МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования

**«Московский технический университет связи и информатики»**

Кафедра «Системное программирование»

Лабораторная работа №6

**«Системные средства синхронизации в ОС GNU/LINUX»**

по дисциплине

Операционные системы

Выполнил: студент гр. БЭИ2202

Кулешов А. С.

Вариант 16

Проверил: Королькова Т.В.

Москва, 2025 г

1. Изучите теоретическую часть лабораторной работы.

2. Исследуйте на конкретном примере особенности 3-х по выбору из

указанных методов синхронизации потоков:

1) семафоры

2) мьютексы

3) сигналы

4) условные переменные

5) барьеры

Задачи для синхронизации придумайте самостоятельно, исходя из особенностей методов.

Примечание

1. Задачи для каждого метода синхронизации должны быть различными.

2. Задачи должны наглядно демонстрировать выбранный метод

синхронизации и учитывать особенности его применения.

Студент, сдающий работу, должен АРГУМЕНТИРОВАННО

обосновать выбранную задачу и метод синхронизации.

**ВЫПОЛНЕНИЕ**

Выполню четыре задания. Выбранные средства синхронизации и пояснения содержаться в таблице. Привожу скриншоты запуска, а также код. Было выбрано четыре, а не три задания, чтобы если одно из них было неверным, то было бы лишнее.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Средство синхронизации | Описание СС | Задача | Пояснение |
| 1 | Мьютекс | Блокирует потоки, так чтобы лишь 1 поток мог пользоваться мьютексом одновременно | Подсчёт простых чисел в диапазоне [2;1000] | В данной задаче каждый поток должен увеличивать некий счётчик, мы используем мьютекс, чтобы избежать race-condition при увеличении этого счётчика |
| 2 | Барьеры | Блокирует потоки до тех пор пока все потоки не достигнут баррьера | Симуляции «Игра Жизнь» | Для обработки очередного состояния доски необходимо иметь предыдущее состояние. Наличие этого состояния гарантируется барьерами. |
| 3 | Семафор | Семафоры используются для синхронизации доступа к общей переменной, чтобы избежать состояния гонки при обновлении итоговой суммы. | Подсчёт суммы массива | Подсчёт происходит частично по каждому потоку, затем итоговая сумма вычисляется как сумма частичных сумм. Процесс финального сложения синхронизируется семафором. |
| 4 | Сигналы | Сигналы посылаются из процессов в процессы при помощи команды kill, а затем их можно отлавливать про помощи handler’ов в процессе, который должен получить сигнал | Демонстрация передвижения на перекрёстке | Каждая из четырёх частей дороги является отдельным процессом (могла быть потоком), Для координации между светофором и участками сегментов дорог используются сигналы. |

Код для задания №1.

#include <pthread.h>

#include <stdio.h>

#include <stdbool.h>

**int** threads\_num = **4**;

**int** range\_max = **1000**;

**int** primes = **0**;

**pthread\_mutex\_t** mutex;

**typedef** **struct** {

**int** start;

**int** end;

} Range;

**bool** **isPrime**(**int** num){

**if** (num < **2**) **return** false;

**for**(**int** i = **2**; i \* i <= num; ++i){

**if** (num % i == **0**) **return** false;

}

**return** true;

}

**void**\* **iterate**(**void**\* arg) {

Range\* range = (Range\*)arg;

**for** (**int** i = range->start; i < range->end; i++) {

pthread\_mutex\_lock(&mutex);

primes += isPrime(i);

pthread\_mutex\_unlock(&mutex);

}

**return** NULL;

}

**int** **main**() {

**pthread\_t** threads[threads\_num];

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

Range ranges[threads\_num];

**for** (**int** i = **0**; i < threads\_num; i++) {

ranges[i].start = i \* range\_max / threads\_num;

ranges[i].end = (i+**1**) \* range\_max / threads\_num;

pthread\_create(&threads[i], NULL, iterate, &ranges[i]);

}

**for** (**int** i = **0**; i < threads\_num; i++) {

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

}

printf("Primes: %d**\n**", primes);

pthread\_mutex\_destroy(&mutex);

**return** **0**;

}

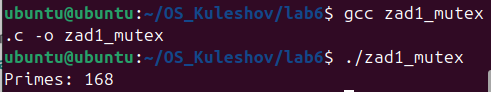


Рисунок 1 – Задание 1

Код для задания №2.

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <unistd.h>

#define SIZE 20

#define NUM\_THREADS 4

#define GENERATIONS 50

**int** current[SIZE][SIZE];

**int** next[SIZE][SIZE];

**pthread\_barrier\_t** barrier;

**void** **initialize\_board**() {

**for** (**int** i = **0**; i < SIZE; i++) {

**for** (**int** j = **0**; j < SIZE; j++) {

current[i][j] = rand() % **2**;

}

}

}

**void** **print\_board**() {

system("clear");

**for** (**int** i = **0**; i < SIZE; i++) {

**for** (**int** j = **0**; j < SIZE; j++) {

printf(current[i][j] ? "O " : ". ");

}

printf("**\n**");

}

usleep(**200000**);

}

**int** **count\_neighbors**(**int** x, **int** y) {

**int** count = **0**;

**for** (**int** dx = -**1**; dx <= **1**; dx++) {

**for** (**int** dy = -**1**; dy <= **1**; dy++) {

**if** (dx == **0** && dy == **0**) **continue**;

**int** nx = (x + dx + SIZE) % SIZE;

**int** ny = (y + dy + SIZE) % SIZE;

count += current[nx][ny];

}

}

**return** count;

}

**void**\* **game\_of\_life**(**void**\* arg) {

**int** thread\_part = \*(**int**\*)arg;

**int** start = thread\_part \* (SIZE / NUM\_THREADS);

**int** end = (thread\_part + **1**) \* (SIZE / NUM\_THREADS);

**for** (**int** gen = **0**; gen < GENERATIONS; gen++) {

**for** (**int** i = start; i < end; i++) {

**for** (**int** j = **0**; j < SIZE; j++) {

**int** neighbors = count\_neighbors(i, j);

**if** (current[i][j] == **1**) {

next[i][j] = (neighbors == **2** || neighbors == **3**) ? **1** : **0**;

} **else** {

next[i][j] = (neighbors == **3**) ? **1** : **0**;

}

}

}

pthread\_barrier\_wait(&barrier);

// only for one thread

**if** (thread\_part == **0**) {

**for** (**int** i = **0**; i < SIZE; i++) {

**for** (**int** j = **0**; j < SIZE; j++) {

current[i][j] = next[i][j];

}

}

print\_board();

}

pthread\_barrier\_wait(&barrier);

}

**return** NULL;

}

**int** **main**() {

**pthread\_t** threads[NUM\_THREADS];

**int** thread\_ids[NUM\_THREADS];

srand(time(NULL));

initialize\_board();

print\_board();

pthread\_barrier\_init(&barrier, NULL, NUM\_THREADS);

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

thread\_ids[i] = i;

pthread\_create(&threads[i], NULL, game\_of\_life, &thread\_ids[i]);

}

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

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

}

pthread\_barrier\_destroy(&barrier);

**return** **0**;

}

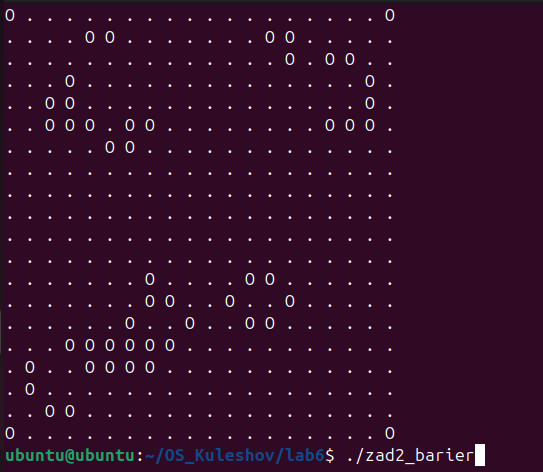


Рисунок 2 – Задание 2

Код для задания №3.

#include <pthread.h>

#include <semaphore.h>

#include <stdio.h>

#include <stdlib.h>

#define NUM\_THREADS 4

#define ARRAY\_SIZE 100

**int** array[ARRAY\_SIZE];

**int** partial\_sums[NUM\_THREADS];

**int** total\_sum = **0**;

**sem\_t** semaphore;

**void**\* **compute\_sum**(**void**\* arg) {

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

**int** start = thread\_id \* (ARRAY\_SIZE / NUM\_THREADS);

**int** end = (thread\_id == NUM\_THREADS - **1**) ? ARRAY\_SIZE : start + (ARRAY\_SIZE / NUM\_THREADS);

**int** sum = **0**;

**for** (**int** i = start; i < end; i++) {

sum += array[i];

}

partial\_sums[thread\_id] = sum;

printf("Thread %d: inter sum = %d**\n**", thread\_id, sum);

sem\_wait(&semaphore);

total\_sum += sum;

sem\_post(&semaphore);

**return** NULL;

}

**int** **main**() {

**pthread\_t** threads[NUM\_THREADS];

**int** thread\_ids[NUM\_THREADS];

srand(time(NULL));

**for** (**int** i = **0**; i < ARRAY\_SIZE; i++) {

array[i] = rand() % **100**;

}

sem\_init(&semaphore, **0**, **1**);

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

thread\_ids[i] = i;

pthread\_create(&threads[i], NULL, compute\_sum, &thread\_ids[i]);

}

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

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

}

sem\_destroy(&semaphore);

printf("Total: %d**\n**", total\_sum);

**return** **0**;

}

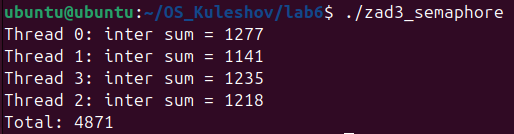


Рисунок 3 – Задание 3

Код для задания №4.

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>

#include <sys/wait.h>

#include <time.h>

**const** **int** NUM\_DIRECTIONS = **4**;

**int** traffic\_light;

**void** **change\_traffic\_light**(**int** signalo) {

traffic\_light = (traffic\_light + **1**) % NUM\_DIRECTIONS;

}

**void** **road\_process**(**int** road\_id) {

srand(time(NULL) ^ getpid());

**while** (**1**) {

sleep(rand() % **5** + **1**);

**int** target\_road;

**do** {

target\_road = rand() % NUM\_DIRECTIONS;

} **while** (target\_road == road\_id);

printf("Auto: WANT %d -> %d **\n**", road\_id, target\_road);

**while** (traffic\_light != road\_id) {

usleep(**100000**);

}

printf("Auto: GO %d -> %d **\n**", road\_id, target\_road);

}

}

**int** **main**() {

traffic\_light = **0**;

signal(SIGALRM, change\_traffic\_light);

**int** childs[NUM\_DIRECTIONS];

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

**pid\_t** pid = fork();

**if** (pid == **0**) {

road\_process(i);

} **else**{

childs[i] = pid;

}

}

**while**(**1**){

kill(childs[**0**], SIGALRM);

kill(childs[**1**], SIGALRM);

kill(childs[**2**], SIGALRM);

kill(childs[**3**], SIGALRM);

change\_traffic\_light(**0**);

printf("Lights: StateChange to %d**\n**", traffic\_light);

sleep(**5**);

}

**return** **0**;

}

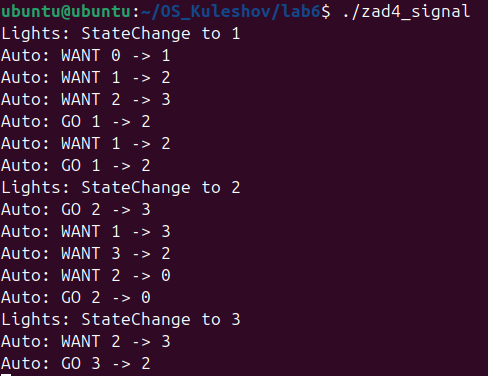


Рисунок 4 – Задание 4