Министерство образования Республики Беларусь
Учреждение Образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра электронных вычислительных машин

Лабораторная работа № 4
«Задача производители - потребители для процессов»

Проверил: Поденок Л.П.
Выполнил: ст. гр. 350501
Маслаков Н.А.

Минск 2025

1. УСЛОВИЕ ЛАБАРАТОРНОЙ 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

Родительский процесс управляет созданием, удалением и взаимодействием с дочерними процессами. Он обрабатывает команды пользователя, такие как создание нового потребителя или производителя, их удаление. Также он отслеживает состояние дочерних процессов и их вывод.

Производитель создает сообщение, заполняя все поля определенным образом. После чего записывает сообщение в очередь, если это позволяют семафоры. Выводит данные о сообщении и ждет пару секунд для удобного отслеживания действий.

Производитель удаляет сообщение из очереди, если это позволяют семафоры. Выводит данные о сообщении и ждет пару секунд для удобного отслеживания действий.

3. ФУНКЦИОНАЛЬНАЯ СТРУКТУРА ПРОЕКТА

Родительский процесс.

`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_procs()`.

Функция `remove_all_procs` используется для удаления всех дочерних процессов.

`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.

Принимаемые параметры:

1) `Queue* q`. Сама очередь.

`void queue_destroy(Queue* q).`

Функция `queue_destroy` используется для корректного удаления очереди.

Принимаемые параметры:

1) `Queue* q`. Сама очередь.

`uint16_t calculate_hash(Message* msg).`

Функция `calculate_hash` используется для корректного удаления очереди.

Принимаемые параметры:

1) `Message* msg`. Сообщение, для которого вычисляется хэш.

4. ПОРЯДОК СБОРКИ И ЗАПУСКА

1) Перейти в каталог проекта.

```
$ cd 'Маслаков Н.А./lab04'
```

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

```
$ make
```

3) Запустить программу.

```
$ ./build/debug/app
```

5. РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

```
~/lab04$ ./build/debug/app
Controls:
 p - Add producer
 c - Add consumer
 P - Remove last producer
 C - Remove last consumer
 k - Remove all processes
 s - Show status
 q - Quit
p
Producer 11446 created
[Producer 11446] Added. |type: 173, hash: 28951, size: 230|
Total: 1
c
Consumer 11448 created
[Consumer 11448] Removed. |type: 0, hash: 0, size: 0| Total: 1
p
[Producer 11446] Added. |type: 182, hash: 29773, size: 229|
Total: 2

Producer 11449 created
[Producer 11449] Added. |type: 63, hash: 13242, size: 114|
Total: 3
c
Consumer 11450 created
[Consumer 11450] Removed. |type: 0, hash: 0, size: 0| Total: 2
[Producer 11446] Added. |type: 190, hash: 32760, size: 248|
Total: 4
[Producer 11449] Added. |type: 90, hash: 6467, size: 52| Total: 5
[Consumer 11448] Removed. |type: 0, hash: 0, size: 0| Total: 3
[Producer 11446] Added. |type: 145, hash: 28600, size: 218|
Total: 6
[Producer 11449] Added. |type: 10, hash: 20223, size: 156|
Total: 7
[Consumer 11450] Removed. |type: 0, hash: 0, size: 0| Total: 4
[Producer 11446] Added. |type: 76, hash: 21345, size: 159|
Total: 8
[Consumer 11448] Removed. |type: 0, hash: 0, size: 0| Total: 5
[Producer 11449] Added. |type: 91, hash: 24210, size: 187|
Total: 9
k
Producer 11449 removed
```

```
Producer 11446 removed
Consumer 11450 removed
Consumer 11448 removed
s

--- Status ---
Queue: 4/0 (used/free)
Messages: added=9, removed=5
Active producers: 0
Active consumers: 0

c
Consumer 11458 created
[Consumer 11458] Removed. |type: 0, hash: 0, size: 0| Total: 6
[Consumer 11458] Removed. |type: 0, hash: 0, size: 0| Total: 7
[Consumer 11458] Removed. |type: 0, hash: 0, size: 0| Total: 8
C
Consumer 11458 removed
s

--- Status ---
Queue: 1/3 (used/free)
Messages: added=9, removed=8
Active producers: 0
Active consumers: 0

q

Program terminated
```