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

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

по теме

ЗАДАЧА ПРОИЗВОДИТЕЛИ-ПОТРЕБИТЕЛИ ДЛЯ ПРОЦЕССОВ

БГУИР КР 1-40 02 01 001 ЛР

Выполнил: студент группы 250541,
Бобрик В. Ю.

Проверил: ст.преподаватель каф. ЭВМ,
Поденок Л. П.

Минск 2025

СОДЕРЖАНИЕ

1 Условия лабораторной работы

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 мьютексили одноместный семафор для монопольного доступа к очереди.

Сообщения имеют следующий формат (размер и смещение в байтах):

Имя	Размер	Смещение	Описание
`type`	1	0	тип сообщения
`hash`	2	1	контрольные данные
`size`	1	3	длина данных в байтах (от 0 до 256)
`data`	((`size`+3)/4)*4	4	данные сообщения

Производители генерируют сообщения, используя системный генератор `rand(3)` для `size` и `data`. В качестве результата для `size` используется остаток от деления на 257.

Если остаток от деления равен нулю, `rand(3)` вызывается повторно. Если остаток от деления равен 256, значение `size` устанавливается равным 0, реальная длина сообщения при этом составляет 256 байт. Таким образом,

размер сообщения лежит в интервале (1, 256). Поле дата имеет длину, кратную 4-м байтам.

При формировании сообщения контрольные данные формируются из всех байт сообщения, исключая байты за концом поля `data`, если они есть. 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.

Следует предусмотреть задержки, чтобы вывод можно было успеть прочитать в процессе работы программы.

Следует предусмотреть защиту от тупиковых ситуаций из-за отсутствия производителей или потребителей.

Следует предусмотреть нажатие клавиши для просмотра состояния (размер очереди, сколько занято и сколько свободно, столько производителей и сколько потребителей).

2 Описание алгоритмов и решений

Очередь сообщений реализована в разделяемой памяти как кольцевой буфер, где используются два индекса: `tail` для добавления новых сообщений (производителям), и `head` для извлечения сообщений (потребителям).

При добавлении сообщение помещается по индексу `tail`, после чего указатель обновляется по формуле:

```
tail = (tail + 1) % QUEUE_SIZE
```

Аналогично, извлечение происходит из позиции `head`, после чего происходит циклический переход:

```
head = (head + 1) % QUEUE_SIZE
```

Для защиты очереди от гонок и обеспечения корректного доступа введены три семафора:

Семафор свободных слотов (индекс 0): инициализируется значением, равным размеру очереди (`QUEUE_SIZE`). Производитель перед внесением нового сообщения уменьшает значение этого семафора, чтобы убедиться, что в очереди есть свободное место.

– Семафор заполненных слотов (индекс 1): изначально равен 0. После добавления сообщения производитель увеличивает этот семафор, сигнализируя потребителю, что есть сообщение на обработку. Потребитель, в свою очередь, ожидает, когда значение этого семафора станет положительным.

Семафор-мьютекс (индекс 2): используется для обеспечения эксклюзивного доступа к общему объекту `MessageQueue` во время операций добавления или извлечения сообщений. Он инициализируется единицей, что позволяет в один момент времени только одному процессу изменять поля очереди.

Алгоритм работы производителя:

Шаг 1. Генерация сообщения.

Шаг 1.1. Тип сообщения (`type`): случайная латинская заглавная буква (`'A' + rand() % 26`).

Шаг 1.2. Формирование размера (`size`): вызывается `rand() % 257` для выбора размера. Если результат равен 0, генерация повторяется, так как размер не может быть нулевым. Если результат равен 256, по соглашению в поле `size` записывается 0, что трактуется как реальная длина, равная 256 байтам.

Шаг 1.3. Подготовка данных (`data`): для обеспечения кратности длины 4-м байтам вычисляется `padded_len = ((actual_len + 3) / 4) * 4`. Первые `actual_len` байт заполняются случайными буквами, оставшиеся байты дополняются нулевыми значениями.

Шаг 2. Вычисление контрольной суммы:

Шаг 2.1. Итерация по `N` байтам, где `N` — либо значение поля `size`, либо 256, если `size = 0`.

Шаг 2.2. Каждый байт приводится к типу `unsigned char` и добавляется к суммарной переменной `hash`.

Шаг 3. Добавление в очередь

Шаг 3.1. Ожидание свободного места. Производитель уменьшает значение семафора 0 (свободных слотов). Если свободных мест нет, процесс блокируется.

Шаг 3.2. Монопольный доступ. Производитель захватывает мьютекс (семафор 2), чтобы иметь эксклюзивный доступ к структуре очереди.

Шаг 3.3. Запись сообщения. Сообщение записывается в позицию `tail`, затем `tail` обновляется, а `free_slots` уменьшается.

Шаг 3.4. Обновление статистики. Счетчик добавленных сообщений (`added_count`) инкрементируется.

Шаг 3.5. После записи мьютекс освобождается, и семафор 1 (заполненных слотов) увеличивается.

Алгоритм работы потребителя:

Шаг 1. Ожидание сообщения.

Шаг 1.1. Ожидание заполненного слота посредством уменьшения семафора 1 (заполненных слотов). Если сообщений нет, процесс блокируется.

Шаг 1.2. Потребитель захватывает мьютекс (семафор 2) для безопасного доступа к очереди.

Шаг 2. Извлечение сообщения.

Шаг 2.1. Сообщение извлекается из позиции `head`, после чего указатель обновляется.

Шаг 2.2. Обновляется количество свободных слотов и увеличивается счетчик извлеченных сообщений (`removed_count`).

Шаг 3. Контроль целостности. После освобождения мьютекса потребитель пересчитывает хэш посчитав сумму байтов извлеченных данных. Если вычисленная контрольная сумма совпадает с сохраненным в сообщении, сообщение считается корректным, иначе отмечается как поврежденное.

Шаг 4. Оповещение об окончании операции. Мьютекс освобождается, и семафор 0 (свободных слотов) увеличивается для информирования производителей о наличии свободных мест.

Основной процесс выделяет разделяемую память для структуры `MessageQueue` и инициализирует семафоры.

3 Функциональная структура проекта

Модуль `common.h` содержит общие определения, константы и структуры данных, используемые всеми компонентами проекта.

`QUEUE_SIZE`: фиксированный размер кольцевого буфера для сообщений.

`SHM_KEY`: ключ для создания/подключения к разделяемой памяти.

`SEM_KEY`: ключ для создания/подключения к набору семафоров.

структура `message` представляет отдельное сообщение.

```
typedef struct {
    char type;                  // Тип сообщения
    unsigned short hash;        // Контрольная сумма
    unsigned char size;         // Размер данных
    char data[256];             // Данные сообщения
} message;
```

структура message_queue описывает очередь сообщений в виде кольцевого буфера.

```
typedef struct {
    Message buffer[QUEUE_SIZE]; // Кольцевой буфер
    int head;                   // Индекс извлечения (голова)
    int tail;                   // Индекс добавления (хвост)
    int added_count;           // Количество добавленных сообщений
    int removed_count;         // Количество извлеченных сообщений
    int free_slots;              // Количество свободных слотов в буфере
} message_queue;
```

Модуль ipc определяет и реализует функции, связанные с операциями межпроцессного взаимодействия: работу с семафорами и вычисление контрольной суммы сообщений.

semaphore_op() – выполняет операцию над семафором в семафорном наборе. Принимает идентификатор набора семафоров, номер конкретного семафора и операцию. Внутри создается структура sembuf, после чего производится операция с помощью semop.

calculate_hash() – вычисляет контрольную сумму для сообщения.

verify_hash() – проверяет корректность сообщения путем пересчета хэша и сравнения его с сохраненным в поле hash.

Модуль main реализует основной процесс системы, отвечающий за инициализацию ресурсов, обработку пользовательского ввода, запуск

производителей и потребителей и мониторинг состояния очереди для предотвращения дедлоков.

`check_deadlock()` – проводит проверку: если активных производителей или потребителей нет и очередь находится в одном из крайних состояний (полностью свободной или полностью заполненной), выводит предупреждение.

`deadlock_monitor()` – функция потока, которая вызывает `check_deadlock()` каждые 5 секунд.

`init_resources()` – создает и инициализирует разделяемую память для очереди сообщений, инициализирует очередь, создает набор семафоров и инициализирует их.

`cleanup_resources()` – завершает работу системы: устанавливается флаг завершения, отправляются сигналы `SIGTERM` всем дочерним процессам, происходит отсоединение и удаление общей памяти и семафорного набора.

`print_status()` – выводит текущее состояние очереди.

`handle_commands()` – реализует цикл обработки пользовательского ввода с командами.

`main()` – точка входа в программу. Вызывает `init_resources()` для подготовки разделяемой памяти и семафоров, создает поток мониторинга дедлоков, вызывающий `deadlock_monitor()`, запускает цикл обработки команд `handle_commands()`.

Модуль `producer.c` – процесс-производитель, генерирующий случайные сообщения и добавляющий их в очередь.

`termination_handler` – обработчик `SIGTERM`, который устанавливает флаг завершения `terminate_flag`.

`main()` – точка входа процесса. Подключается к разделяемой памяти и набору семафоров, инициализирует генератор случайных чисел, циклически генерирует сообщения и добавляет их в очередь.

Модуль `consumer.c` – процесс-потребитель, извлекает сообщения из очереди и осуществляет их проверку.

`termination_handler` – обработчик `SIGTERM`, который устанавливает флаг завершения `terminate_flag`.

`main()` – точка входа процесса. Подключается к разделяемой памяти и набору семафоров, ожидает сообщения, извлекает сообщения из очереди, сверяет контрольную сумму и выводит информацию о сообщении.

4 Порядок сборки и тестирования

Для сборки используется `Makefile`, содержащий следующие ключевые элементы:

Исходные файлы находятся в каталоге `./src`.

Заголовочные файлы расположены в каталоге `./include`.

В зависимости от типа сборки создается каталог `./out/debug` для `debug`-версии или `./out/release` для `release`-версии.

Переменная `BUILD` задается как `debug` по умолчанию. При запуске `make release` устанавливается оптимизированная сборка.

Флаги компиляции задаются через переменные `COMMON_CFLAGS` (определения, предупреждения, стандарт `C11`) и дополняются флагами отладки (`-g -O0`) или оптимизации (`-O2`).

Для сборки в `debug`-режиме необходимо выполнить в терминале:

```
make
```

Это соберет программы в `./out/debug/.`

Для сборки в `release`-режиме необходимо выполнить в терминале:

```
make release
```

Это соберет программы в `./out/release/.`

Для очистки сборки необходимо выполнить в терминале:

```
make clean
```

Для тестирования работы программы выполнить:

```
cd out/release/
./parent
```

5 Методика и результаты тестирования

Программа тестируется вручную пошагово.

При запуске программы `parent` выводится следующая запись:

```
[silvarious@fedora release]$ ./parent
Запуск основного процесса.
Команды: '+' для запуска производителя, '-' для запуска потребителя, 'p' для просмотра состояния, 'q' для выхода.
```

Ввести команду «p». Ожидаемый вывод должен включать количество добавленных и извлечённых сообщений (изначально 0), количество свободных слотов, равное `QUEUE_SIZE`, отсутствие активных процессов-производителей и потребителей.

```
p
Состояние очереди:
Добавлено сообщений: 0
Извлечено сообщений: 0
Свободные места: 10
Занятые места: 0
Производителей: 0, Потребителей: 0
```

Ввести команду «+». Ожидается запуск нового производителя, сообщение о его запуске и каждые 3 секунды добавление новых сообщений.

```
+
Производитель 4047 запущен.
Производитель 4047: добавлено сообщение (тип 'S', размер 182, hash 14283). Всего добавлено: 1
Производитель 4047: добавлено сообщение (тип 'W', размер 177, hash 13851). Всего добавлено: 2
```

```
Производитель 4047: добавлено сообщение (тип 'N', размер 214, hash 16665). Всего добавлено: 3
```

Ввести команду «–». Ожидается запуск нового потребителя, сообщение о его запуске и каждые 3 секунды – извлечение новых сообщений.

```
-
Потребитель 4052 запущен.
Потребитель 4052: извлечено сообщение (тип 'S', размер 182, hash 14283, данные: "MRSMOJXJEKXLUQDBUCWNRFDMTROXRHKEYCSONRZRBYCXQIYLMXYDCDSVWGVONFUOJMEYDGPEETEVBCIOZGTCLLZHSUXHBRXMFCKIIZPOVTJWXRMXZIBMTAUNXTXALUNQYXBIZSXWNISKCHJDPKQKLMAKFXKQTXJUXMCYGBUTKOFOVPRMZHZMVZ"). Всего извлечено: 1
Потребитель 4052: сообщение корректно.
Производитель 4047: добавлено сообщение (тип 'W', размер 114, hash 8789). Всего добавлено: 11
Потребитель 4052: извлечено сообщение (тип 'W', размер 177, hash 13851, данные: "YHTTGENDSQBYRXTDMZRJQLYRUXEQYARYJMRQRFVJXXKQWDUIENUUYSMTRQJPSCNCOGUHNPTKODDLGZVNMPHNJVGALRRGVEKLLEVAVOLKRQXZPSMDKWQTRYWFRNLOUXAHDVHYLUKEKHDZASDMOVFHWDMNRZCLWESBZBAMWKQGSVIUNNGDI"). Всего извлечено: 2
Потребитель 4052: сообщение корректно.
```

Спустя некоторое время ввести команду «p» снова. Сверить счетчики с реальными данными.

```
Состояние очереди:
Добавлено сообщений: 31
Извлечено сообщений: 21
Свободные места: 0
Занятые места: 10
Производителей: 1, Потребителей: 1
```

При запуске программы не запускать производителей. Ожидается сработка `deadlock_monitor` каждые 5 секунд.

```
Запуск основного процесса.
Команды: '+' для запуска производителя, '-' для запуска потребителя, 'p' для просмотра состояния, 'q' для выхода.
---Нет активных производителей! Нужно добавить производителей.---
---Нет активных производителей! Нужно добавить производителей.---
---Нет активных производителей! Нужно добавить производителей.---
```

При заполнении очереди и отсутствии потребителей ожидается срабатывание `deadlock_monitor` каждые 5 секунд.

```
Производитель 4065: добавлено сообщение (тип 'B', размер 19, hash 1447). Всего добавлено: 8
Производитель 4066: добавлено сообщение (тип 'B', размер 19, hash 1447). Всего добавлено: 9
Производитель 4064: добавлено сообщение (тип 'W', размер 22, hash 1718). Всего добавлено: 10
---Нет активных потребителей! Нужно добавить потребителей.---
---Нет активных потребителей! Нужно добавить потребителей.---
---Нет активных потребителей! Нужно добавить потребителей.---
```

Ввести команду «q» для выхода из программы. Ожидается, что все процессы получат `SIGTERM` и завершатся корректно.

```
q
Производитель 4066 завершает работу.
Производитель 4065 завершает работу.
Производитель 4064 завершает работу.
Потребитель 4069 завершает работу.
Потребитель 4070 завершает работу.
Производитель 4072 завершает работу.
Потребитель 4073 завершает работу.
Потребитель 4071 завершает работу.
Завершение программы.
```