МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ



**Дніпровський національний університет  
залізничного транспорту імені академіка В. Лазаряна**

Кафедра «Комп’ютерні інформаційні технології»

**Лабораторна робота №2**

**з дисципліни «Операційні системи»**

**на тему: «Об'єкти ядра. Очікування потоків і процесів. Робота з бар'єрами.»**

Виконав: студент гр. ПЗ1911

Сіньков Г.О.

Прийняла: ас. каф. КІТ

Адрющенко В. О.

Дніпро, 2020

**Лабораторна робота №2**

**Тема.** Об'єкти ядра. Очікування потоків і процесів. Робота з бар'єрами.

**Завдання.** Розробити програми, які моделюють рішення задачі синхронізації з використанням заданих системних ресурсів. Під час запуску процесів передавати параметри через командний рядок та через змінні оточення програми.

**Варіант 12.**

1) Читачі-письменики.

2) Mutex.

3) Різним процесам.

**Текст програми**

main.cpp

#include<pthread.h>

#include<bits/stdc++.h>

#include<semaphore.h>

#include<stdio.h>

using namespace std;

void\* reader(void\*);

void\* writer(void\*);

// общие переменные для потоков

int shared\_var = 10, r\_counter = 0;

sem\_t rcnt; // семафор, используемый при обновлении r\_counter

sem\_t rdwr; //семафоры, используемые для проверки того, разрешено ли чтение/запись в критическую секцию или нет.

sem\_t addt; // дополнительный семафор, чтобы избежать приоритета для читателей и избежать голодания.

// Хранение идентификаторов потоков при условии, что не более 20 читателей и 5 авторов.

pthread\_t r[20], w[5];

//Читатель

void\* reader(void\* i) {

//ENTRY\_SECTION

//ожидание выхода другого читателя/писателя

sem\_wait(&addt);

printf("Reader-%d is in the ENTRY\_SECTION\n", i);

sem\_wait(&rcnt);

r\_counter++;

if (r\_counter == 1)

sem\_wait(&rdwr); //подождите, если писатель пишет

sem\_post(&rcnt);

sem\_post(&addt);

//CRITICAL\_SECTION

printf("Reader-%d is reading the value = %d\n", i, shared\_var);

//EXIT\_SECTION

printf("Reader-%d is in the EXIT\_SECTION\n", i);

sem\_wait(&rcnt);

r\_counter--;

if (r\_counter == 0)

sem\_post(&rdwr);

sem\_post(&rcnt);

}

// WRITER

void\* writer(void\* i) {

//ENTRY\_SECTION

//подождите, пока выполняющийся в данный момент читатель завершит обновление потока r\_counter или писателя, чтобы закончить

sem\_wait(&addt);

printf("writer-%d is in the ENTRY\_SECTION\n", i);

// ждем, если какой-то читатель читает/ пишет писатель

sem\_wait(&rdwr);

//CRITICAL\_SECTION

shared\_var += 15;

printf("writer-%d is writing value= %d \n", i, shared\_var);

//EXIT\_SECTION

printf("writer-%d is in the EXIT\_SECTION\n", i);

sem\_post(&rdwr);

sem\_post(&addt);

}

int main() {

// инициализация семафоров

sem\_init(&rcnt, 0, 1);

sem\_init(&rdwr, 0, 1);

sem\_init(&addt, 0, 1);

//потоки

pthread\_create(&r[0], NULL, reader, (void\*)0);

pthread\_create(&r[1], NULL, reader, (void\*)1);

pthread\_create(&r[2], NULL, reader, (void\*)2);

pthread\_create(&r[3], NULL, reader, (void\*)3);

pthread\_create(&w[0], NULL, writer, (void\*)0);

pthread\_create(&r[4], NULL, reader, (void\*)4);

pthread\_create(&r[5], NULL, reader, (void\*)5);

pthread\_create(&r[6], NULL, reader, (void\*)6);

pthread\_create(&w[1], NULL, writer, (void\*)1);

pthread\_create(&w[2], NULL, writer, (void\*)2);

pthread\_create(&r[7], NULL, reader, (void\*)7);

pthread\_create(&r[8], NULL, reader, (void\*)8);

pthread\_create(&r[9], NULL, reader, (void\*)9);

pthread\_create(&r[10], NULL, reader, (void\*)10);

pthread\_create(&r[11], NULL, reader, (void\*)11);

pthread\_create(&r[12], NULL, reader, (void\*)12);

pthread\_create(&r[13], NULL, reader, (void\*)13);

pthread\_create(&w[3], NULL, writer, (void\*)3);

pthread\_create(&r[14], NULL, reader, (void\*)14);

pthread\_create(&r[15], NULL, reader, (void\*)15);

pthread\_create(&r[16], NULL, reader, (void\*)16);

pthread\_create(&r[17], NULL, reader, (void\*)17);

pthread\_create(&r[18], NULL, reader, (void\*)18);

pthread\_create(&r[19], NULL, reader, (void\*)19);

pthread\_create(&w[4], NULL, writer, (void\*)4);

// ждем завершения потоков

pthread\_join(r[0], NULL);

pthread\_join(r[1], NULL);

pthread\_join(r[2], NULL);

pthread\_join(r[3], NULL);

pthread\_join(w[0], NULL);

pthread\_join(r[4], NULL);

pthread\_join(r[5], NULL);

pthread\_join(r[6], NULL);

pthread\_join(w[1], NULL);

pthread\_join(w[2], NULL);

pthread\_join(r[7], NULL);

pthread\_join(r[8], NULL);

pthread\_join(r[9], NULL);

pthread\_join(r[10], NULL);

pthread\_join(r[11], NULL);

pthread\_join(r[12], NULL);

pthread\_join(r[13], NULL);

pthread\_join(w[3], NULL);

pthread\_join(r[14], NULL);

pthread\_join(r[15], NULL);

pthread\_join(r[16], NULL);

pthread\_join(r[17], NULL);

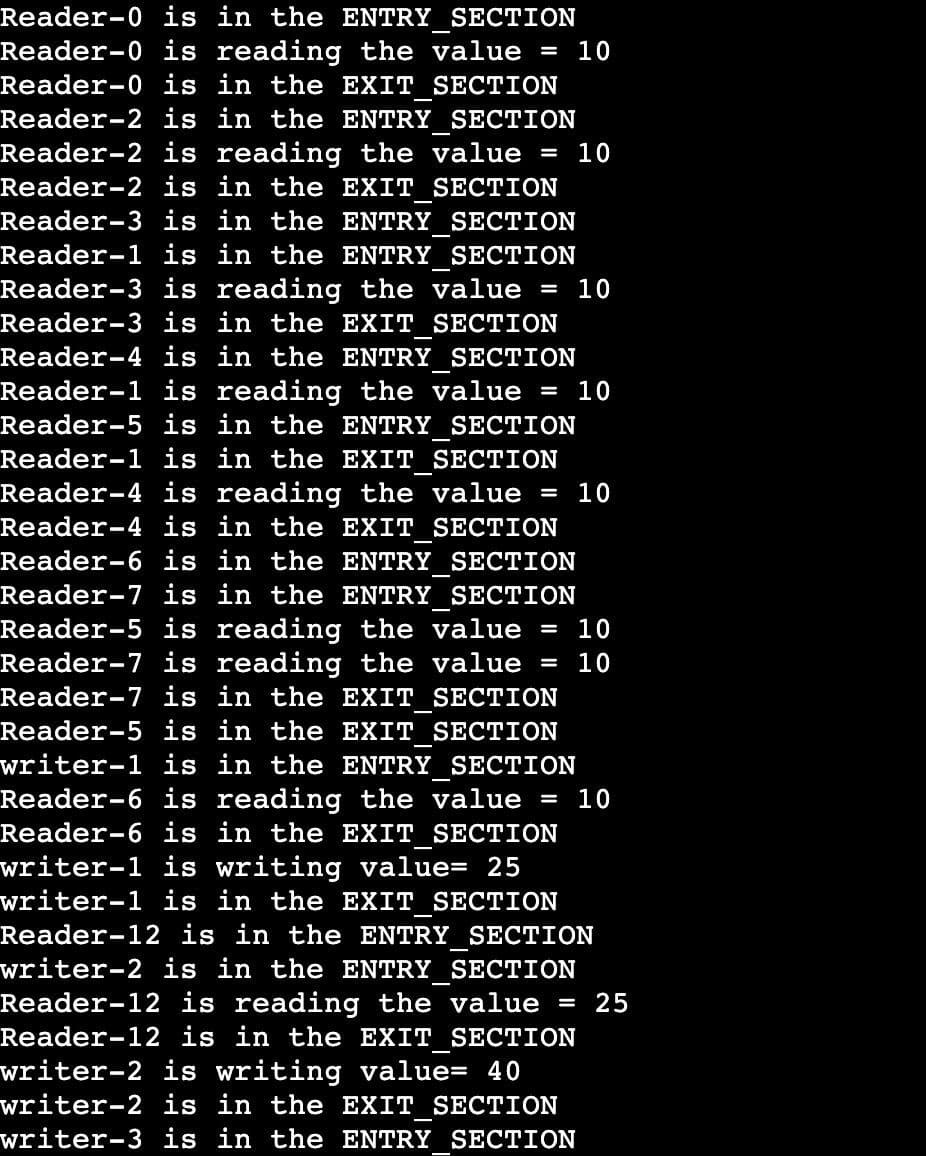
pthread\_join(r[18], NULL);

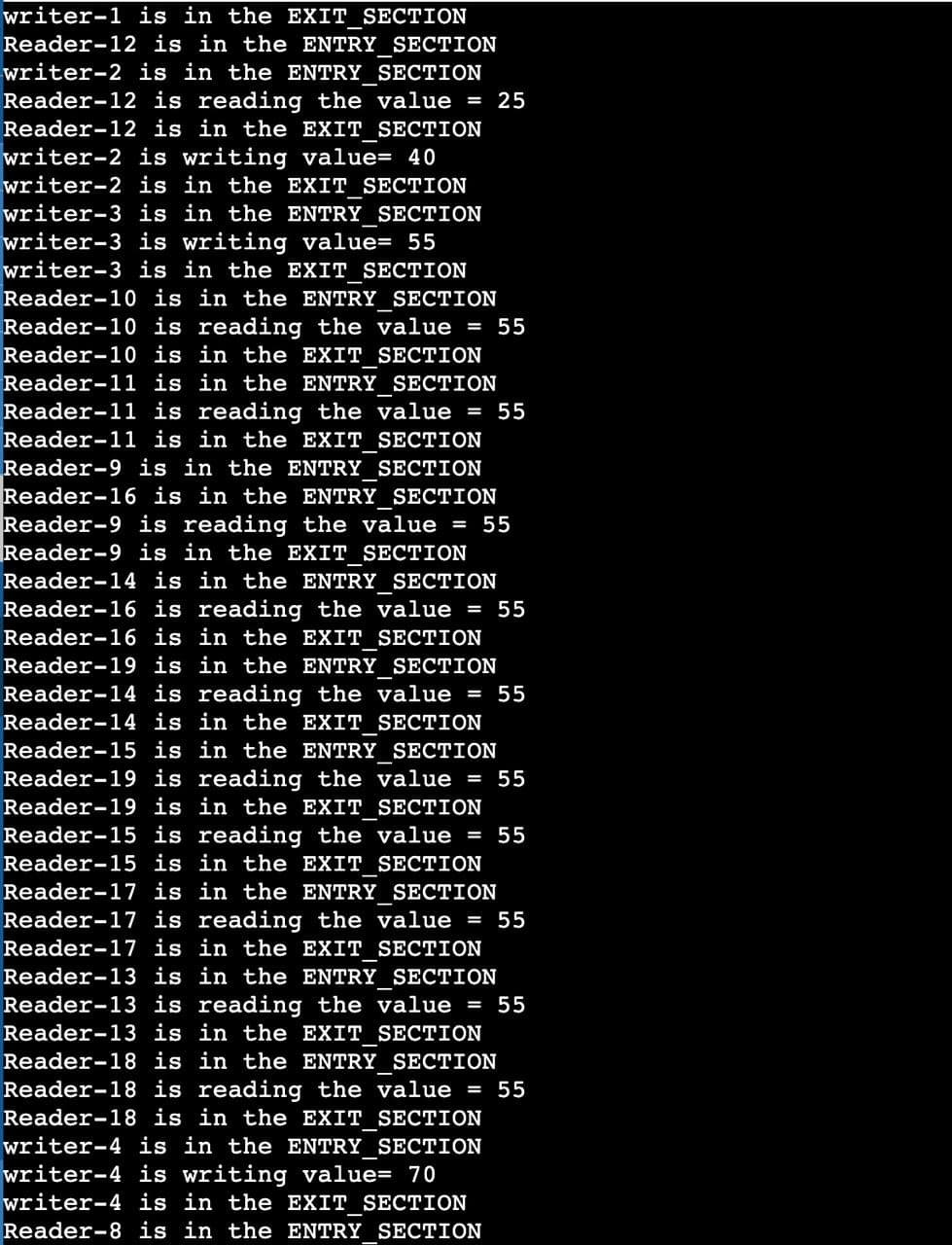
pthread\_join(r[19], NULL);

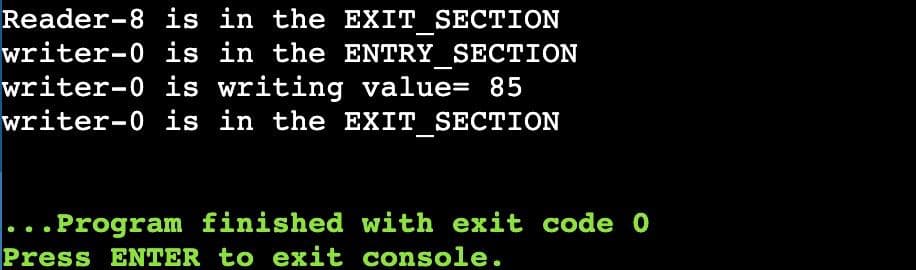
pthread\_join(w[4], NULL);

return 0;

}







Висновок: М'ютекс (англ. mutex, від mutual exclusion - "взаємний виняток") - це базовий механізм синхронізації. Він призначений для організації взаємовиключного доступу до загальних даних для декількох потоків з використанням бар'єрів пам'яті (для простоти можна вважати м'ютекс дверима, що веде до загальних даних).