#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <time.h>

#include <stdbool.h>

#include <string.h>

#include <sys/wait.h>

#include <pthread.h>

#define STUDENT\_FILE "database.txt"

#define TA\_COUNT 5

#define REPEAT\_COUNT 3

typedef struct {

int id; // TA ID

int current\_student; // Current student index

int cycles; // Number of completed cycles

bool finished; // Whether the TA is done marking

} TA;

typedef struct {

volatile int value; // Semaphore value

pthread\_mutex\_t lock;

} semaphore;

// Semaphore initialization

void semaphore\_init(semaphore \*sem, int initial\_value) {

sem->value = initial\_value;

pthread\_mutex\_init(&sem->lock, NULL);

}

// Semaphore wait (P operation)

void semaphore\_wait(semaphore \*sem) {

while (1) {

pthread\_mutex\_lock(&sem->lock);

if (sem->value > 0) {

sem->value--; // Decrement semaphore value

pthread\_mutex\_unlock(&sem->lock);

return;

}

pthread\_mutex\_unlock(&sem->lock);

sched\_yield(); // Yield the CPU to allow other threads/processes to run

}

}

// Semaphore signal (V operation)

void semaphore\_signal(semaphore \*sem) {

pthread\_mutex\_lock(&sem->lock); // Lock the mutex to safely increment the semaphore value

sem->value++; // Increment semaphore value

pthread\_mutex\_unlock(&sem->lock); // Unlock the mutex

}

// Helper function to create a student file

void create\_student\_file() {

FILE \*file = fopen(STUDENT\_FILE, "w");

if (!file) {

perror("Failed to create student file");

exit(1);

}

for (int i = 1; i <= 19; i++) {

fprintf(file, "%04d\n", i);

}

fprintf(file, "9999\n");

fclose(file);

}

// Sleep for a random duration and print activity

void sleep\_random(int max\_time, int ta\_id, const char \*activity) {

int delay = rand() % max\_time + 1;

printf("TA%d is %s (%d seconds).\n", ta\_id, activity, delay);

sleep(delay);

}

// Save marked student data to a file

void save\_to\_file(int ta\_id, int student, int mark) {

char filename[16];

sprintf(filename, "TA%d.txt", ta\_id);

FILE \*file = fopen(filename, "a");

if (!file) {

perror("Failed to open TA file");

exit(1);

}

fprintf(file, "Student: %04d, Mark: %d\n", student, mark);

fclose(file);

}

void marking(int ta\_id, const char \*student){

// Mark the student

int mark = rand() % 11; // Random mark between 0 and 10

save\_to\_file(ta\_id, atoi(student), mark);

char marking[50];

sprintf(marking, "marking student %s", student);

// Simulate marking

sleep\_random(8, ta\_id, marking);

printf("TA%d finished marking student %s.\n", ta\_id, student);

}

// Shared data

volatile int current\_index = 0; // Shared index for the student list

semaphore sem[TA\_COUNT]; // Custom semaphores

int main() {

// Create student file

create\_student\_file();

// Initialize custom semaphores

for (int i = 0; i < TA\_COUNT; i++) {

semaphore\_init(&sem[i], 1); // Initialize each TA's semaphore

}

// Initialize TA array

TA tas[TA\_COUNT];

for (int i = 0; i < TA\_COUNT; i++) {

tas[i].id = i + 1;

tas[i].current\_student = 0;

tas[i].cycles = 0;

tas[i].finished = false;

}

// Fork processes for TAs

for (int i = 0; i < TA\_COUNT; i++) {

if (fork() == 0) {

// Child process (TA)

srand(time(NULL) ^ getpid()); // Seed random number generator

TA \*ta = &tas[i];

while (!ta->finished) {

// Lock semaphores

semaphore\_wait(&sem[ta->id - 1]);

semaphore\_wait(&sem[ta->id % TA\_COUNT]); // (j+1) mod TA\_COUNT

// Simulate accessing the database

printf("Semaphores %d and %d is locked. ",ta->id, (ta->id) % (TA\_COUNT)+1);

sleep\_random(4, ta->id, "accessing the database");

// Access the database

FILE \*db = fopen(STUDENT\_FILE, "r");

if (!db) {

perror("Failed to open student file");

exit(1);

}

fseek(db, current\_index \* 5, SEEK\_SET); // 5 = 4 digits + newline

char student[5];

fscanf(db, "%s", student);

fclose(db);

if (strcmp(student, "9999") == 0) {

current\_index = 0; // Reset index to start

ta->cycles++;

if (ta->cycles == REPEAT\_COUNT) {

ta->finished = true;

marking(ta->id, student);

semaphore\_signal(&sem[ta->id - 1]);

semaphore\_signal(&sem[ta->id % TA\_COUNT]);

break; // Exit the marking loop

}

} else {

current\_index++; // Move to the next student

}

// Release semaphores

semaphore\_signal(&sem[ta->id % TA\_COUNT]);

semaphore\_signal(&sem[ta->id - 1]);

printf("Release Semaphores %d and %d. ",ta->id, (ta->id) % (TA\_COUNT)+1);

marking(ta->id, student);

}

printf("TA%d finished marking.\n", ta->id);

exit(0); // Exit child process

}

}

// Wait for all child processes to finish

for (int i = 0; i < TA\_COUNT; i++) {

wait(NULL);

}

printf("All TAs have finished marking.\n");

return 0;

}