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

Лабораторная работа №5
на тему
«Потоки исполнения, взаимодействие и синхронизация»

Выполнил: студент группы 350501
Гожуленко М.Ю.

Проверил: старший преподаватель каф. ЭВМ
Поденок Л. П.

Минск 2025

1 УСЛОВИЕ ЛАБОРАТОРНОЙ РАБОТЫ

Данная лабораторная работа содержит в себе две лабораторных:

1) Аналогична лабораторной № 4, но только с потоками, posix-семафорами и мьютексом в 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;

2) Аналогична лабораторной № 1, но с использованием условных переменных (см. лекции СПОВМ/ОСиСП).

Требования к сборке аналогичны требованиям из лабораторной № 2.

2 ОПИСАНИЕ АЛГОРИТМОВ И РЕШЕНИЙ

2.1 Общая структура программы

Программа реализует модель «производитель–потребитель» с использованием многопоточности и POSIX-примитивов синхронизации. Основной поток управляет созданием/остановкой рабочих потоков, обрабатывает пользовательский ввод и обеспечивает динамическое изменение размера очереди. Ключевые особенности:

1) Использование потоков (`pthread`) вместо процессов;
2) Динамическое изменение размера очереди с синхронизацией;
3) Механизм отложенного уменьшения размера очереди;
4) Защита от конфликтов при параллельном доступе.

2.2 Алгоритм работы основного процесса

Инициализация:
1) Выделение памяти для структуры очереди (`init_queue`);
2) Инициализация семафоров:
`SEM_MUTEX`: контроль доступа к очереди;
`SEM_EMPTY`: счётчик свободных слотов;
`SEM_FULL`: счётчик занятых слотов;
`SEM_RESIZE`: синхронизация операций изменения размера.
3) Регистрация обработчиков сигналов `SIGINT`/`SIGTERM`;
4) Создание массивов для хранения идентификаторов потоков.
Управление потоками:
1) Команды с клавиатуры:
p: создание потока-производителя (`pthread_create`);
c: создание потока-потребителя;
P/C: остановка последнего созданного потока;
s: вывод статуса очереди;
q: корректное завершение.
2) Ограничение `MAX_PRODUCERS`/`MAX_CONSUMERS`.
Завершение работы:
1) Установка флага `should_terminate = true`;
2) Принудительная отмена рабочих потоков (`pthread_cancel`);
3) Очистка ресурсов: уничтожение семафоров, освобождение памяти.

2.3 Алгоритм работы производителя

Цикл выполнения:
1) Генерация сообщения (`create_message`);

2) Проверка отложенных операций изменения размера (`check_and_perform_resize`);

3) Ожидание свободного слота (`sem_wait(SEM_EMPTY)`);

4) Захват мьютекса очереди (`sem_wait(SEM_MUTEX)`);

5) Проверка доступности слотов с учётом резерва для уменьшения;

6) Добавление сообщения в хвост очереди;

7) Обновление счетчиков (`tail`, `free`, `added`);

8) Освобождение мьютекса и инкремент `SEM_FULL`;

9) Задержка 100-600 мс.

Особенности:

- Резервирование слотов при отложенном уменьшении размера;
- Автоматическая проверка возможности выполнения отложенного resize.

2.4 Алгоритм работы потребителя

Цикл выполнения:

1) Ожидание заполненного слота (`sem_wait(SEM_FULL)`);

2) Захват мьютекса очереди;

3) Извлечение сообщения из головы очереди;

4) Обновление счетчиков (`head`, `free`, `extracted`);

5) Проверка необходимости выполнения отложенного уменьшения;

6) Освобождение мьютекса и инкремент `SEM_EMPTY`;

7) Проверка целостности сообщения (`calculate_hash`);

8) Задержка 200-700 мс.

Особенности:

- Инициация проверки отложенного уменьшения размера;
- Динамическая адаптация к изменениям размера очереди.

2.5 инамическое изменение размера очереди

Механизм изменения размера:

1) Захват `sem_resize` и `resize_mutex`;

2)Проверка возможности немедленного выполнения:

- Увеличение: всегда разрешено (до `MAX_QUEUE_SIZE`);
- Уменьшение: только если занято ≤ нового размера.

3) При невозможности уменьшения:

- Установка флага `resize_decrease_pending`;
- Резервирование слотов (`reserved_slots`).

4) Выделение нового буфера и копирование данных;

5) Корректировка семафоров `SEM_EMPTY`/`SEM_FULL`;

6) Обработка вложенных запросов на изменение размера.

Ключевые особенности:

- Атомарная операция копирования данных;
- Сохранение порядка сообщений;
- Перерасчёт значений семафоров.
- Механизм отложенного выполнения для уменьшения.

2.6 Дополнительные функции

Синхронизация:
- Мьютекс `resize_mutex`: защита флагов изменения размера;
- Семафор `sem_resize`: предотвращение параллельных `resize`.

Контроль целостности:
- `calculate_hash`: проверка CRC-подобной хеш-суммы;
- Выравнивание данных сообщения до 4 байт.

Управление ресурсами:
- `cleanup`: корректное освобождение памяти и деструкция семафоров;
- `signal_handler`: обработка аварийного завершения.

Особенности реализации:
- Неблокирующий ввод (`kbhit`);
- Статистика работы (добавленные/извлеченные сообщения);
- Визуализация статуса системы (команда '`s`').

3 ФУНКЦИОНАЛЬНАЯ СТРУКТУРА ПРОЕКТА

Основной процесс (управляющий поток), реализован через функцию `main`, выполняет инициализацию очереди, запускает потоки производителей и потребителей, обрабатывает ввод пользователя с клавиатуры и обеспечивает завершение программы.

Основные действия:

1) Инициализация очереди и всех синхронизационных объектов (`init_queue()`);

2) Обработка пользовательского ввода:

`p` — запуск нового производителя (`create_producer()`);

`c` — запуск нового потребителя (`create_consumer()`);

`P` — остановка одного производителя (`stop_producer()`);

`C` — остановка одного потребителя (`stop_consumer()`);

`s` — вывод текущего состояния очереди (`show_status()`);

`q` — завершение всех потоков и освобождение ресурсов (cleanup()).

3) Обработка сигналов завершения (`SIGINT`, `SIGTERM`) через `signal_handler(int sig)`;

4) Управление списками потоков производителей и потребителей.

Производитель (поток), реализован через функция: `void* producer_thread(void* arg)`.

Параметры:

`int id` — идентификатор производителя (передаётся через `arg` для логирования).

Основные действия:

1) Создание сообщения через `create_message()`;

2) Проверка необходимости отложенного уменьшения очереди (`check_and_perform_resize()`);

3) Ожидание наличия свободного слота (условие `cond_empty`);

4) Захват мьютекса (`queue_mutex`) и помещение сообщения в очередь;

5) Обновление индексов и счётчиков (`tail`, `free`, `added`);

6) Освобождение мьютекса и сигнал потоку-потребителю (`cond_full`);

7) Печать информации о добавленном сообщении;

8) Пауза перед следующей итерацией (`usleep`);

9) Корректное завершение по флагу `should_terminate`.

Потребитель (поток), реализован через функция: `void* consumer_thread(void* arg)`.

Параметры:

`int id` — идентификатор потребителя.

Основные действия:

1) Ожидание появления сообщений (условие `cond_full`);

2) Захват мьютекса (`queue_mutex`) и извлечение сообщения;

3) Обновление индексов и счётчиков (`head`, `free`, `extracted`);

4) Освобождение мьютекса и сигнал производителям (`cond_empty`);

5) Проверка хеш-суммы (целостность данных) через `calculate_hash()`;

6) Печать информации о сообщении и результате проверки;

7) Обработка отложенного уменьшения очереди (если необходимо);

8) Пауза между итерациями;

9) Корректное завершение при установке `should_terminate`.

Поддержка динамического изменения размера очереди, реализована через функции:

`resize_queue(int new_size)` — изменение размера очереди;

`check_and_perform_resize()` — проверка возможности уменьшения;

`can_decrease_queue_size()` — логика условий безопасного уменьшения;

Флаги: `resize_in_progress`, `resize_decrease_pending`, `pending_resize`, `resize_target_size`;

Семафорная синхронизация: resize_mutex, resize_cond;

Зарезервированные ячейки reserved_slots защищают от потери данных при уменьшении.

Вспомогательные функции:

`init_queue()` — инициализация очереди: выделение памяти, установка начальных значений индексов и счётчиков.

`create_message(Message* msg)` — генерация случайного сообщения с хешем и выравниванием.

`calculate_hash(Message* msg)` — вычисление простой хеш-суммы сообщения.

`kbhit()` — неблокирующая проверка ввода с клавиатуры.

`signal_handler(int sig)` — установка флага завершения и пробуждение всех потоков.

`create_producer()` / `create_consumer()` — запуск потоков с учётом ограничений по количеству.

`stop_producer()` / `stop_consumer()` — остановка потоков через `pthread_cancel` и `pthread_detach`.

`show_status()` — вывод информации о текущем состоянии очереди.

`cleanup()` — освобождение всех ресурсов: завершение потоков, удаление очереди, уничтожение мьютексов и переменных условия.

Структуры данных:

`Message` — сообщение в очереди, содержит:

`type` — тип сообщения;

`size` — размер полезной нагрузки;

`data[]` — буфер данных;

`hash` — хеш-сумма для проверки целостности.

`Queue` — структура динамической очереди:

`Message* buffer` — буфер сообщений;

`int head`, `tail` — индексы извлечения и добавления;

`int free` — количество свободных ячеек;

`int current_size` — текущий размер буфера;

`unsigned long added`, `extracted` — счётчики сообщений;

`int reserved_slots` — зарезервированные слоты при уменьшении размера.

4 ПОРЯДОК СБОРКИ И ЗАПУСКА

1) Перейти в каталог проекта:

```
$ cd 'Гожуленко М.Ю./lab05'
```

2) Собрать проект с помощью make:

```
$ make
```

3) Запустить программу:

```
$ ./threadProc
```

5 РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

```
Система производителей и потребителей запущена
Управление:
p - создать производителя
c - создать потребителя
P - остановить производителя
C - остановить потребителя
s - показать статус
q - выход
pСоздан производитель #1 (PID: 8017)
Производитель 0: добавлено сообщение (тип=1, размер=52, хеш=39887),
всего добавлено
: 1
Производитель 0: добавлено сообщение (тип=3, размер=117,
хеш=51491), всего добавлен
о: 2
Производитель 0: добавлено сообщение (тип=2, размер=89, хеш=50380),
всего добавлено
: 3
Производитель 0: добавлено сообщение (тип=7, размер=144, хеш=7957),
всего добавлено
: 4
Производитель 0: добавлено сообщение (тип=4, размер=204,
хеш=59803), всего добавлен
о: 5
cСоздан потребитель #1 (PID: 8018)
Потребитель 0: извлечено сообщение (тип=1, размер=52, хеш=39887),
проверка хеша: OK
, всего извлечено: 1
Производитель 0: добавлено сообщение (тип=9, размер=183,
хеш=16550), всего добавлен
о: 6
Производитель 0: добавлено сообщение (тип=3, размер=173,
хеш=34444), всего добавлен
о: 7
Потребитель 0: извлечено сообщение (тип=3, размер=117, хеш=51491),
проверка хеша: O
K, всего извлечено: 2
Производитель 0: добавлено сообщение (тип=0, размер=199,
хеш=13508), всего добавлен
о: 8
Производитель 0: добавлено сообщение (тип=8, размер=129,
хеш=37108), всего добавлен
о: 9
Потребитель 0: извлечено сообщение (тип=2, размер=89, хеш=50380),
проверка хеша: OK
```

```
, всего извлечено: 3
Производитель 0: добавлено сообщение (тип=9, размер=158,
хеш=45252), всего добавлен
о: 10
Остановлен производитель (PID: 8017)
Остановлен потребитель (PID: 8018)
Предотвращение тупика: создание производителя и потребителя
Создан производитель #1 (PID: 8019)
Создан потребитель #1 (PID: 8020)
qЗавершение работы...
Программа завершена
```