Курсовой проект по курсу «Операционные системы»

Цель курсового проекта

1. Приобретение практических навыков в использовании знаний, полученных в течении курса
2. Проведение исследования в выбранной предметной области

Задание

Необходимо спроектировать и реализовать программный прототип в соответствии с выбранным вариантом. Произвести анализ и сделать вывод на основании данных, полученных при работе программного прототипа.

Возможные варианты курсового проекта:

Аллокаторы памяти

Исследование 2 аллокаторов памяти: необходимо реализовать два алгоритма аллокации памяти и сравнить их по следующим характеристикам:

- Фактор использования
- Скорость выделения блоков
- Скорость освобождения блоков
- Простота использования аллокатора

Каждый аллокатор памяти должен иметь функции аналогичные стандартным функциям free и malloc (realloc, опционально). Перед работой каждый аллокатор инициализируется свободными страницами памяти, выделенными стандартными средствами ядра. Необходимо самостоятельно разработать стратегию тестирования для определения ключевых характеристик аллокаторов памяти. При тестировании нужно свести к минимуму потери точности из-за накладных расходов при измерении ключевых характеристик, описанных выше.
В отчете необходимо отобразить следующее:

- Подробное описание каждого из исследуемых алгоритмов
- Процесс тестирования
- Обоснование подхода тестирования
- Результаты тестирования
- Заключение по проведенной работе

Виртуальная память

Необходимо создать рабочую модель виртуальной памяти. Требуется взять алгоритм аллокации памяти и использовать его совместно с алгоритмом замещения страниц. При разработке предусмотреть следующий интерфейс использования:

- CreateMemory
- Malloc
- Free

Параметры функций могут отличаться в зависимости от выбранных алгоритмов. Исследовать полученную модель виртуальной памяти – ее плюсы и минусы. Возможно сделать данные алгоритмы, переопределив стандартные механизмы аллокации языка C++. Также проработать модель тестирования полученного программного решения.

Сервер сообщений

Необходимо создать собственный сервер сообщений. В качестве механизма передачи сообщений возможно использовать следующие:

1. Pipes
2. Sockets
3. Files/Shared memory

При работе с сервером сообщений необходимо предусмотреть следующие механизмы (в зависимости от варианта):

- Долговременное хранение сообщений
- Транзактивность
- Система имен очередей
- Возможность настройки переадресации сообщений по фильтрам
- Приоритеты сообщений
- И другие возможные, в зависимости от варианта

Основные операции, которые должен поддерживать сервер сообщений и библиотека по работе с ним:

- CreateQueue
- DeleteQueue
- ConnectToQueue
- Push (Send)
- Pop/Top (Receive)

Текстовый препроцессор

Проработать полноценный текстовый препроцессор. За основу можно взять такие текстовые препроцессоры, как ed, sed и прочие. Основные особенности текстового препроцессора, которые могут быть реализованы (в зависимости от варианта):

- Транзактивность операций
- Кеширование при поиске
- Поддержка регулярных выражений
- Использование технологии MemoryMap
- И др.

Планирование процессов и потоков

Необходимо создать и протестировать библиотеку, реализующую функционал по работе с потоками. Библиотека должна быть надстройкой над существующими системными вызовами. Возможные особенности библиотеки (в зависимости от варианта):

- Приоритезация потоков

- Решение взаимоблокировок
- Упорядочивание ресурсов
- Удобная обертка при работе с несколькими аргументами при создании потоков
- Расписание потоков
- И другие

Создание клиента для передачи мгновенных личных сообщений

На основе любой из выбранных технологий:

- Pipes
- Sockets
- Сервера очередей
- И другие

Создать собственный клиент быстрых сообщений (возможно, и сервер – зависит от выбранной архитектуры), который бы работали в рамках сети. Также есть возможность создания собственного клиента быстрых сообщений для уже существующих систем обмена сообщений (например, telegram).

Проектирование консольной клиент-серверной игры

На основе любой из выбранных технологий:

- Pipes
- Sockets
- Сервера очередей
- И другие

Создать собственную игру более, чем для одного пользователя. Игра может быть устроена по принципу: клиент-клиент, сервер-клиент.

Варианты

Список вариантов

Консоль-серверная игра. Необходимо написать консоль-серверную игру. Необходимо написать 2 программы: сервер и клиент. Сначала запускается сервер, а далее клиенты соединяются с сервером. Сервер координирует клиентов между собой. При запуске клиента игрок может выбрать одно из следующих действий (возможно больше, если предусмотрено вариантом):

- Создать игру, введя ее имя
- Присоединиться к одной из существующих игр по имени игры

1. Морской бой. Общение между сервером и клиентом необходимо организовать при помощи pipe'ов. Каждый игрок должен при запуске ввести свой логин. Для каждого игрока должна вестись статистика игр (сколько побед/поражений). Игрок может посмотреть свою статистику
2. Морской бой. Общение между сервером и клиентом необходимо организовать при помощи pipe'ов. Каждый игрок должен при запуске ввести свой логин. Должна быть предоставлена возможность отправить приглашение на игру другому игроку по логину
3. Морской бой. Общение между сервером и клиентом необходимо организовать при помощи очередей сообщений (например, ZeroMQ). Каждый игрок должен при запуске ввести свой логин. Для каждого игрока должна вестись статистика игр (сколько побед/поражений). Игрок может посмотреть свою статистику
4. Морской бой. Общение между сервером и клиентом необходимо организовать при помощи очередей сообщений (например, ZeroMQ). Каждый игрок должен при запуске ввести свой логин. Должна быть предоставлена возможность отправить приглашение на игру другому игроку по логину
5. Морской бой. Общение между сервером и клиентом необходимо организовать при помощи memory map. Каждый игрок должен при запуске ввести свой логин. Для каждого игрока должна вестись статистика игр (сколько побед/поражений). Игрок может посмотреть свою статистику
6. Морской бой. Общение между сервером и клиентом необходимо организовать при помощи memory map. Каждый игрок должен при запуске ввести свой логин. Должна быть предоставлена возможность отправить приглашение на игру другому игроку по логину
7. «Быки и коровы» (угадывать необходимо слова). Общение между сервером и клиентом необходимо организовать при помощи pipe'ов. При создании каждой игры необходимо указывать количество игроков, которые будут участвовать. То есть угадывать могут несколько игроков. Если кто-то из игроков вышел из игры, то игра должна быть продолжена.
8. «Быки и коровы» (угадывать необходимо числа). Общение между сервером и клиентом необходимо организовать при помощи pipe'ов. При

создании каждой игры необходимо указывать количество игроков, которые будут участвовать. То есть угадывать могут несколько игроков. Должна быть реализована функция поиска игры, то есть игрок пытается войти в игру не по имени, а просто просит сервер найти ему игру.

9. «Быки и коровы» (угадывать необходимо слова). Общение между сервером и клиентом необходимо организовать при помощи очередей сообщений (например, ZeroMQ). При создании каждой игры необходимо указывать количество игроков, которые будут участвовать. То есть угадывать могут несколько игроков. Если кто-то из игроков вышел из игры, то игра должна быть продолжена.
10. «Быки и коровы» (угадывать необходимо числа). Общение между сервером и клиентом необходимо организовать при помощи очередей сообщений (например, ZeroMQ). При создании каждой игры необходимо указывать количество игроков, которые будут участвовать. То есть угадывать могут несколько игроков. Должна быть реализована функция поиска игры, то есть игрок пытается войти в игру не по имени, а просто просит сервер найти ему игру.
11. «Быки и коровы» (угадывать необходимо слова). Общение между сервером и клиентом необходимо организовать при помощи memory map. При создании каждой игры необходимо указывать количество игроков, которые будут участвовать. То есть угадывать могут несколько игроков. Если кто-то из игроков вышел из игры, то игра должна быть продолжена.
12. «Быки и коровы» (угадывать необходимо числа). Общение между сервером и клиентом необходимо организовать при помощи memory map. При создании каждой игры необходимо указывать количество игроков, которые будут участвовать. То есть угадывать могут несколько игроков. Должна быть реализована функция поиска игры, то есть игрок пытается войти в игру не по имени, а просто просит сервер найти ему игру.

Сравнение алгоритмов аллокаторов памяти (детальное описание задания описано выше). Каждый аллокатор должен обладать следующим интерфейсом (могут быть отличия в зависимости от особенностей алгоритма):

- Allocator* createMemoryAllocator(void *realMemory, size_t memory_size) (создание аллокатора памяти размера memory_size)
- void* alloc(Allocator * allocator, size_t block_size) (выделение памяти при помощи аллокатора размера block_size)
- void* free(Allocator * allocator, void * block) (возвращает выделенную память аллокатору)

13. Необходимо сравнить два алгоритма аллокации: списки свободных блоков (первое подходящее) и блоки по 2 в степени n
14. Необходимо сравнить два алгоритма аллокации: списки свободных блоков (первое подходящее) и алгоритм Мак-Кьюзи-Кэрелса
15. Необходимо сравнить два алгоритма аллокации: списки свободных блоков (первое подходящее) и алгоритм двойников

16. Необходимо сравнить два алгоритма аллокации: алгоритм Мак-Кьюзи-Кэрелса и блоки по 2 в степени n
17. Необходимо сравнить два алгоритма аллокации: алгоритм Мак-Кьюзи-Кэрелса и алгоритм двойников
18. Необходимо сравнить два алгоритма аллокации: блоки по 2 в степени n и алгоритм двойников
19. Необходимо сравнить два алгоритма аллокации: списки свободных блоков (наиболее подходящее) и блоки по 2 в степени n
20. Необходимо сравнить два алгоритма аллокации: списки свободных блоков (наиболее подходящее) и алгоритм Мак-Кьюзи-Кэрелса
21. Необходимо сравнить два алгоритма аллокации: списки свободных блоков (наиболее подходящее) и алгоритм двойников

Клиент-серверная система для передачи мгновенных сообщений. Базовый функционал должен быть следующим:

- Клиент может присоединиться к серверу, введя логин
- Клиент может отправить сообщение другому клиенту по его логину
- Клиент в реальном времени принимает сообщения от других клиентов

22. Необходимо предусмотреть возможность создания «групповых чатов». Связь между сервером и клиентом должна быть реализована при помощи pipe'ов
23. Необходимо предусмотреть возможность создания «групповых чатов». Связь между сервером и клиентом должна быть реализована при помощи очередей сообщений (например, ZeroMQ)
24. Необходимо предусмотреть возможность создания «групповых чатов». Связь между сервером и клиентом должна быть реализована при помощи memory map
25. Необходимо предусмотреть возможность хранения истории переписок (на сервере) и поиска по ним. Связь между сервером и клиентом должна быть реализована при помощи pipe'ов
26. Необходимо предусмотреть возможность хранения истории переписок (на сервере) и поиска по ним. Связь между сервером и клиентом должна быть реализована при помощи очередей сообщений (например, ZeroMQ)
27. Необходимо предусмотреть возможность хранения истории переписок (на сервере) и поиска по ним. Связь между сервером и клиентом должна быть реализована при помощи memory map
28. Необходимо предусмотреть возможность отправки отложенного сообщения или себе, или другому пользователю. При выходе с сервера отправка всё равно должна осуществиться. Связь между сервером и клиентом должна быть реализована при помощи pipe'ов
29. Необходимо предусмотреть возможность отправки отложенного сообщения или себе, или другому пользователю. При выходе с сервера отправка всё равно должна осуществиться. Связь между сервером и

клиентом должна быть реализована при помощи очередей сообщений (например, ZeroMQ)

30. Необходимо предусмотреть возможность отправки отложенного сообщения или себе, или другому пользователю. При выходе с сервера отправка всё равно должна осуществиться. Связь между сервером и клиентом должна быть реализована при помощи memory map

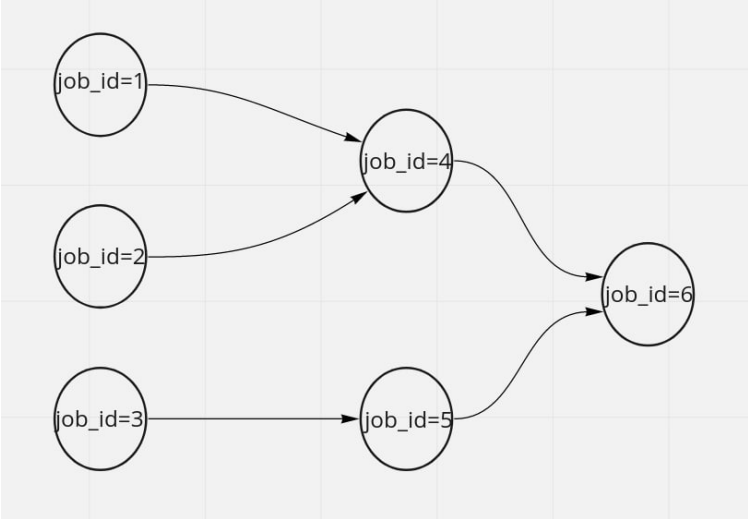
Создание планировщика «процессов-задач»:

31. См. приложение DAG_Job_Scheduler
32. См. приложение DAG_Job_Scheduler
33. См. приложение DAG_Job_Scheduler
34. См. приложение DAG_Job_Scheduler
35. См. приложение DAG_Job_Scheduler
36. См. приложение DAG_Job_Scheduler
37. См. приложение DAG_Job_Scheduler
38. См. приложение DAG_Job_Scheduler
39. См. приложение DAG_Job_Scheduler

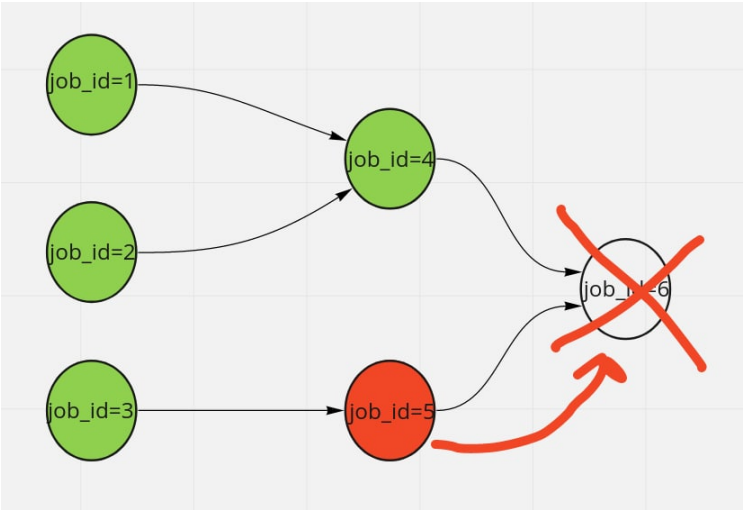
Создание планировщика DAG*'а «джобов» (jobs)**

На языке C\C++ написать программу, которая:

1. По конфигурационному файлу в формате yaml, json или ini принимает спроектированный DAG джобов и проверяет на корректность: отсутствие циклов, наличие только одной компоненты связанности, наличие стартовых и завершающих джоб. Структура описания джоб и их связей произвольная.

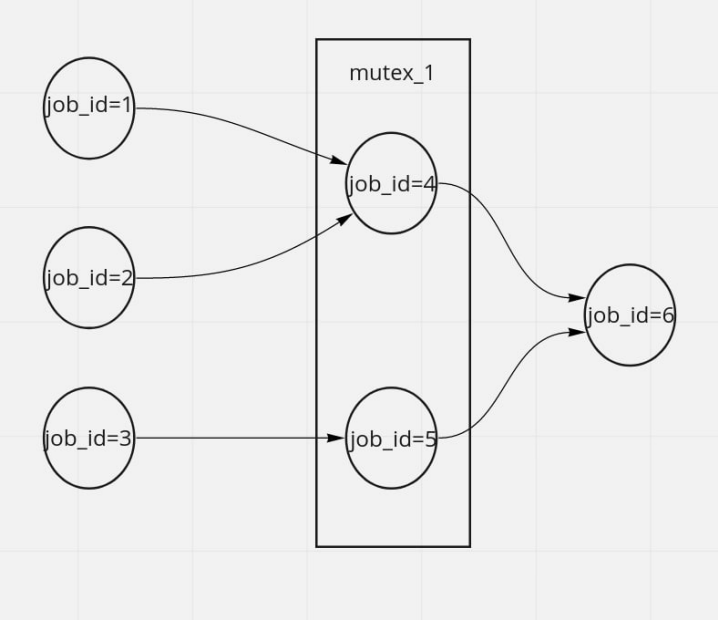


2. При завершении джобы с ошибкой, необходимо прервать выполнение всего DAG'а и всех запущенных джоб.



3. (на оценку 4) Джобы должны запускаться максимально параллельно. Должны быть ограниченны параметром – максимальным числом одновременно выполняемых джоб.

4. (на оценку 5) Реализовать для джобов один из примитивов синхронизации мьютекс\семафор\барьер. То есть в конфиге дать возможность определять имена семафоров (с их степенями)\мьютексов\барьеров и указывать их в определение джобов в конфиге. Джобы указанные с одним мьютексом могут выполняться только последовательно (в любом порядке допустимом в DAG). Джобы указанные с одним семафором могут выполнятся параллельно с максимальным числом параллельно выполняемых джоб равным степени семафору. Джобы указанные с одним барьером имеют следующие свойство – зависимые от них джобы начнут выполнятся не раньше того момента времени, когда выполнятся все джобы с указанным барьером.



* DAG - Directed acyclic graph. Направленный ациклический граф.

** Джоб(Job) – процесс, который зависит от результата выполнения других процессов (если он не стартовый), которые исполняются до него в DAG, и который порождает данные от которых может быть зависят другие процессы, которые исполняются после него в DAG (если он не завершающий).

Варианты:

31. Yaml\Mutex

32. Yaml\Semaphore
33. Yaml\Barrier
34. Json\Mutex
35. Json\Semaphore
36. Json\Barrier
37. Ini\Mutex
38. Ini\Semaphore
39. Ini\Barrier