ЛАБОРАТОРНАЯ РАБОТА №2

Тема: Процессы и потоки в ОС Linux (Ubuntu): сравнение, механизмы синхронизации. Парадигмы межпроцессорного взаимодействия.

Цель работы: Изучить различия между процессами и потоками в ОС Linux (Ubuntu), а также освоить механизмы синхронизации и межпроцессорного взаимодействия для обеспечения корректной работы программ в многозадачной среде.

Цель работы обуславливает постановку и решение следующих **задач**:

1) Изучить основные концепции, относящиеся к процессам и потокам в ОС Linux (Ubuntu).
2) Получить основное понятие о создании потоков в Linux с использованием библиотеки POSIX Threads (pthread.h).
3) Изучить основные механизмы синхронизации: мьютексы, семафоры, спинлоки, барьеры, блокировки читателей/писателей, futex, каналы.
4) Изучить механизмы межпроцессорного взаимодействия в ОС Linux (Ubuntu).
5) Выполнить индивидуальное задание, связанное с применением механизмов синхронизации и межпроцессорного взаимодействия в многозадачной среде.

Ход выполнения лабораторной работы

1. Выполнить индивидуальное задание, закрепляющее на практике полученные знания (номер задания соответствует номеру студента по журналу; если этот номер больше, чем максимальное число заданий, тогда вариант задания вычисляется по формуле: номер по журналу % максимальный номер задания, где % — остаток от деления).

1.1. Решение задачи, соответствующей условиям варианта, представить двумя программными реализациями: а) с использованием потоков, посредством обращения к ресурсам библиотеки POSIX Threads (pthread.h); б) с использованием процессов, посредством системных вызовов. Следует использовать указанные в задании механизмы синхронизации и межпроцессорного взаимодействия. Необходимо оптимизировать программы. Например, целесообразно позволить потокам "спать" до тех пор, пока не выполнится нужное условие, избегая активного ожидания, которое может сильно нагрузить процессор. Обязательно следует сравнить программы между собой посредством метрик для оценки: утилизации ресурсов CPU, расхода памяти, производительности (то есть необходимо оценить выполнение производственных циклов задачи в единицу времени; например, это может быть количество съеденных кондитерских изделий за 1 мин. и т.п.). Произвести эксперименты на виртуальной машине с 2, 3, 4, 6 ядрами.

2. Подготовить отчёт по работе, который должен включать: краткое описание всех использованных механизмов синхронизации и межпроцессорного взаимодействия; программы, написанные в ходе выполнения лабораторной работы; протоколы тестирования программ; выводы. Исходные коды программ должны быть прикреплены к отчету в виде отдельных файлов.

Общая для всех вариантов задача

Змей Горыныч имеет три головы, каждая из которых независимо от других голов ест продукцию с двух кондитерских фабрик. Каждая фабрика производит разные типы кондитерских изделий

(торты, пирожные, конфеты, пряники) с различной скоростью. Головам нужно получать продукцию из общего склада, но склад ограничен по объему. Задача — организовать взаимодействие между фабриками (производителями) и головами Змея Горыныча (потребителями) так, чтобы они корректно синхронизировались при производстве и потреблении продукции, избегая конфликтов, минимизируя ситуации простоя и переполнения склада.

Каждая фабрика производит разные типы продукции с различной скоростью:

Фабрика 1 (производит торты и пирожные):

Выпуск тортов: 1 торт каждые 70 мс.
Выпуск пирожных: 2 пирожных каждые 50 мс.

Фабрика 2 (производит конфеты и пряники):

Выпуск конфет: 3 конфеты каждые 40 мс.
Выпуск пряников: 1 пряник каждые 60 мс.

Вместимость склада: 10 единиц продукции.

Потребление продукции:

Каждая из трех голов Змея Горыныча потребляет продукцию с разной скоростью:

Голова 1:

Потребляет торты и конфеты.
Потребляет 1 продукт каждые 80 мс.

Голова 2:

Потребляет пирожные и пряники.
Потребляет 1 продукт каждые 90 мс.

Голова 3:

Потребляет любые виды продукции (универсальная голова).
Потребляет 1 продукт каждые 70 мс.

Вариантные уточнения к задаче

1. Для реализации склада использовать, в случае процессов, разделяемую память (sys/shm.h), а для реализации потоков — обычный массив. Для синхронизации, в случае процессов, использовать семафоры (sys/sem.h), а для потоков — мьютексы (pthread_mutex_t) и условные переменные (pthread_cond_t), если необходимо.

2. Для реализации склада использовать, в случае процессов, именованный канал FIFO (mkfifo), для реализации потоков — очередь с блокировкой. Для синхронизации, в случае процессов, использовать семафоры (sys/sem.h), а для потоков — futex (linux/futex.h, sys/syscall.h).

3. Вводится дополнительное условие в логику задачи. После заполнения склада производство кондитерской продукции останавливается и возобновляется только тогда, когда головы Змея Горыныча полностью съедят продукцию и склад очистится. Для реализации склада использовать, в случае процессов, разделяемую память (sys/shm.h), а для реализации потоков — связанный список. Для синхронизации, в случае процессов, использовать семафоры (sys/sem.h), а для потоков — мьютексы (pthread_mutex_t), барьерные синхронизации (pthread_barrier_t) и условные переменные (pthread_cond_t), если необходимо.

4. Для реализации склада использовать, в случае процессов, разделяемую память (sys/shm.h), а для потоков — обычный массив. Для синхронизации, в случае процессов, использовать атомарные операции (например, "сравнение и замена" — Compare-and-Swap, CAS: __sync_bool_compare_and_swap) и спинлоки для доступа к разделяемой памяти, а для потоков — спинлоки (pthread_spinlock_t).

5. В реализации использовать для процессов и потоков файлы (sys/file.h) и механизм блокировки файлов (flock).

6. Вводится дополнительное условие в логику задачи. Змей Горыныч имеет три головы, каждая из которых может одновременно проверять содержимое склада, чтобы узнать, какие продукты доступны, но при этом только одна голова может забрать продукцию за один раз. Для реализации склада использовать, в случае процессов, очереди сообщений (Message Queues) (sys/msg.h), а для потоков — очередь и блокировки читателей/писателей (pthread_rwlock_t).

7. Для реализации склада использовать, в случае процессов, локальные сокеты (sys/socket.h) для передачи данных между процессами, а для потоков — асинхронные очереди с использованием сигналов (signal.h). Можно использовать мьютексы (pthread_mutex_t).

8. Вводится дополнительное условие в логику задачи. Каждая голова Змея Горыныча должна уметь проверять содержимое склада несколько раз подряд, прежде чем выбрать продукт. Для корректной работы программы, проверка и забор продукции должны быть защищены, в случае потоков, рекурсивными мьютексами (pthread_mutexattr_settype), а в случае процессов — семафорами (sys/sem.h). Голова может вызвать функцию проверки доступности продукта несколько раз (например, чтобы проверить разные типы продукции). Важно, чтобы каждая голова корректно освобождала мьютекс после завершения работы.

9. Для реализации склада, в случае процессов, использовать механизм Memory-Mapped Files (mmap), который позволяет отобразить файл или область памяти в адресное пространство нескольких процессов. Все процессы (головы Змея Горыныча и фабрики) должны отобразить файл в свое адресное пространство с помощью mmap, чтобы работать с ним как с общим складом. Для реализации потоков — обычный массив. Для синхронизации, в случае процессов, использовать семафоры (sys/sem.h), а для потоков — мьютексы (pthread_mutex_t) и условные переменные (pthread_cond_t), если необходимо.

10. Для организации синхронизации между процессами и между потоками использовать механизм эвентов, eventfd (sys/eventfd.h) для сигнализации о наличии или отсутствии продукции

на складе. Это должно позволить фабрикам уведомлять головы Змея Горыныча о том, что продукция добавлена на склад, а головы могут ждать сигнал от эвента, прежде чем потреблять продукцию. Здесь можно использовать очередь или буфер, в котором будут храниться элементы, представляющие конкретные виды продукции.

11. В случае потоков следует использовать pthread_cond_broadcast(), который позволит всем потребителям (головам Змея Горыныча) одновременно получать уведомление о поступлении новой продукции на склад. Как только фабрика добавляет продукт, она вызывает широковещательный сигнал, разблокирующий все ожидающие потоки. Это позволяет нескольким головам одновременно проверить склад и забрать продукцию, что увеличивает эффективность программы в многозадачной среде. Для процессов, чтобы посылать уведомление группе процессов, использовать сигналы (signal.h). Допустимо, в случае процессов, использовать семафоры (sys/sem.h), а для потоков — мьютексы (pthread_mutex_t) и условные переменные (pthread_cond_t), если необходимо.

12. Для организации синхронизации между процессами следует использовать pipe (анонимные каналы). Необходимо реализовать механизм передачи данных между фабриками и головами Змея Горыныча через pipe, обеспечивая корректную синхронизацию потребления и производства продукции. Каждая фабрика должна писать данные в pipe, а головы — читать. Для синхронизации доступа к pipe можно также использовать семафоры (sys/sem.h) для управления очередностью доступа процессов к общему ресурсу. Создайте два анонимных канала: один для передачи продукции с фабрик, другой — для сигналов от голов (например, сигнала о том, что продукт забран, и производство можно продолжить). Для организации синхронизации между потоками используйте блокирующую очередь и мьютексы (pthread_mutex_t) с условными переменными (pthread_cond_t). Каждая фабрика должна добавлять продукцию в очередь, а головы Змея Горыныча будут извлекать её из этой очереди. Мьютексы будут использоваться для защиты доступа к общей очереди, а условные переменные — для организации ожидания, когда очередь пуста или заполнена.

Контрольные вопросы

1. Чем процесс отличается от потока в операционной системе Linux?
2. Какие системные вызовы используются для создания и управления процессами в Linux?
3. Как можно создать поток в Linux с использованием POSIX Threads?
4. Что такое мьютексы, и как они помогают при синхронизации потоков?
5. В чем отличие семафоров от мьютексов? Когда лучше использовать семафоры?
6. Что такое спинлоки и когда их применение более эффективно, чем использование мьютексов?
7. Как работают барьеры для синхронизации потоков и процессов?
8. Что такое futex, и какие преимущества он имеет перед традиционными механизмами синхронизации?
9. Как использовать сокеты для межпроцессного взаимодействия в Linux?
10. Какие особенности применения сигналов для асинхронной обработки в потоках?
11. В каких случаях стоит использовать блокировки читателей/писателей (pthread_rwlock_t)?
12. Какие преимущества и недостатки использования разделяемой памяти и FIFO для межпроцессного взаимодействия?
13. Как использовать именованные и анонимные каналы (pipe) для синхронизации процессов?
14. Что такое рекурсивные мьютексы, и когда их использование оправдано в многопоточных программах?

15. В чем заключается роль Memory-Mapped Files (mmap) в организации совместного доступа к памяти между процессами?
16. Как использовать eventfd для синхронизации между процессами и потоками?
17. Как работают условные переменные (pthread_cond_t) и когда целесообразно использовать широковещательные сигналы (pthread_cond_broadcast) для потоков?
18. Какие принципиальные отличия существуют между использованием атомарных операций, таких как Compare-and-Swap (CAS) и спинлоков для синхронизации процессов?
19. Как использовать сигналы для передачи уведомлений между процессами?