// Implementation of classical problems (dining philosopher) using threads & semaphore

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

#define NUM\_PHILOSOPHERS 5

// Semaphores for each fork

sem\_t forks[NUM\_PHILOSOPHERS];

// Philosopher function

void \*philosopher(void \*num) {

int id = \*(int \*)num;

// Each philosopher thinks, picks up forks, eats, and then puts down forks

while (1) {

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

sleep(1);

// Pick up left fork

sem\_wait(&forks[id]);

printf("Philosopher %d picked up fork %d (left fork).\n", id, id);

// Pick up right fork

sem\_wait(&forks[(id + 1) % NUM\_PHILOSOPHERS]);

printf("Philosopher %d picked up fork %d (right fork).\n", id, (id + 1) % NUM\_PHILOSOPHERS);

// Eating

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

sleep(2);

// Put down right fork

sem\_post(&forks[(id + 1) % NUM\_PHILOSOPHERS]);

printf("Philosopher %d put down fork %d (right fork).\n", id, (id + 1) % NUM\_PHILOSOPHERS);

// Put down left fork

sem\_post(&forks[id]);

printf("Philosopher %d put down fork %d (left fork).\n", id, id);

// Thinking

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

sleep(1);

}

return NULL;

}

int main() {

pthread\_t philosophers[NUM\_PHILOSOPHERS];

int philosopher\_ids[NUM\_PHILOSOPHERS];

// Initialize semaphores (one for each fork)

for (int i = 0; i < NUM\_PHILOSOPHERS; i++) {

sem\_init(&forks[i], 0, 1);

}

// Create philosopher threads

for (int i = 0; i < NUM\_PHILOSOPHERS; i++) {

philosopher\_ids[i] = i;

pthread\_create(&philosophers[i], NULL, philosopher, &philosopher\_ids[i]);

}

// Join philosopher threads

for (int i = 0; i < NUM\_PHILOSOPHERS; i++) {

pthread\_join(philosophers[i], NULL);

}

// Destroy semaphores

for (int i = 0; i < NUM\_PHILOSOPHERS; i++) {

sem\_destroy(&forks[i]);

}

return 0;

}