Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра ЭВМ

Дисциплина: Операционные системы и системное программирование

ОТЧЁТ
к лабораторной работе №4
на тему
Задача производители-потребители для процессов.

Выполнил студент гр.230501 Лазовский И.А.

Проверил старший преподаватель кафедры ЭВМ
Поденок Л.П.

Минск 2024

1 УСЛОВИЕ ЛАБОРАТОРНОЙ РАБОТЫ.

Задание

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 и счетчик извлеченных.

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 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.

При формировании сообщения контрольные данные формируются из всех байт сообщения. 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.

message: Эта структура представляет собой сообщение. Она содержит следующие поля:

type: Тип сообщения (представленный в виде uint8_t).

hash: Хэш сообщения (представленный в виде uint16_t).

size: Размер сообщения (представленный в виде uint8_t).

data: Указатель на данные сообщения (представленные в виде строки char*).

queue: Эта структура представляет собой очередь сообщений. Она включает следующие поля:

head: Указатель на начало очереди.

h: Индекс начала очереди (представленный в виде int).

tail: Указатель на конец очереди.

t: Индекс конца очереди (представленный в виде int).

buff: Буфер для хранения сообщений (массив структур message).

count_added: Количество добавленных сообщений (представленное в виде int).

count_extracted: Количество извлеченных сообщений (представленное в виде int).

Функции:

getSize(): Генерирует случайный размер сообщения (от 1 до 256 байт).

getType(): Определяет тип сообщения (0 или 1) на основе его размера.

getData(): Генерирует случайные данные для сообщения (строку из случайных букв).

FNV1_HASH(): Вычисляет хэш сообщения с использованием алгоритма FNV-1.

createMessage(): Создает новое сообщение с случайными данными (размер, тип, данные и хэш).

start(): Инициализирует разделяемую память и семафоры.

deleteConsumers(), deleteProducers(): Удаляют потребителей и производителей.

fromProgExit(): Очищает ресурсы при завершении программы.

viewStatus(): Выводит информацию о текущем состоянии очереди.

addMessage(), extractedMessage(): Добавляют и извлекают сообщения из очереди.
addConsumer(), addProducer(): Создают потребителей и производителей.
menu(): Выводит меню с опциями.
viewProcesses(): Выводит информацию о запущенных процессах.

3. ФУНКЦИОНАЛЬНАЯ СТРУКТУРА ПРОЕКТА.

Проект собирается с помощью makefile. Пример запуска:

```
ilua@fedora:~/Рабочий стол/labFor$ build/debug/main
```

Для запуска проекта нам требуется в терминале запустить программу main. Где программа сразу переходит в цикл обработки символов

В проекте имеется каталог для сборки debug и release. Каталог git для системы контроля версий моего проекта. Директория src с исходным кодом. И makefile для компиляции и сборки проекта.

4. ПОРЯДОК СБОРКИ И ИСПОЛЬЗОВАНИЯ.

Для компиляции и сборки проекта используется makefile.

Порядок сборки:

4CC: Это неявная переменная, которая указывает на компилятор. В данном случае используется gcc.

CFLAGS_DEBUG: Флаги компиляции для режима отладки.

CFLAGS_RELEASE: Флаги компиляции для релизного режима.

DEBUG: Путь к директории с отладочными файлами.

RELEASE: Путь к директории с релизными файлами.

OUT_DIR: Исходно установлен на $(DEBUG), но может изменяться на $(RELEASE) в зависимости от режима сборки.

all: Цель, которая собирает исполняемый файл main.

$(prog): Это правило для сборки исполняемого файла main. Оно зависит от объектных файлов ($(objects)).

$(OUT_DIR)/%.o: Это правило для сборки объектных файлов из исходных файлов .c.

ifeq ($(MODE), release): Если переменная MODE установлена в release, то используются флаги для релизного режима.

`.PHONY: clean`: Это объявление говорит make, что `clean` - это фиктивная цель (не связанная с файлами).

`clean`: Удаляет все файлы в директориях `$(DEBUG)` и `$(RELEASE)`.

Порядок использования.

1. Компиляция:

Для отладочной сборки: `make` или `make MODE=debug`

Для релизной сборки: `make MODE=release`

2. Очистка:

`make clean` - удаляет все объектные файлы и исполняемый файл.

3. Запуск:

После успешной компиляции запустите исполняемый файл, например: ./build/debug/main.

5. МЕТОД ТЕСТИРОВАНИЯ И РЕЗУЛЬТАТ ТЕСТИРОВАНИЯ.

```
ilua@fedora:~/Рабочий стол/labs/project/labFor#
build/debug/main
Write 'm' to display menu.
1
producer00 producer message: HASH=781A, counter_added=1
1
producer01 producer message: HASH=7319, counter_added=2
3
consumer_00 consumer message: HASH=781A, counter_extracted=1

producer00 producer message: HASH=820B, counter_added=3
s
----------------------------
Queue max size:15
Current size:2
Added:3
Extracted:1
Consumers:1
Producers:2
----------------------------
producer01 producer message: HASH=EA05, counter_added=4
pconsumer_00 consumer message: HASH=7319,
counter_extracted=2

consumer_00 , 20272
producer00 , 20270
producer01 , 20271
producer00 producer message: HASH=17AE, counter_added=5
e
```

```
Was delete consumer with name:consumer_00 , pid:20272
Was delete producer with name:producer01 , pid:20271
Was delete producer with name:producer00 , pid:20270
```