        pthread_mutex_unlock(&forks);
}
```

The main function initializes the condition variable, mutex and creates the threads. The functions for each thread involve a single philosopher where they wait a random time before picking up forks and then waits before returning them with the use of a mutex for the forks.