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
Е.С. Шахлан

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

Минск 2025

1 УСЛОВИЕ 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 мьютекс или одноместный семафор для монопольного доступа к очереди.

Сообщения имеют формат, представленный в таблице 1.1 (размер и смещение в байтах).

Таблица 1.1 — Формат сообщений

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 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.

В качестве семафоров используются семафоры `System V`.

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).

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 разделяемой памяти (`Shared Memory`) и семафоров для синхронизации процессов. Она состоит из одного управляющего процесса (основной процесс, запускаемый пользователем), который может динамически создавать дочерние процессы двух типов.

Производители (`producer`) – создают сообщения и добавляют их в очередь.

Потребители (`consumer`) – извлекают сообщения из очереди и проверяют их целостность.

Основной процесс управляет взаимодействием с пользователем, создаёт и завершает дочерние процессы, а также обеспечивает корректное завершение работы всей системы.

2.2 Алгоритм работы основного процесса

Инициализация:

1) Выделяется разделяемая память под структуру очереди;

2) Инициализируются три семафора:

- `mutex` — мьютекс для синхронизации доступа к очереди;

- `free_space` — количество свободных мест в очереди;

- `items` — количество заполненных мест в очереди.

3) Регистрируются обработчики сигналов `SIGINT` и `SIGTERM`.

Пользователю доступно управление с клавиатуры через интерфейс управления:

- `p` – создать производителя;

- `c` – создать потребителя;

- `d` – завершить одного производителя;

- `r` – завершить одного потребителя;

- `l` – вывести статус очереди;

- `q` – завершение программы.

Создание процессов:

1) При получении команды создается новый процесс с помощью `fork()`;

2) В дочернем процессе вызывается функция `create_prod()` или `create_cons()`;

3) Основной процесс сохраняет `PID` дочернего процесса в массив и продолжает выполнение.

Если в системе нет активных производителей и потребителей, а в очереди остались данные (или извлечены), автоматически создается один производитель и потребитель.

Завершение:

1) Удаляется производитель;

2) Удаляется потребитель;

3) Завершаются все дочерние процессы;

4) Удаляется разделяемая память и семафоры;

5) Освобождаются динамически выделенные ресурсы.

2.3 Алгоритм работы производителя

Процесс производителя выполняет следующие действия в цикле:

1) Ожидает свободное место в очереди с помощью `sem_wait(free_space)`;

2) Выполняет `sem_wait(mutex)`, чтобы убедиться, что есть свободное место;

3) Создает сообщение;

4) Копирует сообщение в конец очереди `(queue->tail)` и обновляет счетчики;

5) Освобождает мьютекс и увеличивает семафор `sem_post(items)`;

6) Выводит информацию о добавленном сообщении;

7) Делает паузу перед созданием следующего сообщения;

8) При получении сигнала завершения завершает свою работу.

2.4 Алгоритм работы потребителя

Процесс потребителя в цикле выполняет:

1) Ожидает доступность сообщения `sem_wait(items);`

2) Захватывает мьютекс очереди `sem_wait(mutex);;`

3) Извлекает сообщение из начала очереди `(queue->head);`

4) Обновляет счетчики и перемещает указатель головы;

5) Освобождает мьютекс и увеличивает семафор `free_space`;

6) Сообщает о свободном месте;

7) Выводит информацию о полученном сообщении;

8) Засыпает на случайный промежуток времени;

9) При получении сигнала завершения – корректно завершает работу.

2.5 Формирование и управление очередью

Структура очереди (`message_queue_t`) располагается в разделяемой памяти и содержит:

Массив сообщений (`messages[MAX_MSGS]`);

Индексы `head`, `tail`, счетчик `count`.

Инициализация производится функцией `init_queue()`.

Семафоры управляют синхронизацией доступа к очереди и создаются через `sem_init()`:

- `mutex` – двоичный семафор (значение 1);

- `empty` – начально равен размеру очереди;

- `full` – начально равен 0.

3 ФУНКЦИОНАЛЬНАЯ СТРУКТУРА ПРОЕКТА

Основной процесс (управляющий):

Функция `main` – является точкой входа в программу. Выполняет инициализацию `IPC`-ресурсов (разделяемой памяти и семафоров), предоставляет интерактивный интерфейс управления дочерними процессами, а также обрабатывает завершение программы.

Передаваемые параметры:

- `int argc` – количество аргументов командной строки;

- `char* argv[]` – массив аргументов командной строки.

Основные действия:

1) Создание и инициализация разделяемой памяти и семафоров;

2) Обработка пользовательского ввода с клавиатуры:

- `p` – запуск процесса производителя;

- `c` – запуск процесса потребителя;

- `d` – завершение одного производителя;

- `r` – завершение одного потребителя;

- `l` – вывод текущего состояния очереди;

- `q` – завершение всех процессов и очистка ресурсов.

3) Управление списками дочерних процессов;

4) Обработка сигналов `SIGINT` и `SIGTERM`.

Производитель (дочерний процесс):

Функция `create_prod()` реализует цикл, в котором создаются и помещаются в очередь сообщения. Синхронизация обеспечивается с помощью семафоров.

Основные действия:

1) Проверка максимального количества производителей;

2) Создание нового процесса с помощью `fork()`;

3) Создание сообщения для отправки;

4) Захват мьютекса (`mutex`), помещение сообщения в очередь;

5) Освобождение мьютекса, инкремент `full`;

6) Печать информации о действии;

7) Задержка перед следующей итерацией;

7) Обработка сигнала завершения.

Потребитель (дочерний процесс):

Функция `create_cons()` извлекает сообщения из очереди и проверяет их корректность по хешу.

Основные действия:

1) Ожидание наличия сообщений (`full`);

2) Захват мьютекса (`mutex`), извлечение сообщения;

3) Освобождение мьютекса, инкремент `empty`;

4) Проверка целостности сообщения (сравнение хешей);

5) Вывод результатов;

6) Задержка перед следующей итерацией;

7) Обработка сигнала завершения.

Функция `init_queue()` инициализирует структуру очереди в разделяемой памяти: обнуляет поля, устанавливает индексы и счетчики.

Функция `del_prod()` освобождает все ресурсы производителя.

Функция `del_cons()` освобождает все ресурсы потребителя;

Функция `hash(mes* msg)` создает контрольную сумму на основе содержимого `msg`.

Передаваемые параметры:

- `mes* msg` – сообщение.

Функция `prod_mes(mes* msg)` генерирует случайное сообщение.

Передаваемые параметры:

- `mes* msg` – сообщение.

Функция `consume_mes(mes* msg)` проверяет целостность сообщения.

Передаваемые параметры:

- `mes* msg` – сообщение.

Функция `put_msg(mes* msg)` добавляет сообщение в очередь.

Передаваемые параметры:

- `mes* msg` – сообщение.

Функция `get_msg(mes* msg)` читает сообщение из очереди .

Передаваемые параметры:

- `mes* msg` – сообщение.

Функция `check_sems()` создаёт и настраивает общую память и семафоры.

4 ПОРЯДОК СБОРКИ И ЗАПУСКА

1) Перейти в каталог проекта:

```
$ cd 'Шахлан Е.С./lab04'
```

2) Собрать проект с помощью `make`:

```
$ make
```

3) Запустить программу:

```
$ ./main
```

5 РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

```
p - create producer
c - create consumer
d - delete producer
r - delete consumer
l - show processes
q - quit program
p
Pid = 12242
Message = 8591B4C5
Messages injected counter = 1
Pid = 12242
Message = 63C33205
Messages injected counter = 2
Pid = 12242
Message = 83F69C47
Messages injected counter = 3
Queue is full, prod with pid = 12242 is waiting
s
c
Pid = 12242
Message = 8C53FEAB
Messages injected counter = 4
Check sum (= 1260593675) not equal msg_hash (= 0)
Pid = 12282
Message = 0
Messages extracted counter = 1
Check sum (= 1260593675) not equal msg_hash (= 0)
Queue is full, prod with pid = 12242 is waiting
Pid = 12282
Message = 0
Messages extracted counter = 2
Pid = 12242
Message = F7A96E0C
Messages injected counter = 5
```

```
Queue is full, prod with pid = 12242 is waiting
Check sum (= 1260593675) not equal msg_hash (= 0)
Pid = 12282
Message = 0
Messages extracted counter = 3
Pid = 12242
Message = 65C135E3
Messages injected counter = 6
Check sum (= 1260593675) not equal msg_hash (= 0)
Queue is full, prod with pid = 12242 is waiting
Pid = 12282
Message = 0
Messages extracted counter = 4
Pid = 12242
Message = 8C53FEAB
Messages injected counter = 7
Check sum (= 1260593675) not equal msg_hash (= 0)
Queue is full, prod with pid = 12242 is waiting
Pid = 12282
Message = 0
Messages extracted counter = 5
Pid = 12242
Message = 5D9FAFAF
Messages injected counter = 8
qCheck sum (= 1260593675) not equal msg_hash (= 0)
Queue is full, prod with pid = 12242 is waiting
Pid = 12282
Message = 0
Messages extracted counter = 6
Pid = 12242
Message = 33CCC771
Messages injected counter = 9
```