Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей
Кафедра электронных вычислительных машин

Лабораторная работа №5
по курсу
«Операционные системы и системное программирование»
на тему
«Потоки исполнения, взаимодействие и синхронизация»

Выполнил: студент группы 350501
Русак Г.Д.
Проверил: старший преподаватель каф. ЭВМ
Поденок Л.П.

Минск 2025

1 ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

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.

Программы компилируются с ключами

```
-W -Wall -Wextra -std=c11 -pedantic
```

Допускается использование ключей

```
-Wno-unused-parameter -Wno-unused-variable
```

Для компиляции, сборки и очистки используется `make`.

2 ОПИСАНИЕ АЛГОРИТМОВ И РЕШЕНИЙ

Программа реализует классическую модель взаимодействия производителей и потребителей с использованием разделяемой очереди сообщений на основе кольцевого буфера. Вся работа выполняется в рамках одного процесса с использованием многопоточности. Основной поток управляет созданием рабочих потоков-производителей и потребителей, а также обеспечивает их синхронизацию.

Архитектура системы построена вокруг кольцевого буфера, который содержит указатели на сообщения, индексы головы и хвоста, текущее количество сообщений и максимальную емкость. Для синхронизации доступа используются мьютексы, семафоры `POSIX` или условные переменные. Мьютекс защищает критическую секцию при работе с разделяемыми данными, а условные переменные обеспечивают эффективное ожидание изменения состояния буфера.

Потоки-производители генерируют сообщения случайного размера, используя функцию `rand()`. Каждое сообщение содержит тип, контрольную сумму `CRC16`, размер данных и сами данные. Перед добавлением в очередь производитель блокирует мьютекс и ожидает появления свободного места через условную переменную `not_full`. После добавления сообщения производитель сигнализирует потребителям через переменную `not_empty` о появлении новых данных.

Потоки-потребители аналогичным образом получают доступ к очереди, ожидая появления сообщений через условную переменную `not_empty`. После извлечения сообщения потребитель проверяет его целостность путем повторного вычисления `CRC16` и сравнения с хранящимся значением. В случае несоответствия выводится предупреждение. Освободившееся место сигнализируется производителям через переменную `not_full`.

Основной поток предоставляет интерфейс управления, позволяющий динамически создавать и завершать потоки производителей и потребителей, изменять размер буфера и выводить статистику работы системы. Завершение работы выполняется корректно, с оповещением всех потоков и освобождением ресурсов.

3 ФУНКЦИОНАЛЬНАЯ СТРУКТУРА ПРОЕКТА

Проект реализует модель взаимодействия производителей (`producers`) и потребителей (`consumers`) через общую очередь сообщений с использованием многопоточного взаимодействия и синхронизации. Система состоит из нескольких ключевых модулей, каждый из которых выполняет определенную функцию.

Главный управляющий модуль (`main.c`) выполняет следующие функции:

1) Инициализация системы. Создание и настройка кольцевого буфера, установка обработчика сигнала через `handle_sigint()`;

2) Управление синхронизационными примитивами в функциях `initialize_sync_primitives()` и `cleanup_sync_primitives()`;

3) Создание сообщения в функции `generate_message()`;

4) Управление потоками;

5) Обработка пользовательского ввода;

6) Завершение работы и освобождение ресурсов в функции `graceful_shutdown()`.

Модуль работы с кольцевым буфером реализован в файле `ring.c` и выполняет следующие функции:

1) Добавление элементов в буфер с помощью функции `push()`;

2) Извлечение элементов из буфера `pop()`;

3) Инициализация сообщений с помощью функции `init_mes()`;

4) Вывод сообщения с помощью функции `print_mes()`;

5) Освобождение памяти сообщения с помощью функции `mes_clear()`;

6) Освобождение памяти буфера с помощью функции `ring_clear()`;

7) Вычисление контрольной суммы с помощью функции `crc16()`.

Модуль работы с потоками и синхронизацией реализован в файле `utils.c` и выполняет следующие функции:

1) Обработка сигналов с помощью функции `thread_stop_handler()`;

2) Реализация потребителей с использованием семафоров с помощью функции `consumer_routine()`;

3) Реализация производителей с использованием семафоров с помощью функции `producer_routine()`;

4) Реализация потребителей с использованием условных переменных с помощью функции `consumer_routine_cond()`;

5) Реализация производителей с использованием условных переменных с помощью функции `producer_routine_cond()`.

4 ПОРЯДОК СБОРКИ И ЗАПУСКА ПРОЕКТА

Порядок сборки и запуска состоит в следующем:

1) Клонировать репозиторий, используя команду

`$git clone https://github.com/Everolfe/lab05-OSASP`,

или разархивировать каталог с проектом;

2) Перейти в каталог с проектом

`$cd lab04-OSASP`,

или

`$cd "Русак Г.Д./lab05"`;

3) Собрать проект используя `make`;

4) После сборки проекта можно использовать, прописав

`$./build/release/ipc/ipc`.

5 РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ

1)

```
grusak@fedora:~/tar_working_dir/Русак Г.Д./lab05$
./build/release/ipc/ipc
Select synchronization method:
1 - POSIX semaphores and mutex
2 - Conditional variables
Your choice: 1
=== IPC Producer-Consumer Program ===
Using POSIX semaphores
Commands:
  p - Add new producer thread
  c - Add new consumer thread
  r - Remove last producer thread
  d - Remove last consumer thread
  s - Show statistics
  + - Increase ring buffer size
  - - Decrease ring buffer size
  q - Quit program (graceful shutdown)
  h - help
Current ring size: 10
```

2)

```
--Append 5 message:
Message   type:   0,   hash:   0a20,   size:   58,   data:
jsUYibYebMWsiQYoyGYXYMZEVypzvJegEBeoCFUfTsxDiXtigsIeeHkCHz
--Ejected 4 message:
Message   type:   0,   hash:   0a20,   size:   58,   data:
jsUYibYebMWsiQYoyGYXYMZEVypzvJegEBeoCFUfTsxDiXtigsIeeHkCHz
```

3)

```
====================
Added: 14
Getted: 8
Producers count: 3
Consumers count: 2
Current size: 6
Max size: 10
====================
```

4)

```
Producer 6 has finished
Last producer thread removed
r
No producer threads to remove
Consumer 7 has finished
Last consumer thread removed
d
No consumer threads to remove
```