Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей
Кафедра электронных вычислительных машин

Лабораторная работа №4
по курсу
«Операционные системы и системное программирование»
на тему
«Задача производители-потребители для процессов»

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

Минск 2025

1 ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

Основной процесс создает очередь сообщений, после чего ожидает и обрабатывает нажатия клавиш, порождая и завершая процессы двух типов — производители и потребители.

Очередь сообщений представляет собой классическую структуру — кольцевой буфер, содержащий указатели на сообщения, и пара указателей на голову и хвост. Помимо этого очередь содержит счетчик добавленных сообщений, счетчик извлеченных и количество свободного места в очереди.

Производители формируют сообщения и, если в очереди есть место, перемещают их туда. Потребители, если в очереди есть сообщения, извлекают их оттуда, обрабатывают и освобождают память с ними связанную.

Для работы используются два семафора для заполнения и извлечения, а также мьютекс или одноместный семафор для монопольного доступа к очереди.

Производители генерируют сообщения, используя системный генератор случайных чисел `rand(3)` или `rand_r(3)` для `size` и `data`. В качестве результата для `size` используется остаток от деления на 256. Реальный размер сообщения на единицу больше и лежит в интервале (1, 256).

Поле `data` имеет длину, кратную 4-м байтам. При формировании сообщения контрольные данные формируются только из байт сообщения длиной `size` + 1. Значение поля `hash` при вычислении контрольных данных принимается равным нулю.

Для расчета контрольных данных можно использовать любой подходящий алгоритм на выбор студента.

После помещения значения в очередь перед освобождением мьютекса очереди производитель инкрементирует счетчик добавленных сообщений. Затем после освобождения мьютекса выводит строку на `stdout`, содержащую помимо всего новое значение этого счетчика.

Потребитель, получив доступ к очереди, извлекает сообщение и удаляет его из очереди.

Перед освобождением мьютекса очереди инкрементирует счетчик извлеченных сообщений. Затем после освобождения мьютекса проверяет контрольные данные и выводит строку на `stdout`, содержащую помимо всего новое значение счетчика извлеченных сообщений.

При получении сигнала о завершении процесс должен завершить свой цикл и только после этого завершиться, не входя в новый.

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. Архитектура системы включает главный управляющий процесс, процессы-производители и процессы-потребители, синхронизированные через механизмы межпроцессного взаимодействия.

Основной процесс выполняет инициализацию системы: создает очередь сообщений, инициализирует структуры данных для кольцевого буфера и настраивает необходимые объекты синхронизации. Очередь сообщений организована как кольцевой буфер фиксированного размера, содержащий указатели на сообщения, индексы головы и хвоста, а также счетчики добавленных и извлеченных сообщений. Для управления доступом к разделяемым ресурсам используются два счетных семафора `POSIX` (для контроля заполненности и свободного места) и один семафор функционирующий как мьютекс.

Процессы-производители генерируют сообщения случайного размера, используя системный вызов `rand()`. Каждое сообщение содержит поле типа, контрольные данные, размер полезной нагрузки и сами данные. Контрольные данные вычисляются с использованием алгоритма `CRC16` на основе содержимого сообщения. Перед добавлением в очередь производитель проверяет наличие свободного места через соответствующий семафор, после чего получает эксклюзивный доступ к очереди через мьютекс, добавляет сообщение, обновляет счетчики и индексы, а затем освобождает ресурсы.

Процессы-потребители работают аналогично: проверяют наличие сообщений через семафор заполненности, получают доступ к очереди, извлекают сообщение, обновляют состояние буфера и счетчики. После извлечения потребитель проверяет целостность сообщения путем повторного вычисления контрольных данных и сравнивает полученное значение с хранящимся в сообщении. В случае несоответствия выводится предупреждение. Память, выделенная под сообщение, освобождается после обработки.

3 ФУНКЦИОНАЛЬНАЯ СТРУКТУРА ПРОЕКТА

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

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

1) Инициализация системы. Создание и настройка разделяемой памяти, семафоров и очереди сообщений и обработчика сигналов;

2) Управление именованными семафорами в функциях `initialize_semaphores()` и `close_semaphores()`;

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

4) Вывод информации о сообщении в функции display_message();

5) Реализация алгоритма `CRC16` для подсчета контрольных данных в функции `crc16()`;

6) Создание процесса-производителя в функции `producer()`;

7) Создание процесса-потребителя в функции `consumer()`;

8) Обработка сигнала завершения в функции `handler_stop_proc()`;

9) Обработка пользовательского ввода в функции `menu()`.

Завершение работы и освобождение ресурсов.

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

1) Инициализация очереди с помощью функции `init_ring()`;

2) Добавление сообщений с помощью функции `push_message()`;

3) Извлечение сообщений с помощью функции `pop_message()`;

4) Освобождение памяти с помощью функции `clear_buff()`.

Для хранения потребителей и прозводителей используется стек, реализованый в файле `stack.c`. Данный модуль выполняет следующие функции:

1) Добавление нового элемента на вершину стека `push_stack()`;

2) Удаление верхнего элемента из стека `pop_stack()`;

3) Возвращение `PID` из вершины стека без удаления `get_top_pid()`;

4) Очистка стека `free_stack()`;

5) Возвращение количества элементов стека `stack_size()`.

4 ПОРЯДОК СБОРКИ И ЗАПУСКА ПРОЕКТА

Порядок сборки и запуска состоит в следующем:

1) Клонировать репозиторий, используя команду

`$git clone https://github.com/Everolfe/lab04-OSASP`,

или разархивировать каталог с проектом;

2) Перейти в каталог с проектом

`$cd lab04-OSASP`,

или

`$cd "Русак Г.Д./lab04"`;

3) Собрать проект используя `make`;

4) После сборки проекта можно использовать, прописав

`$./build/release/ipc`

5 РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ

1)

```
$./build/release/ipc
Shmid segment : 393261
############################
Select an action:
p - Add a producer
c - Add a consumer
d - Delete last producer
k - Delete last consumer
s - Show status
q - Quit
############################
p
```

2)

```
pid: 3666 produce msg: hash=FE1A (total: 14)
pid: 3666 produce msg: hash=D74E (total: 15)
pid: 3666 skipping - no free slots
pid: 3669 consume msg: hash=FE1A (total: 14)
pid: 3669 consume msg: hash=D74E (total: 15)
pid: 3669 skipping - no messages available
```

3)

```
=== Queue Status ===
  Buffer size:    15
  Free slots:     13
  Used slots:     2
  Active producers: 2
  Active consumers: 3
  Total produced: 24
  Total consumed: 22
====================
```

4)

```
pid: 3724 produce msg: hash=4EC3 (total: 2)
pid: 3722 produce msg: hash=5642 (total: 3)
Producer (PID: 3724) terminated
Producer (PID: 3722) terminated
pid: 3726 consume msg: hash=4EC3 (total: 2)
pid: 3725 consume msg: hash=5642 (total: 3)
Consumer (PID: 3726) terminated
Consumer (PID: 3725) terminated
```