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 №4 на тему
“Задача производители-потребители для процессов”

Выполнил:
студ. гр. 350501 Везенков М. Ю.

Проверил: ст.пр. Поденок Л. П.

Минск 2025

СОДЕРЖАНИЕ

1 ЦЕЛЬ 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 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 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 ВЫПОЛНЕНИЕ ЛАБОРАТОРНОЙ РАБОТЫ

2.1 Описание программы

Данная программа представляет собой реализацию классической задачи "Производитель-Потребитель", демонстрирующую взаимодействие процессов посредством межпроцессного взаимодействия (IPC) в UNIX-подобных операционных системах. Архитектурно программа состоит из основного родительского процесса, который управляет жизненным циклом одного или нескольких процессов-производителей и одного или нескольких процессов-потребителей.

Производители генерируют сообщения и помещают их в разделяемый буфер (очередь), а потребители извлекают сообщения из этого буфера и обрабатывают их. Синхронизация доступа к разделяемой очереди между процессами осуществляется с использованием системных семафоров System V. Программа использует разделяемую память System V для размещения самой очереди сообщений. Взаимодействие с пользователем происходит через стандартный ввод, настроенный в неблокирующий и необрабатывающий режим для мгновенного реагирования на команды без ожидания символа новой строки. Родительский процесс перехватывает сигналы завершения (SIGINT, SIGTERM) для корректного завершения работы всех дочерних процессов и освобождения выделенных IPC ресурсов.

Основное взаимодействие между процессами происходит через кольцевую очередь фиксированной емкости, расположенную в сегменте разделяемой памяти. Производители и потребители получают доступ к этой очереди по указателю, полученному после присоединения сегмента разделяемой памяти. Сообщения в очереди имеют фиксированную структуру, включающую тип, размер данных, сами данные и хеш для проверки целостности.

Синхронизация доступа к очереди и управление свободными/занятыми слотами реализованы с помощью набора из трех семафоров System V: мьютекс для исключительного доступа к структуре очереди (индексы головы/хвоста, счетчики), семафор "empty" для отслеживания свободных слотов (ожидание производителя, если очередь полна), и семафор "full" для отслеживания занятых слотов (ожидание потребителя, если очередь пуста). Флаг завершения в родительском процессе используется для координации graceful shutdown.

2.2 Описание основных функций

Родительский процесс, реализованный в файле main.c, содержит главную функцию main(), которая инициализирует генератор

псевдослучайных чисел, настраивает стандартный ввод в неблокирующий режим без эха с помощью функций из utils.c, инициализирует механизмы IPC (разделяемую память и семафоры) через вызов initialize_ipc() из ipc_manager.c, регистрирует обработчики сигналов SIGINT и SIGTERM для установки флага завершения g_terminate_flag, а также устанавливает функцию cleanup_resources() через atexit() для гарантированного освобождения ресурсов при выходе.

Основной цикл программы ожидает ввода команд с клавиатуры в неблокирующем режиме, используя функцию kbhit(). В зависимости от введенного символа, вызываются соответствующие функции из ipc_manager.c: create_new_producer() или create_new_consumer() для создания новых дочерних процессов с использованием fork() и последующим вызовом producer_run() или consumer_run() в дочернем процессе соответственно, stop_last_producer() или stop_last_consumer() для завершения последнего запущенного процесса соответствующего типа путем отправки сигнала SIGTERM и ожидания его завершения с помощью waitpid(), display_status() для вывода текущего состояния очереди и количества активных процессов, или установка флага g_terminate_flag при вводе 'q'. Родительский процесс хранит PIDы запущенных производителей и потребителей в статических массивах.

Модуль ipc_manager.c содержит ключевые функции для управления IPC и дочерними процессами. Функция initialize_ipc() создает сегмент разделяемой памяти с помощью shmget() и присоединяет его к адресному пространству процесса с помощью shmat(), затем создает набор семафоров с помощью semget() и инициализирует их значения: мьютекс в 1 (свободен), "empty" в емкость очереди (все слоты свободны), "full" в 0 (нет занятых слотов).

Функция cleanup_resources(), вызываемая при завершении родительского процесса, отправляет сигнал SIGTERM всем запущенным дочерним процессам, ожидает их завершения, отсоединяет разделяемую память с помощью shmdt() и удаляет IPC объекты (разделяемую память и семафоры) из системы с помощью shmctl(IPC_RMID) и semctl(IPC_RMID). Функция semaphore_op() выполняет атомарную операцию над указанным семафором с помощью semop().

Функции create_new_producer/consumer() добавляют PIDы созданных процессов в соответствующие массивы, а remove_pid_from_list() удаляет PIDы завершившихся процессов. display_status() безопасно, под защитой мьютекса, считывает и выводит статистику очереди и счетчики процессов.
Процесс-производитель, реализованный в producer.c, содержит функцию producer_run(), которая является точкой входа для каждого экземпляра производителя. Эта функция регистрирует обработчик сигнала SIGTERM для установки локального флага s_child_terminate_flag. В основном цикле производитель генерирует случайное сообщение, включая его тип, размер и

данные, и вычисляет для него хеш. Затем он пытается добавить сообщение в разделяемую очередь, выполняя последовательность операций над семафорами: ожидание свободного слота (операция -1 над SEM_EMPTY_IDX), ожидание доступа к мьютексу (операция -1 над SEM_MUTEX_IDX). После успешного получения семафоров, производитель в критической секции добавляет сообщение в очередь, обновляет индексы и счетчики в разделяемой памяти, затем освобождает мьютекс (операция +1 над SEM_MUTEX_IDX) и сигнализирует о появлении нового заполненного слота (операция +1 над SEM_FULL_IDX). В цикле также предусмотрена задержка с использованием nanosleep() и обработка прерывания системных вызовов сигналами (EINTR).

Процесс-потребитель, реализованный в consumer.c, содержит функцию consumer_run(), аналогичную функции производителя по структуре. Он также регистрирует обработчик SIGTERM. В своем основном цикле потребитель ожидает появления сообщения в очереди, выполняя операцию -1 над SEM_FULL_IDX. Затем он ожидает доступа к мьютексу (операция -1 над SEM_MUTEX_IDX). Получив доступ, потребитель в критической секции считывает сообщение из очереди, обновляет индексы и счетчики в разделяемой памяти (увеличивает free_slots и extracted_count), после чего освобождает мьютекс (операция +1 над SEM_MUTEX_IDX). Затем он сигнализирует об освобождении слота (операция +1 над SEM_EMPTY_IDX).

После извлечения сообщения потребитель проверяет его целостность, пересчитывая хеш и сравнивая его с оригинальным, и выводит результат обработки и общую статистику. Как и производитель, потребитель включает задержку с использованием nanosleep() и обрабатывает EINTR. В обеих дочерних процессах флаги завершения и счетчики, модифицируемые в обработчиках сигналов, объявлены как volatile sig_atomic_t для обеспечения безопасности при работе с сигналами.

2.3 Описание порядка сборки и использования

Сборка проекта осуществляется с помощью стандартной утилиты make и предоставленного файла Makefile. Makefile поддерживает два основных режима сборки: отладочный (debug) и релизный (release). По умолчанию при вызове просто команды make или make debug-build активируется отладочный режим, который включает флаги компилятора -g3 и -ggdb для добавления подробной отладочной информации.

Релизный режим активируется командой make MODE=release и включает флаг оптимизации -O2, а также обрабатывает предупреждения как ошибки с флагом -Werror. Результаты компиляции, включая объектные и исполняемые файлы, помещаются в подкаталоги build/debug или build/release соответственно.

Для сборки проекта в отладочном режиме достаточно выполнить

команду make или make debug-build в корневой директории проекта.

Для сборки в релизном режиме используется команда make release-build. Команда make clean удаляет скомпилированные файлы и каталоги сборки build/. После успешной сборки, исполняемый файл родительского процесса (parent) будет расположен в соответствующем подкаталоге build/debug/ или build/release/. Для запуска программы необходимо выполнить соответствующий исполняемый файл parent. Например, для запуска отладочной версии используется команда make run, которая автоматически выполняет сборку в отладочном режиме и запускает файл build/debug/prod_cons_ipc.

Для запуска релизной версии используется make run-release. Программа работает в интерактивном режиме, ожидая ввода команд с клавиатуры. Команды включают: 'p' для создания нового производителя, 'c' для создания нового потребителя, 'P' для остановки последнего производителя, 'C' для остановки последнего потребителя, 's' для отображения текущего состояния очереди и количества процессов, и 'q' для завершения программы. Программа завершает работу при вводе 'q' или получении сигналов SIGINT или SIGTERM. Корректное завершение гарантируется за счет механизма atexit() и обработки сигналов.

3 РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ

```
lab04 git:(main) make run
Debug build complete in build/debug
Running DEBUG version build/debug/prod_cons_ipc...
build/debug/prod_cons_ipc
[Main] Initializing system...
[IPC Init] Shared memory and semaphores initialized successfully.
--- Producer/Consumer Control ---
  p: Add Producer   c: Add Consumer
  P: Remove Producer C: Remove Consumer
  s: Show Status    q: Quit
--------------------------------
Enter command (p,c,P,C,s,q):
[Parent] Created Producer 1 (PID: 7488)
Enter command (p,c,P,C,s,q): [Producer 1] Started.
[Producer 1] Added msg (Type:187 Size:143 Hash:8939). Total Added: 1
[Producer 1] Added msg (Type:74 Size:0 Hash:2442). Total Added: 2
[Parent] Created Consumer 1 (PID: 7489)
Enter command (p,c,P,C,s,q): [Consumer 1] Started.
[Consumer 1] Extracted msg (Type:187 Size:143 Hash:8939 -> OK). Total
Extracted: 1
[Producer 1] Added msg (Type:154 Size:227 Hash:50114). Total Added: 3
--- System Status ---
Queue Capacity: 10
Queue Occupied: 2
Queue Free:     8
Total Added:    3
Total Extracted:1
Producers:      1 / 10
Consumers:      1 / 10
--------------------
Enter command (p,c,P,C,s,q): [Consumer 1] Extracted msg (Type:74 Size:0
Hash:2442 -> OK). Total Extracted: 2
[Producer 1] Added msg (Type:241 Size:137 Hash:30020). Total Added: 4
[Main] Quit command received. Initiating shutdown...
[Main] Exiting.
[Cleanup] Starting resource cleanup...
[Cleanup] Sending SIGTERM to remaining children...
[Cleanup] Signaling semaphores to unblock children...
[Cleanup] Waiting for children to exit...
[Producer] SIGTERM received
[Producer 1] Terminating gracefully...
[Consumer] SIGTERM received
[Consumer 1] Terminating gracefully...
[Cleanup] Finished waiting for children.
[Cleanup] Resource cleanup complete.
```

Рисунок 3.1 – пример создания и убийства производителя и потребителя

4 ВЫВОД

В ходе выполнения данной лабораторной работы были глубоко изучены фундаментальные концепции и механизмы взаимодействия процессов в среде UNIX-подобных операционных систем. Разработанная программа эффективно демонстрирует модель Производитель-Потребитель, центральным элементом которой является родительский процесс, ответственный за динамическое управление (создание и завершение) дочерних процессов, выступающих в роли производителей и потребителей.

Взаимодействие между этими процессами реализуется через общий буфер (очередь), расположенный в сегменте разделяемой памяти System V, с использованием семафоров System V для обеспечения необходимой синхронизации и защиты доступа к разделяемым ресурсам.

Программа также позволила закрепить навыки работы с сигналами для обеспечения graceful shutdown всех запущенных компонентов системы и безопасной обработки асинхронных событий, а также методы настройки стандартного ввода терминала для реализации интерактивного управления в режиме реального времени без буферизации.