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
Маслаков Н. А.

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

Наша программа представляет собой классическую иллюстрацию задачи «производители – потребители», где несколько потоков с общим доступом к памяти координируются для безопасной передачи сообщений. Главный поток отвечает за создание и настройку очереди, за её динамическое расширение и сжатие по запросам пользователя, а также за запуск и завершение работающих в фоне потоков-производителей и потоков-потребителей. Каждый «производитель» непрерывно генерирует сообщение с некоторыми случайными данными и контрольной суммой, выполняет попытку поместить его в очередь, дожидаясь освобождения места, и после успешной вставки сигналит другим потокам, что новая порция данных готова. Параллельно «потребители» забирают эти сообщения, проверяют контрольную сумму, отображают результат и освобождают занятую память.

Ключевым элементом синхронизации выступает один мьютекс, который защищает целостность внутренней структуры очереди, и две условные переменные. Первая переменная позволяет «производителям» останавливаться в ожидании, пока в буфере не появится свободное место, вторая – «потребителям», если очередь пуста. Благодаря такому механизму потоки не расходуют лишнего процессорного времени на активное ожидание, а при изменении размера очереди (командами «+» и «–») под тем же мьютексом перераспределяется пространство буфера, после чего все ожидающие потоки автоматически пробуждаются и повторно проверяют условия продолжения работы. Это гарантирует отсутствие гонок и дедлоков при параллельном доступе к общему ресурсу.

По сравнению с более старой версией на семафорах и разделяемой памяти между процессами, где требовались System V–семафоры и выделение общих сегментов, здесь все сущности живут внутри одного процесса и взаимодействуют напрямую через динамическую кучу. Запуск и остановка потоков с помощью `pthread_create`, `pthread_cancel` и `pthread_join` происходит гораздо быстрее и проще, чем `fork` / `kill` в многопроцессном варианте. Преимущество же использования условных переменных в том, что они позволяют чётко разделить архитектуру ожидания и сигналов, сохраняя при этом лёгкость понимания и масштабирования: буфер можно увеличивать и уменьшать в любых моментах работы, не меняя базовой логики взаимодействия.

3 ФУНКЦИОНАЛЬНАЯ СТРУКТУРА ПРОЕКТА

Основной поток выполнения.

`void create_producer()`.

Функция `create_producer` создает потребителя.

`void create_consumer()`.

Функция `create_consumer` используется для создания производителя.

`void remove_last_producer()`.

Функция `remove_last_producer` используется для удаления последнего добавленного производителя.

`void remove_last_consumer()`.

Функция `remove_last_consumer` используется для удаления последнего добавленного потребителя.

`void remove_all()`.

Функция `remove_all_procs` используется для удаления всех потоков.

`void print_status()`.

Функция `print_status` используется для вывода статуса программы.

Производитель

`void producer_task()`.

Функция `producer_task` используется для выполнения основной логики производителя.

Потребитель

`void consumer_task()`.

Функция `consumer_task` используется для выполнения основной логики потребителя.

Очередь

`Queue* queue_init()`.

Функция `queue_init` используется для инициализации очереди.

`void queue_push(Queue* q, Message* msg)`.

Функция `queue_push` используется для добавления элемента в очередь.

Принимаемые параметры:

1) `Queue* q`. Сама очередь.

2) `Message* msg`. Сообщение для добавления.

`Message* queue_pop(Queue* q)`.

Функция `queue_ pop` используется для удаления элемента из очереди.

Принимает параметр `Queue* q`(сама очередь).

`void queue_destroy(Queue* q).`

Функция `queue_destroy` используется для корректного удаления очереди.

Принимает параметр `Queue* q`(сама очередь).

`uint16_t calculate_hash(Message* msg).`

Функция `calculate_hash` используется для корректного удаления очереди.

Принимает параметр `Message* msg`(сообщение, для которого вычисляется хэш).

Для задания 5.2 все сигнатуры функции остаются теми же, меняется только внутренняя логика.

4 ПОРЯДОК СБОРКИ И ЗАПУСКА

1) Перейти в каталог проекта.

$ `cd 'Маслаков Н.А./lab05'`

2) Собрать проект с помощью `make`. по умолчанию сборка происходит в режиме отладки.

$ `make`

3) Запустить программу.

$ `./build/debug/app`

При необходимости можно изменить `Makefile` для выбора варианта задания.

5 РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

```
hechert@fedora:~/tar_working_dir/Маслаков_Н.А./lab05/build/
debug$ ./app
Controls:
 p - Add producer
 c - Add consumer
 P - Remove last producer
 C - Remove last consumer
 + - Increase queue size
 - - Decrease queue size
 k - Kill all threads
 s - Show status
 q - Quit
pp
[Main] Producer thread created (id=139813216396992)
[Main] Producer thread created (id=139813208004288)
[Producer 139813208004288] Added |type:132 hash:16061 size:140|
Total:2
[Producer 139813216396992] Added |type:103 hash:20565 size:146|
Total:2
cc
[Main] Consumer thread created (id=139813199611584)
[Main] Consumer thread created (id=139813191218880)
[Consumer 139813199611584] Removed |type:132 hash:16061 size:140|
Total:1
[Consumer 139813191218880] Removed |type:103 hash:20565 size:146|
Total:2
[Producer 139813216396992] Added |type:219 hash:29652 size:231|
Total:3
[Producer 139813208004288] Added |type:122 hash:13601 size:104|
Total:4
[Producer 139813216396992] Added |type:159 hash:30946 size:232|
Total:5
[Consumer 139813199611584] Removed |type:219 hash:29652 size:231|
Total:3
[Producer 139813208004288] Added |type:226 hash:17127 size:128|
Total:6
[Producer 139813216396992] Added |type:58 hash:1067 size:7| Total:7
```

```
s

--- Status ---
Queue: 4/0 (used/free)
Messages: added=7, removed=3
Active producers: 2
Active consumers: 2

[Consumer 139813191218880] Removed |type:122 hash:13601 size:104|
Total:4
[Producer 139813216396992] Added |type:239 hash:2258 size:19|
Total:8
+
[Main] Queue increased: capacity = 5
[Producer 139813208004288] Added |type:178 hash:7482 size:52|
Total:9
[Consumer 139813199611584] Removed |type:159 hash:30946 size:232|
Total:5
[Producer 139813216396992] Added |type:38 hash:4802 size:44|
Total:10
[Consumer 139813191218880] Removed |type:226 hash:17127 size:128|
Total:6
[Producer 139813208004288] Added |type:106 hash:2537 size:18|
Total:11
s

--- Status ---
Queue: 5/0 (used/free)
Messages: added=11, removed=6
Active producers: 2
Active consumers: 2

q
[Main] Producer thread removed (id=139813208004288)
[Main] Producer thread removed (id=139813216396992)
[Main] Consumer thread removed (id=139813191218880)
[Main] Consumer thread removed (id=139813199611584)
Program terminated
```