**Задачи**

Задача о парикмахере

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

#define NUM\_SEATS 5

sem\_t customers;

sem\_t barber;

pthread\_mutex\_t mutex;

int waiting = 0;

void\* barber\_thread(void\* arg) {

while (1) {

sem\_wait(&customers);

pthread\_mutex\_lock(&mutex);

waiting--;

printf("Barber: taking a customer. Customers waiting: %d\n", waiting);

sem\_post(&barber);

pthread\_mutex\_unlock(&mutex);

printf("Barber: cutting hair...\n");

sleep(3);

}

}

void\* customer\_thread(void\* arg) {

pthread\_mutex\_lock(&mutex);

if (waiting < NUM\_SEATS) {

waiting++;

printf("Customer: sitting in the waiting room. Customers waiting: %d\n", waiting);

sem\_post(&customers);

pthread\_mutex\_unlock(&mutex);

sem\_wait(&barber);

printf("Customer: leaving after the haircut.\n");

} else {

printf("Customer: leaving, no free seats.\n");

pthread\_mutex\_unlock(&mutex);

}

return NULL;

}

int main() {

pthread\_t barber\_tid;

pthread\_t customer\_tids[15];

int i;

sem\_init(&customers, 0, 0);

sem\_init(&barber, 0, 0);

pthread\_mutex\_init(&mutex, NULL);

pthread\_create(&barber\_tid, NULL, barber\_thread, NULL);

for (i = 0; i < 15; i++) {

pthread\_create(&customer\_tids[i], NULL, customer\_thread, NULL);

sleep(1);

}

for (i = 0; i < 15; i++) {

pthread\_join(customer\_tids[i], NULL);

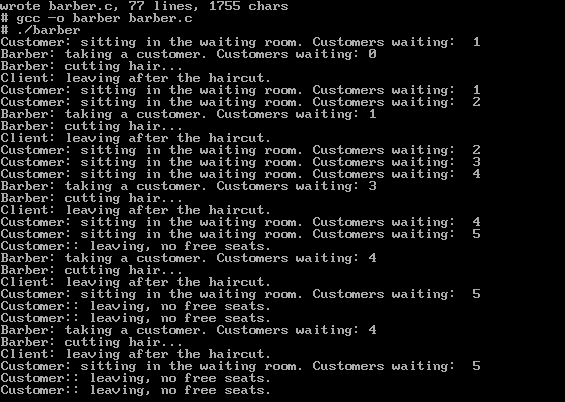
}

pthread\_join(barber\_tid, NULL);

return 0;

}

}



Задача о читателях и писателях

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

sem\_t resource\_access; // Semaphore for resource access (writer)

sem\_t read\_count\_access; // Semaphore for access to reader count

int read\_count = 0; // Number of readers

void\* reader(void\* arg) {

int reader\_id = \*((int\*)arg);

while (1) {

// Reader requests access to the reader count

sem\_wait(&read\_count\_access);

read\_count++;

if (read\_count == 1) {

// The first reader blocks access to the writer

sem\_wait(&resource\_access);

}

sem\_post(&read\_count\_access);

// Reading data

printf("Reader %d: reading data\n", reader\_id);

sleep(2);

// Reader finishes reading

sem\_wait(&read\_count\_access);

read\_count--;

if (read\_count == 0) {

// The last reader unlocks the resource for the writer

sem\_post(&resource\_access);

}

sem\_post(&read\_count\_access);

sleep(1); // Wait before trying to read again

}

}

void\* writer(void\* arg) {

int writer\_id = \*((int\*)arg);

while (1) {

// Writer requests access to the resource

sem\_wait(&resource\_access);

// Writing data

printf("Writer %d: writing data\n", writer\_id);

sleep(3);

// Writer finishes writing

sem\_post(&resource\_access);

sleep(2); // Wait before trying to write again

}

}

int main() {

pthread\_t readers[5], writers[3];

int reader\_ids[5], writer\_ids[3];

int i; // Declare variable i outside the loop

// Initialize semaphores

sem\_init(&resource\_access, 0, 1);

sem\_init(&read\_count\_access, 0, 1);

// Create reader threads

for (i = 0; i < 5; i++) {

reader\_ids[i] = i + 1;

pthread\_create(&readers[i], NULL, reader, &reader\_ids[i]);

}

// Create writer threads

for (i = 0; i < 3; i++) {

writer\_ids[i] = i + 1;

pthread\_create(&writers[i], NULL, writer, &writer\_ids[i]);

}

// Wait for all reader threads to finish

for (i = 0; i < 5; i++) {

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

}

// Wait for all writer threads to finish

for (i = 0; i < 3; i++) {

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

}

return 0;

}

