Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и системное программирование

ОТЧЁТ

к лабораторной работе №4

на тему

**ВЗАИМОДЕЙСТВИЕ ПРОЦЕССОВ (ПОТОКОВ): ВЗАИМНОЕ**

**ИСКЛЮЧЕНИЕ И СИНХРОНИЗАЦИЯ**

Студент: Сенько Н.С.

Преподаватель: Гриценко Н.Ю.

Минск 2024

**СОДЕРЖАНИЕ**

[1 Формулировка задачи 3](#_Toc177507035)

[2 Описание функций программы 4](#_Toc177507036)

[2.1 Создание и запуск процессов «обедающих философов» 4](#_Toc177507037)

[2.2 Алгоритм действий «обедающих философов» 4](#_Toc177507038)

[Заключение 5](#_Toc177507040)

[Список использованных источников 6](#_Toc177507041)

[Приложение А (обязательное) Исходный код программы 7](#_Toc177507042)

1 ФОРМУЛИРОВКА ЗАДАЧИ

Реализация модели взаимодействия процессов (потоков) «писатели- читатели» с возможностью параметризации и наглядного (не обязательно графического) представления результатов.

Поскольку речь идет о модели, реальные данные не обязательны, можно ограничиться моделированием обращений к ним (обращение характеризуется параметрами запроса, моментом обращения, длительностью исполнения). Но можно реализовать и макет системы, использующей реальные (тестовые, «учебные») данные в соответствии с этой моделью. Таким образом, вместо моделирования обращений с заданными характеристиками могут быть сами обращения к «настоящим» данным (прикладные функции системы можно позаимствовать из предыдущих работ).

Обеспечение корректного функционирования, т.е. избегание как «грязного» считывания данных и одновременно минимизация блокировок.

Изменяемые параметры модели: количество «писателей» и «читателей»; интенсивность их обращений к ресурсам, длительность использования ресурса, размер блока данных и т.п. (характеристики случайных величин при моделировании); штрафы за простой участников; величина тайм-аутов и др.

Результаты моделирования: соотношение времени активности/блокировки участников; соотношение успешных/неуспешных обращений; общая эффективность (пропускная способность) по «записи» и «чтению» и др.

2 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Согласно формулировке задачи, были спроектированы следующие функции программы:

– функция создания и запуска потоков «писателей»;

– функция создания и запуска потоков «читателей».

## **2.1** **Создание и запуск процессов «обедающих философов»**

При запуске программы создаются потоки «писателей», которые записывают в переменную новые значения и потоки «читателей», которые пытаются прочитать это значение [1, 2].

## **2.2** **Алгоритм действий «писатель-читатель»**

Отдельная переменная в коде отвечает за алгоритм действий «писатель-читатель». Например, базовый метод в котором читатели пытаются прочитать переменную и ждет освобождения, когда она не доступна [3, 4]. (рисунок 2.1).

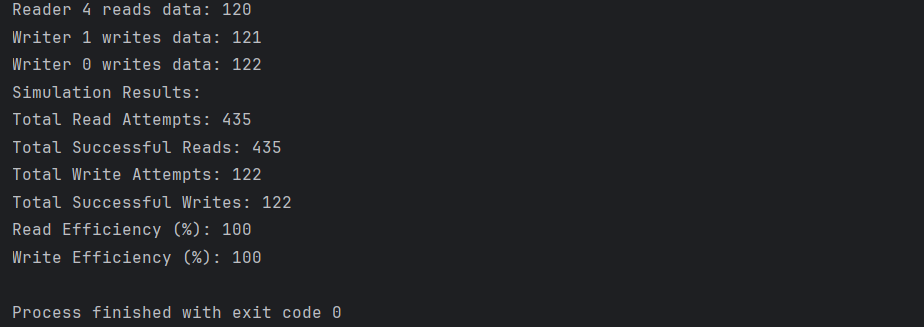


Рисунок 2.1 – Результат записи и чтения данных для каждого потока

ЗАКЛЮЧЕНИЕ

В данной лабораторной работе была реализована модель взаимодействия процессов «писатели-читатели» с возможностью параметризации и наглядного представления результатов. Целью работы было обеспечить корректное функционирование модели с минимизацией блокировок и избежанием «грязного» считывания данных.

Сначала был разработан алгоритм для синхронизации потоков «писателей» и «читателей», учитывающий изменяемые параметры, такие как количество участников, интенсивность их обращений, длительность использования ресурсов и размер блоков данных. Затем было обеспечено отсутствие условий гонки и «грязного» чтения данных путем применения механизмов блокировок и временных интервалов. Для достижения оптимального баланса между блокировкой и параллельностью были подобраны параметры моделирования, например, тайм-ауты и штрафы за простой. Была предоставлена возможность изменять параметры модели для исследования их влияния на поведение системы, начиная от количества «писателей» и «читателей» и заканчивая характеристиками случайных величин.

При реализации и тестировании системы был создан макет, который может работать как с моделируемыми, так и с тестовыми данными для наглядности и проверки работоспособности. В результате моделирования было проведено множество анализов: оценено соотношение времени активности и времени блокировок процессов, проанализировано количество успешных и неуспешных обращений к данным, а также определена общая эффективность модели в терминах пропускной способности операций записи и чтения.

Результаты показали, что модель обладает высокой гибкостью и адаптивностью при изменении параметров. Эффективность процессов в значительной степени зависела от оптимального подбора тайм-аутов и минимизации блокировок. Наглядное представление результатов позволило выявить зависимости и провести анализ эффективности различных конфигураций модели. В заключение, проделанная работа доказала жизнеспособность разработанной модели и предоставила инструментарий для дальнейшего углублённого изучения темы взаимодействия процессов в системах с конкурентным доступом к данным.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] CreateThread function [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createthread

[2] CreateSemaphore function [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-createsemaphorea

[3] ReleaseSemaphore function [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-releasesemaphore

[4] WaitForSingleObject function [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-waitforsingleobject

ПРИЛОЖЕНИЕ А  
(обязательное)  
Исходный код программы

#include <windows.h>

#include <iostream>

#include <vector>

// Параметры

const int num\_readers = 5;

const int num\_writers = 2;

const int read\_duration = 100; // мс

const int write\_duration = 150; // мс

const int simulation\_time = 10; // секунды

// Глобальные переменные для управления доступом к ресурсу

SRWLOCK rw\_lock = SRWLOCK\_INIT;

int shared\_data = 0;

// Метрики для анализа

LONG read\_attempts = 0;

LONG successful\_reads = 0;

LONG write\_attempts = 0;

LONG successful\_writes = 0;

// События для остановки потоков

HANDLE stop\_event;

// Объект для синхронизации вывода в консоль

HANDLE io\_mutex;

// Функция для потока-читателя

DWORD WINAPI reader(LPVOID lpParam) {

int id = (int)(intptr\_t)lpParam;

while (WaitForSingleObject(stop\_event, 0) != WAIT\_OBJECT\_0) {

InterlockedIncrement(&read\_attempts);

AcquireSRWLockShared(&rw\_lock);

InterlockedIncrement(&successful\_reads);

WaitForSingleObject(io\_mutex, INFINITE);

std::cout << "Reader " << id << " reads data: " << shared\_data << std::endl;

ReleaseMutex(io\_mutex);

ReleaseSRWLockShared(&rw\_lock);

Sleep(read\_duration);

}

return 0;

}

// Функция для потока-писателя

DWORD WINAPI writer(LPVOID lpParam) {

int id = (int)(intptr\_t)lpParam;

while (WaitForSingleObject(stop\_event, 0) != WAIT\_OBJECT\_0) {

InterlockedIncrement(&write\_attempts);

AcquireSRWLockExclusive(&rw\_lock);

InterlockedIncrement(&successful\_writes);

++shared\_data;

WaitForSingleObject(io\_mutex, INFINITE);

std::cout << "Writer " << id << " writes data: " << shared\_data << std::endl;

ReleaseMutex(io\_mutex);

ReleaseSRWLockExclusive(&rw\_lock);

Sleep(write\_duration);

}

return 0;

}

int main() {

// Инициализация мьютекса и события

io\_mutex = CreateMutex(NULL, FALSE, NULL);

stop\_event = CreateEvent(NULL, TRUE, FALSE, NULL);

// Создаем потоки-читатели

std::vector<HANDLE> reader\_threads(num\_readers);

for (int i = 0; i < num\_readers; ++i) {

reader\_threads[i] = CreateThread(NULL, 0, reader, (LPVOID)(intptr\_t)i, 0, NULL);

}

// Создаем потоки-писатели

std::vector<HANDLE> writer\_threads(num\_writers);

for (int i = 0; i < num\_writers; ++i) {

writer\_threads[i] = CreateThread(NULL, 0, writer, (LPVOID)(intptr\_t)i, 0, NULL);

}

// Запускаем симуляцию

Sleep(simulation\_time \* 1000); // Симуляция на 10 секунд

// Сигнализируем о завершении

SetEvent(stop\_event);

// Дожидаемся завершения всех потоков

WaitForMultipleObjects(num\_readers, reader\_threads.data(), TRUE, INFINITE);

WaitForMultipleObjects(num\_writers, writer\_threads.data(), TRUE, INFINITE);

// Закрываем дескрипторы

for (HANDLE thread : reader\_threads) {

CloseHandle(thread);

}

for (HANDLE thread : writer\_threads) {

CloseHandle(thread);

}

CloseHandle(io\_mutex);

CloseHandle(stop\_event);

// Вывод результатов моделирования

std::cout << "Simulation Results:\n";

std::cout << "Total Read Attempts: " << read\_attempts << "\n";

std::cout << "Total Successful Reads: " << successful\_reads << "\n";

std::cout << "Total Write Attempts: " << write\_attempts << "\n";

std::cout << "Total Successful Writes: " << successful\_writes << "\n";

std::cout << "Read Efficiency (%): " << (successful\_reads \* 100.0 / read\_attempts) << "\n";

std::cout << "Write Efficiency (%): " << (successful\_writes \* 100.0 / write\_attempts) << "\n";

return 0;

}