Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ
Факультет компьютерных систем и сетей Кафедра электронных вычислительных машин
Дисциплина: Операционные системы и системное программирование

ОТЧЕТ
по лабораторной работе №5 на тему
“Потоки исполнения, взаимодействие и синхронизация”

Выполнил:
студ. гр. 350501 Везенков М. Ю.

Проверил: ст.пр. Поденок Л. П.

Минск 2025

СОДЕРЖАНИЕ

1 ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ

Здесь две задачи. Обе «производители-потребители» для потоков.

Изучаемые системные вызовы: `pthread_create`(), `pthread_exit`(), `pthread_join`(), `pthread_yield`() , `pthread_cancel`(), `pthread_cond_init`(), `pthread_cond_destroy`(), `pthread_cond_*wait`(), `pthread_cond_signal`().

Две лабораторных в одной:

5.1) Аналогична лабораторной `No` 4, но только с потоками, `posix`-семафорами и мьютексомв рамках одного процесса.

Дополнительно обрабатывается еще две клавиши – увеличение и уменьшение размера очереди. Следует предусмотреть обработку запроса на уменьшение очереди таким образом, чтобы при появлении пустого места уменьшался размер очереди, а не очередной производитель размещал там свое сообщение.

5.2 Аналогична лабораторной № 5.1, но с использованием условных переменных (см. лекции СПОВМ/ОСиСП).

Требования к сборке аналогичны требованиям из лабораторной № 2.

2 ВЫПОЛНЕНИЕ ЛАБОРАТОРНОЙ РАБОТЫ

2.1 Описание программы

Данная программа представляет собой многопоточное приложение, реализующее классическую задачу "Производитель-Потребитель" с использованием потоков POSIX (pthread) и разделяемой структуры данных (очереди) для обмена сообщениями. Программа демонстрирует две основные стратегии синхронизации доступа к разделяемым ресурсам между потоками: с использованием POSIX семафоров и с использованием мьютекса и переменных состояния POSIX. Выбор механизма синхронизации осуществляется при запуске программы через аргументы командной строки.

Основной поток программы отвечает за инициализацию системы, создание и управление потоками производителей и потребителей в ответ на команды пользователя, а также за корректное завершение работы и освобождение ресурсов. Потоки производителей генерируют сообщения и добавляют их в общую очередь, а потоки потребителей извлекают сообщения из очереди и обрабатывают их.

Центральным элементом программы является структура данных queue_t, представляющая собой кольцевую очередь переменной емкости, расположенную в памяти процесса, к которой имеют доступ все потоки. Эта структура включает буфер для хранения сообщений, индексы головы и хвоста, текущее количество элементов и максимальную емкость, а также примитивы синхронизации.

В зависимости от выбранного режима синхронизации, queue_t содержит либо два POSIX семафора (empty_slots для свободных мест и full_slots для занятых мест) и мьютекс (mutex), либо мьютекс (mutex) и две переменные состояния POSIX (not_empty и not_full). Мьютекс используется для обеспечения исключительного доступа к самой структуре очереди (индексам, счетчикам), а семафоры или переменные состояния используются для блокирования/разблокирования потоков в зависимости от состояния очереди (полная или пустая).

Программа поддерживает динамическое изменение емкости очереди во время выполнения. Для обеспечения корректного завершения работы в многопоточной среде используется глобальный атомарный флаг g_terminate_flag, который устанавливается основным потоком при получении сигналов завершения или команды выхода, позволяя рабочим потокам gracefully завершить свои циклы. Утилитарные функции для настройки терминала в неблокирующий режим и обработки ввода с клавиатуры также являются частью системы для реализации интерактивного управления.

2.2 Описание основных функций

Основная логика программы распределена между основным потоком (функция `main` в `main.c`) и функциями точек входа для рабочих потоков (`producer_thread_func` в `producer.c` и `consumer_thread_func` в `consumer.c`), а также функциями управления очередью и синхронизацией в `queue_manager.c`.

Функция `main` парсит аргументы командной строки для определения механизма синхронизации (`-m sem` или `-m cond`). Она инициализирует генератор случайных чисел и настраивает стандартный ввод терминала в неблокирующий режим без эха с использованием функций из `utils.c`, что позволяет считывать команды посимвольно. Далее вызывается `queue_create` из `queue_manager.c` для динамического выделения и инициализации структуры очереди и соответствующих примитивов синхронизации (семафоров или переменных состояния и мьютекса) в зависимости от выбранного режима.

Регистрируются обработчики сигналов `SIGINT` и `SIGTERM` для основного потока (`main_signal_handler`), которые устанавливают глобальный флаг `g_terminate_flag`, сигнализируя рабочим потокам о необходимости завершения. Также регистрируется функция `cleanup_threads` через `atexit` для гарантированного выполнения процедур очистки при выходе из программы.

Основной цикл `main` ожидает ввода команд с клавиатуры в неблокирующем режиме. Команды '`p`' и '`c`' создают новые потоки производителей и потребителей соответственно с помощью `pthread_create`, передавая им указатель на общую очередь и уникальный `ID` через динамически выделяемую структуру `thread_args_t`. Команды '`P`' и '`C`' логически уменьшают счетчики активных производителей/потребителей, влияя на возможность создания новых потоков и отображение статуса, но не останавливая уже запущенные потоки.

Команды '+' и '-' вызывают `queue_resize` для динамического изменения емкости очереди. Команда '`s`' выводит текущий статус системы, включая емкость и заполненность очереди, а также количество потоков, используя безопасные функции доступа к данным очереди (`queue_get_*`). Команда '`q`' устанавливает флаг `g_terminate_flag`, инициируя завершение. В цикле предусмотрена небольшая пауза с помощью `nanosleep`.

Функция `cleanup_threads`, вызываемая при завершении, устанавливает `g_terminate_flag`, пытается разбудить все потенциально заблокированные потоки (путем постинга на семафоры или широковещательного оповещения переменных состояния), ожидает завершения всех созданных потоков с помощью `pthread_join`, уничтожает примитивы синхронизации и освобождает память очереди с помощью `queue_destroy`, и восстанавливает исходные настройки терминала.

Функции producer_thread_func и consumer_thread_func в своих циклах постоянно пытаются добавить/извлечь сообщения из очереди, используя queue_add и queue_remove соответственно, которые внутри реализуют логику блокирования/разблокирования на семафорах или переменных состояния и работают с мьютексом для доступа к данным очереди. Они проверяют g_terminate_flag для выхода из цикла и завершения потока, а также обрабатывают ошибки и прерывания системных вызовов (например, EINTR) для корректной реакции на сигналы.

2.3 Описание порядка сборки и использования

Сборка проекта осуществляется с использованием утилиты make и предоставленного файла Makefile. Makefile поддерживает два режима сборки: отладочный (debug) и релизный (release). По умолчанию используется отладочный режим (make или make debug-build), который включает флаги компиляции -g3 и -ggdb для отладочной информации.

Релизный режим (make MODE=release или make release-build) включает оптимизацию (-O2) и обрабатывает предупреждения как ошибки (-Werror). Компиляция исходных файлов (.c) производится с базовыми флагами (-std=c11 -pedantic -W -Wall -Wextra -Wmissing-prototypes -Wstrict-prototypes -D_POSIX_C_SOURCE=200809L). При линковке обязательно используется флаг -pthread для поддержки потоков POSIX. Объектные (.o) и исполняемые файлы помещаются в подкаталоги build/debug/ или build/release/ в зависимости от выбранного режима сборки.

Для сборки проекта в отладочном режиме выполните команду make или make debug-build. Для сборки в релизном режиме выполните make release-build. Команда make clean удаляет все скомпилированные файлы и каталоги сборки.

Для запуска собранной программы используются специальные цели в Makefile. make run выполняет сборку в отладочном режиме и запускает исполняемый файл prod_cons_threads из build/debug/, используя семафоры по умолчанию. make run-sem явно указывает использовать семафоры (-m sem), а make run-cond явно указывает использовать переменные состояния (-m cond) при запуске отладочной версии.

Аналогичные цели run-release, run-release-sem и run-release-cond предусмотрены для запуска релизной версии. Например, для запуска отладочной версии с использованием переменных состояния выполните make run-cond. Программа ожидает команд пользователя ('p', 'c', 'P', 'C', '+', '-', 's', 'q') на стандартном вводе. Завершение программы происходит по команде 'q' или при получении сигналов SIGINT (Ctrl+C) или SIGTERM.

3 РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ

```
--- Producer/Consumer Control (Mode: sem) ---
 p: Add Producer   c: Add Consumer
 P: Remove Last Producer (stops adding)   C: Remove Last Consumer (stops
adding)
 +: Increase Queue Capacity   -: Decrease Queue Capacity
 s: Show Status    q: Quit
------------------------------------------------
Enter command:
[Main] Producer thread created.
Enter command: [Producer 1] Started.
[Producer 1] Added msg (Type:61 Size:34 Hash:21060). Total Added: 1
[Main] Consumer thread created.
Enter command: [Consumer 1] Started.
[Consumer 1] Extracted msg (Type:61 Size:34 Hash:21060 -> OK). Total
Extracted: 1
--- System Status ---
Mode:            Semaphores
Queue Capacity: 10
Queue Occupied: 2
Queue Free:     8
Total Added:    3
Total Extracted:1
Producers:      1 / 10 (Created: 1)
Consumers:      1 / 10 (Created: 1)
--------------------
Enter command:
[Main] Quit command received...
[Main] Exiting.
[Cleanup] Starting cleanup...
[Cleanup] Signaling sync primitives to unblock threads...
[Cleanup] Joining producer threads...
[Consumer 1] Terminating.
[Producer 1] Terminating.
[Cleanup] Joined producer thread 1.
[Cleanup] Joining consumer threads...
[Cleanup] Joined consumer thread 1.
[Queue Destroy] Destroying queue resources...
[Queue Destroy] Queue resources destroyed.
[Cleanup] Cleanup complete.
```

Рисунок 3.1 – Пример работы с использованием семафоров

```
[Main] Using Condition Variables.
[Main] Initializing system...
[Queue Create] Queue initialized successfully (CondVar Mode).
--- Producer/Consumer Control (Mode: cond) ---
 p: Add Producer   c: Add Consumer
 P: Remove Last Producer (stops adding)   C: Remove Last Consumer (stops
adding)
 +: Increase Queue Capacity   -: Decrease Queue Capacity
 s: Show Status    q: Quit
------------------------------------------------
Enter command:
[Main] Producer thread created.
Enter command: [Producer 1] Started.
[Producer 1] Added msg (Type:91 Size:246 Hash:15232). Total Added: 1
[Main] Consumer thread created.
Enter command: [Consumer 1] Started.
[Consumer 1] Extracted msg (Type:91 Size:246 Hash:15232 -> OK). Total
Extracted: 1
--- System Status ---
Mode:            CondVars
Queue Capacity: 10
Queue Occupied: 0
Queue Free:     10
Total Added:    1
Total Extracted:1
Producers:      1 / 10 (Created: 1)
Consumers:      1 / 10 (Created: 1)
---------------------
Enter command: [Consumer 1] Queue empty, waiting...
[Main] Quit command received...
[Main] Exiting.
[Cleanup] Starting cleanup...
[Cleanup] Signaling sync primitives to unblock threads...
[Cleanup] Joining producer threads...
[Consumer 1] Terminating.
[Producer 1] Terminating.
[Cleanup] Joined producer thread 1.
[Cleanup] Joining consumer threads...
[Cleanup] Joined consumer thread 1.
[Queue Destroy] Destroying queue resources...
[Queue Destroy] Queue resources destroyed.
[Cleanup] Cleanup complete.
```

Рисунок 3.2 – Пример работы с использованием условных переменных

4 ВЫВОД

В рамках данной лабораторной работы были успешно изучены и практически применены механизмы многопоточного программирования с использованием библиотеки `POSIX Threads`. Была реализована задача "Производитель-Потребитель", демонстрирующая взаимодействие потоков через разделяемую структуру данных - кольцевую очередь. Особое внимание было уделено реализации и сравнению двух ключевых подходов к синхронизации потоков: с использованием `POSIX` семафоров и с использованием мьютекса в комбинации с переменными состояния `POSIX`.

Программа позволила наглядно продемонстрировать, как эти примитивы используются для управления доступом к разделяемой очереди и координации работы производителей (ожидание свободных слотов) и потребителей (ожидание заполненных слотов). Дополнительно были освоены техники динамического управления ресурсами (изменение размера очереди во время выполнения), безопасной обработки сигналов в многопоточной среде для корректного завершения всех потоков и освобождения ресурсов, а также настройки терминала для интерактивного ввода команд без буферизации.

Таким образом, лабораторная работа предоставила ценный опыт в проектировании и реализации параллельных приложений с использованием стандартных средств `POSIX`.