Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное учреждение высшего образования

«Ульяновский государственный технический университет»

Дисциплина «Операционные системы».

**Лабораторная работа №4. Проектирование пула потоков**

**Выполнил**

Студент группы ИВТИИбд-21

Пешков М. А.

**Проверил(а):**

Преподаватель Беляев К. С.

Ульяновск 2025

**Тема:** Реализация потокового пула (Thread Pool) с использованием pthread и семафоров  
**Цель работы:**

Изучить и реализовать механизм потокового пула на языке C с использованием POSIX потоков (pthread), синхронизации с помощью мьютексов и семафоров. Научиться эффективно управлять группой рабочих потоков для выполнения множества задач, минимизируя накладные расходы на создание и уничтожение потоков.

**Используемый язык:** C (GCC, Unix/Linux среда)

## **Задание:**

1. Разработать структуру данных для хранения задач в очереди.
2. Создать пул из фиксированного числа потоков, которые будут обрабатывать задачи из очереди.
3. Реализовать функцию отправки задач в очередь с безопасным доступом из нескольких потоков.
4. Обеспечить корректное завершение всех потоков и освобождение ресурсов.
5. Проверить работу пула на примере простой задачи.

### **Теоретические сведения**

Пул потоков — это программный механизм, позволяющий эффективно управлять ограниченным числом потоков для выполнения множества задач. Вместо того чтобы создавать новый поток на каждую задачу, создается фиксированное количество рабочих потоков, которые получают задачи из общей очереди и выполняют их по мере поступления.

Основные преимущества пула потоков:

* **Снижение накладных расходов**: Создание и уничтожение потоков — затратные операции. Пул потоков позволяет переиспользовать уже созданные потоки, что экономит ресурсы.
* **Управление параллелизмом**: Фиксированное число потоков ограничивает количество одновременно выполняемых задач, предотвращая чрезмерное переключение контекста и перегрузку системы.
* **Упрощение синхронизации**: Очередь задач служит общей точкой взаимодействия между потоками, а средства синхронизации (мьютексы, семафоры) обеспечивают корректный и безопасный доступ к разделяемым данным.

Для реализации пула потоков используются следующие базовые элементы:

* **Очередь задач** — буфер фиксированного размера, куда помещаются функции и их аргументы для выполнения.
* **Рабочие потоки** — набор потоков, которые циклически получают задачи из очереди и выполняют их.
* **Мьютекс** — для синхронизации доступа к очереди задач, чтобы избежать состояния гонки.
* **Семафор** — для оповещения рабочих потоков о появлении новых задач.

При завершении работы пула необходимо корректно остановить все рабочие потоки, чтобы избежать утечек ресурсов и повреждений данных.

### **Структура проекта:**

#### main.c

Основной код, реализующий:

* трансляцию адресов;
* TLB (FIFO);
* таблицу страниц;
* подкачку из файла.

#### BACKING\_STORE.bin

Файл с 256 страницами по 256 байт (всего 65 536 байт).

#### addresses.txt

Файл с логическими адресами.

**Ход работы:**

**1. Структура задачи и очередь**

* Для хранения задачи использовалась структура Task, содержащая указатель на функцию и аргумент:

typedef struct {

void (\*function)(void \*);

void \*arg;

} Task;

* Очередь задач реализована с кольцевым буфером и содержит массив задач, индексы front, rear, и счётчик count для отслеживания количества задач. Для синхронизации доступа используется мьютекс pthread\_mutex\_t и семафор sem\_t для ожидания наличия задач.

**2. Инициализация пула потоков**

* Функция pool\_init() инициализирует очередь, мьютекс, семафор и создаёт MAX\_THREADS рабочих потоков через pthread\_create.
* Каждый поток запускает функцию worker, которая в бесконечном цикле ждёт сигнал от семафора и обрабатывает задачи.

**3. Отправка задачи в пул**

* pool\_submit() — функция, добавляющая новую задачу в очередь.
* В критической секции (защищённой мьютексом) добавляется задача, увеличивается счётчик, обновляются индексы.
* После добавления задачи вызывается sem\_post, чтобы разблокировать один из ожидающих потоков.

**4. Обработка задач**

* Рабочие потоки ждут семафор sem\_wait(&queue.has\_tasks), который блокирует поток, пока в очереди нет задач.
* После получения задачи поток выполняет функцию с аргументом.
* Поток продолжает работать, пока не установлен флаг завершения shutdown.

**5. Завершение работы пула**

* В pool\_shutdown() устанавливается флаг завершения, после чего происходит «пробуждение» всех потоков через sem\_post.
* Каждый поток, проснувшись, проверяет флаг и выходит из цикла, завершая работу.
* Основной поток ждёт завершения всех рабочих с помощью pthread\_join.
* Освобождаются ресурсы: уничтожаются мьютекс и семафор.

### **Вывод**

Реализация потокового пула показала, что данный подход является эффективным и удобным для управления многопоточностью в современных приложениях. Пул потоков позволяет значительно снизить накладные расходы, связанные с постоянным созданием и уничтожением потоков, так как рабочие потоки создаются один раз и переиспользуются для обработки множества задач.

Важным аспектом работы с потоками является правильная организация синхронизации доступа к разделяемым ресурсам, таким как очередь задач. Использование мьютексов гарантирует исключительный доступ и предотвращает состояние гонки, а семафоры позволяют потокам эффективно ожидать появления новых задач без затрат ресурсов на активное ожидание.

Также стоит отметить, что корректное завершение пула потоков — не менее важная часть реализации. Установка флага завершения и уведомление всех потоков о необходимости выхода позволяют избежать зависаний и утечек ресурсов.

Таким образом, опыт работы с потоковым пулом развивает навыки параллельного программирования и управления синхронизацией, что критично для разработки производительных и надежных многопоточных систем. Полученные знания могут быть легко применены при реализации серверных приложений, систем обработки данных, и других проектов, требующих эффективного распределения задач между потоками.