БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра информатики

Факультет НиДО

Специальность ИиТП

Контрольная работа № 2

по дисциплине «Операционные системы и среды»

Выполнил студент: Дегтярев А.А.

группа 393551

Зачетная книжка № 902021-26

Минск 2016

**Вариант 2**

*Реализация многозадачности и многопоточности в операционных системах.*

Цель работы – изучение, сопоставление и анализ существующих разновидностей архитектур, особенностей организации, функционирования операционных систем: общего назначения, специализированных, реального времени.

Пожалуй, стоит начать с определения многозадачности - это свойство операционной системы или «среды выполнения» обеспечивать возможность параллельной (или псевдопараллельной(конкурентной)) обработки нескольких задач. Истинная многозадачность операционной системы возможна только в распределённых вычислительных системах.

Итак, в общем, существует 2 типа многозадачности: **Процессная** и **поточная**

Процессная – связана с процессами, одновременно выполняющихся программах, управляемая планировщиком ОС;

Поточная – с потоком;

В чистой форме многозадачные среды обеспечивают *разделение ресурсов*, в большинстве случаев – ресурсов(времени) процессора и памяти. Задача активизируеся в строго определенные интервалы времени.

Усложненные системы проводят разделение ресурсов динамически:

Каждая задача имеет свой приоритет, в соответствии с которым получает процессорное время и память

Система организует очереди задач так, чтобы все задачи получили ресурсы, в зависимости от приоритетов и стратегии системы

Система организует обработку прерываний, по которым задачи могут активироваться, деактивироваться и удаляться

По окончании положенного кванта времени ядро временно переводит задачу из состояния выполнения в состояние готовности, отдавая ресурсы другим задачам. При нехватке памяти страницы невыполняющихся задач могут быть вытеснены на диск (своппинг), а потом, через определённое системой время, восстанавливаться в памяти

Система обеспечивает защиту адресного пространства задачи от несанкционированного вмешательства других задач

Система обеспечивает защиту адресного пространства своего ядра от несанкционированного вмешательства задач

Система распознаёт сбои и зависания отдельных задач и прекращает их

Система решает конфликты доступа к ресурсам и устройствам, не допуская тупиковых ситуаций общего зависания от ожидания заблокированных ресурсов

Система гарантирует каждой задаче, что рано или поздно она будет активирована

Система обрабатывает запросы реального времени

Система обеспечивает коммуникацию между процессами

Основной трудностью реализации многозадачной среды является её надёжность, выраженная в защите памяти, обработке сбоев и прерываний, предохранении от зависаний и тупиковых ситуаций.

Кроме надёжности, многозадачная среда должна быть эффективной. Затраты ресурсов на её поддержание не должны: мешать процессам проходить, замедлять их работу, резко ограничивать память.

Стоит отметить что многозадачность может быть реализована не только в операционной, но и языковой среде. Особенно актуально может быть для исполняемых языков, либо языков с виртуальной машиной, которые могут использовать планировщик задач;

Варианты псевдопараллельной многозадачности:

**Невытесняющая –** ос одновременно загружает несколько приложений, процессорное время предоставляется только одному из них. Для выполнения фоновой задачи, приложение должно быть активировано.

**Совместная –** переключением задачи управляет сама задача, указывая что готова отдать процессорное время другим задачам. Активная программа получает все процессорное время, периодически отдавай управление следующей задаче.

**Вытесняющая –** ос сама передает управление от одной задачи к другой, по окончанию операций ввода/вывода, событий, истечению таймеров квантов времени или поступлению каких-то иных сигналов; Распределение осуществляется планировщиком процессов. Задачи могут иметь приоритеты, что обеспечивает гибкое управление процессами. Используется во всех современных пользовательских ос.

Многопоточность – это свойство ос/вм или приложения в возможности процесса состоять из нескольких потоков, которые могут выполнятся без предписанного порядка во времени «паралелльно». В отличие от процессов, потоки выполняются в одном адресном пространстве процесса, имеют общие дескрипторы; Потоки являются легковесными сущностями. Существенно меньше затрат по времени на создание потока. В случаях с аппаратной поддержкой потоков – процессор может выполнять несколько потоков одновременно за каждый такт; Так же в современных многоядерных архитектурах процессоров возможна одновременная многопоточность.

Потоки в **userspace**(в рамках ос) модели N:1 предполагается, что все потоки выполнения уровня пользователя отображаются на единую планируемую сущность уровня ядра, и ядро ничего не знает о составе прикладных потоков выполнения. При таком подходе переключение контекста может быть сделано очень быстро, и, кроме того, он может быть реализован даже на простых ядрах, которые не поддерживают многопоточность. Однако, одним из главных недостатков его является то, что в нём нельзя извлечь никакой выгоды из аппаратного ускорения на многопоточных процессорах или многопроцессорных компьютерах, потому что только один поток выполнения может быть запланирован на любой момент времени. Эта модель используется в GNU Portable Threads.

-GNU Portable Threads

-FSU Pthreads

-Thread Manager компании Apple

-REALbasic (включая API для совместного использования потоков)

-Netscape Portable Runtime (включая реализацию файберов в пользовательском пространстве)

**kernelspace** потоки реализрваны на уровне ядра ос, реализуют таблицу потоков. Более тяжеловесные(потребляют много памяти, адреса и стек). Наиболее простой способ, и используется в большинстве современных ОС.

-Light Weight Kernel Threads (LWKT) в различных версиях BSD

-Потоковость MxN

-Библиотека потоков POSIX (NPTL) для Linux, реализация стандарта POSIX Threads

-Apple Multiprocessing Services, версия 2.0 и последующие, использует встроенное микроядро в Mac OS 8.6, в более поздних версиях сделана модификация с целью последующего сопровождения.

-Windows начиная с Windows 95, Windows NT и после них.

**fibers –** несколькоuserspace потоков исполняюшихся в одном kernelspace потоке. В модели M:N некоторое число M прикладных потоков выполнения отображаются на некоторое число N сущностей ядра или «виртуальных процессоров». Модель является компромиссной между моделью уровня ядра («1:1») и моделью уровня пользователя («N:1»). Вообще говоря, «M:N» потоковость системы являются более сложной для реализации, чем ядро или пользовательские потоки выполнения, поскольку изменение кода как для ядра, так и для пользовательского пространства не требуется. В M:N реализации библиотека потоков отвечает за планирование пользовательских потоков выполнения на имеющихся планируемых сущностях. При этом переключение контекста потоков делается очень быстро, поскольку модель позволяет избежать системных вызовов. Тем не менее, увеличивается сложность и вероятность инверсии приоритетов, а также неоптимальность планирования без обширной (и дорогой) координации между пользовательским планировщиком и планировщиком ядра.

«Scheduler activations» используется в собственной библиотеке приложений потоков POSIX для NetBSD (модель M:N в противоположность модели 1:1 ядра или модели приложений пользовательского пространства)

-Marcel из проекта PM2

-ОС для суперкомпьютера Tera/Cray MTA

-Windows 7

Файберы могут быть реализованы без поддержки операционной системы, хотя некоторые операционные системы и библиотеки предоставляют явную поддержку для них.

Библиотека Win32 содержит API для файберов[4] (Windows NT 3.51 SP3 и выше)

Ruby как реализация «зелёных потоков»

Многие языки программирования поддерживают потоки иначе. Большинство реализаций С и С++ (до C++11) сами по себе не обеспечивают прямой поддержки потоков, но обеспечивают доступ к потокам, предоставляемым операционной системой, через API. Некоторые языки программирования более высокого уровня Java, Python, и .NET, предоставляют потоковость разработчику в виде абстрактной специфической платформы, отличающейся от реализации потоков в среде выполнения разработчика. Ряд других языков программирования также пытается полностью абстрагировать концепцию параллелизма и потоковости от разработчика (Cilk, OpenMP, MPI…).

Некоторые интерпретирующие языки программирования, такие, как Руби и CPython (реализация Python), поддерживают потоки, но имеют ограничение, которое известно как глобальная блокировка интерпретатора (GIL). GIL является взаимной блокировкой исключений, выполняемых интерпретатором, которая может уберечь интерпретатор от одновременной интерпретации кода приложений в двух или более потоках одновременно, что фактически ограничивает параллелизм на многоядерных системах (в основном для потоков, связанных через процессор, а не для потоков, связанных через сеть).

В потоках/процессах возникают проблемы одновременного использования ресурсов. ОС Windows, Unix, и прочие предоставляют решения в виде различных примитивов синхронизации: мютексты(семафор с 1 ресурсом), семафоры(счетчик ресурсов), события (windows), критические секции(блокирующие доступ к ресурсу, легче и быстрее мютекса), условные переменные.

Доступ к каким-то участком памяти для нескольких процессов можно обеспечить используя участки разделяемой памяти. (однако стоит понимать что этот ресурс может быть изменен другим процессом).

Специализированные устройства вроде векторных процессоров – позволяют исполнять операцию над всеми элементами массива данных. Распараллеливание в этом случае чаще выполняется уже на этапе компиляции – оптимизации задачи для конкретного механизма векторизации;

В случае крупных серверных платформ используются распределенные ОС, которые динамический или автоматически распределяют работы по различным машинам системы для обработки. В масштабах вычислительной системы – распределенная ОС может рассматриваться как набор ос отдельных компьютеров сети. Системы работают независимо в смысле принятия решения о создании/завершении собственных процессов и управления ресурсами;

В случае ОС реального времени (ОСРВ) – большинство задач выполняется в соответствии с приоритетом. Чем выше приоритет – тем должна быть больше реактивность задачи; Высокая реактивность достигается все тем же приоритетным вытесняющим планированием. Планировщику разрешается останавливать выполнение любой задачи в произвольный момент времени, если установлено, что задача должна быть запущена незамедлительно.

Для синхронизации ОСРВ обычно использует сигналы и двоичные семафоры.

Стоит отметить, что основное отличие ОСРВ от обычных ОС лишь в строгой детерминированности времени исполнения процесса.