расчет заработной платы, формирование отчетностей и деклараций; Спонтания – уникальный онлайн-ресурс для проведения видеоконференций, главной особенностью которого является возможность использования с различных устройств, работающих платформе Android, IOS [4]. С его помощью возможно проведение переговоров из любых стран мира в реальном времени.

Облачные технологии предоставляют предприятиям малого и среднего бизнеса возможность повышения эффективности своей работы, при этом снизив финансовые затраты на улучшение программного обеспечения, подобрав тарифный план с необходимым набором функций.

Список литературы

1. Облачные вычисления [Электронный ресурс]. – URL: https://ru.wikipedia.org/wiki/Облачные_вычисления (Дата обращения: 20.12.2015).

2. Во что обходится миграция систем в облака? [Электронный ресурс]// CNews: Издание о высоких технологиях. – URL: http://www.cnews.ru. (Дата обращения: 20.12.2015).

3. Мартышкин А.И. Исследование алгоритмов планирования процессов в системах реального времени // Современные методы и средства обработки пространственно-временных сигналов: сборник статей XIII Всероссийской научно-технической конференции / Под ред. И.И. Сальникова. – Пенза, 2015. – С.118-124.

4. Бершадская Е.Г., Зубков А.В. Оценка возможностей моделирования производственным систем // Международный студенческий научный вестник. 2015, – № 3-2, – С. 265-266.

ОБЗОР СРЕДСТВ УПРАВЛЕНИЯ ПРОЦЕССАМИ И РЕСУРСАМИ МНОГОПРОЦЕССОРНЫХ ОПЕРАЦИОННЫХ СИСТЕМ

Карасева Е.А., Мартышкин А.И.

Пензенский государственный технологический университет, Пенза, e-mail: alexey314@ya.ru

В качестве примера рассмотрим семафоры, мьютексы и мониторы.

В 1965 году Дейкстра предложил использовать переменную для подсчета сигналов запуска, сохраненных на будущее [1]. Им был предложен новый тип переменных – **семафоры,** значение которых может быть нулем (в случае отсутствия сохраненных сигналов активизации) или некоторым положительным числом, соответствующим количеству отложенных активизирующих сигналов.

Дейкстра предложил две операции, *down*и*up* (обобщения *sleep* и *wakeup*)**.** Операция *down* сравнивает значение семафора с нулем. Если значение семафора больше нуля, операция *down* уменьшает его и просто возвращает управление. Если значение семафора равно нулю, процедура *down* не возвращает управление процессу, а процесс переводится в состояние ожидания. Все операции проверки значения семафора, его изменения и перевода процесса в состояние ожидания выполняются как единое и неделимое **элементарное действие.** Тем самым гарантируется, что после начала операции ни один процесс не получит доступа к семафору до окончания или блокирования операции. Элементарность операции чрезвычайно важна для разрешения проблемы синхронизации и предотвращения состояния состязания.

Операция *up* увеличивает значение семафора. Если с этим семафором связаны один или несколько ожидающих процессов, которые не могут завершить более раннюю операцию down, один из них выбирается системой (например, случайным образом) и ему разрешается завершить свою операцию *down*. Таким образом, после операции up, примененной к семафору, связанному с несколькими ожидающими процессами, значение семафора так и останется равным 0, но число ожидающих процессов уменьшится на единицу. Операция увеличения значения семафора и активизации процесса тоже неделима. Ни один процесс не может быть блокирован во время выполнения операции up, как ни один процесс не мог быть блокирован во время выполнения операции wakeup в предыдущей модели [1].

Иногда используется упрощенная версия семафора, называемая мьютексом [1]. Мьютекс не способен считать, он может лишь управлять взаимным исключением доступа к совместно используемым ресурсам или кодам. Реализация мьютекса проста и эффективна, что делает использование мьютексов особенно полезным в случае потоков, действующих только в пространстве пользователя. **Мьютекс** – переменная, которая может находиться в одном из двух состояний: блокированном или неблокированном. Поэтому для описания мьютекса требуется всего один бит, хотя чаще используется целая переменная, у которой 0 означает неблокированное состояние, а все остальные значения соответствуют блокированному состоянию. Значение мьютекса устанавливается двумя процедурами. Если поток (или процесс) собирается войти в критическую область, он вызывает процедуру *mutexlock.* Если мьютекс не заблокирован (то есть вход в критическую область разрешен), запрос выполняется и вызывающий поток может попасть в критическую область [1, 2].

Напротив, если мьютекс заблокирован, вызывающий поток блокируется до тех пор, пока другой поток, находящийся к критической области, не выйдет из нее, вызвав процедуру *mutex_unlock.* Если мьютекс блокирует несколько потоков, то из них случайным образом выбирается один.

В случае потоков ситуация кардинально меняется, поскольку нет прерываний по таймеру, останавливающих слишком долго работающие потоки. Поток, пытающийся получить доступ к семафору и находящийся в состоянии активного ожидания, зациклится навсегда, поскольку он не позволит предоставить процессор другому потоку, желающему снять блокировку.

Если два или больше процессов разделяют частично или полностью адресные пространства, различие между процессами и потоками частично размывается, но тем не менее все равно остается. Два процесса с общим адресным пространством все равно обладают разными открытыми файлами, аварийными таймерами и прочими характеристиками, присущими процессам, в то время как два потока, разделяющие адресное пространство, разделяют и все остальное. И в любом случае несколько процессов, совместно использующих адресное пространство, никогда не будут столь же эффективны, как потоки на уровне пользователя, поскольку управление потоками всегда происходит через ядро.

Чтобы упростить написание программ, в 1974 году Хоар и Бринч Хансен предложили примитив синхронизации более высокого уровня, называемый **монитором** [1]**.** Монитор – набор процедур, переменных и других структур данных, объединенных в особый модуль или пакет. Процессы могут вызывать процедуры монитора, но у процедур, объявленных вне монитора, нет прямого доступа к внутренним структурам данных монитора.

Реализации взаимных исключений способствует важное свойство монитора: при обращении к монитору в любой момент времени активным может быть только один процесс. Мониторы являются структурным компонентом языка программирования, поэтому компилятор знает, что обрабатывать вызовы процедур монитора следует иначе, чем вызовы остальных процедур. Обычно при вызове процедуры монитора первые несколько команд процедуры проверяют, нет ли в мониторе активного процесса. Если активный

процесс есть, вызывающему процессу придется подождать, в противном случае запрос удовлетворяется.

Переменные состояния не являются счетчиками. В отличие от семафоров они не аккумулируют сигналы, чтобы впоследствии воспользоваться ими. Это означает, что в случае выполнения операции signal на переменной состояния, с которой не связано ни одного блокированного процесса, сигнал будет утерян. Проще говоря, операция wait должна выполняться прежде, чем signal. Это правило существенно упрощает реализацию. На практике это правило не создает проблем, поскольку отслеживать состояния процессов при необходимости не очень трудно. Процесс, который собирается выполнить signal, может оценить необходимость этого действия по значениям переменных.

Существует несколько языков программирования, поддерживающих мониторы, хотя и не всегда в соответствии с моделью Хоара и Бринча Хансена. Один из таких языков – Java, объектно-ориентированный язык, поддерживающий потоки на уровне пользователя и позволяющий группировать методы (процедуры) в классы. Добавление в описание метода ключевого слова synchronized гарантирует, что если хотя бы один поток начал выполнение этого метода, ни один другой поток не сможет выполнять другой синхронизированный метод из этого класса.

Мониторы являются структурным компонентом языка программирования, и компилятор должен их распознавать и организовывать взаимное исключение. В Pascal, C и многих других языках нет мониторов. В этих языках также нет и семафоров, но их легко добавить: нужно всего лишь присоединить к библиотеке две короткие программы, написанные на ассемблере и реализующие системные вызовы up и down. Компиляторы при этом не обязаны знать об их существовании. Разумеется, операционная система должна знать о семафорах, но даже если у вас операционная система с семафорами, вы можете писать программы для нее на C или C++. Если же операционная система с мониторами, необходим язык со встроенными мониторами.

Другая проблема, связанная с мониторами и семафорами, состоит в том, что они были разработаны для решения задачи взаимного исключения в системе с одним или несколькими процессорами, имеющими доступ к общей памяти. Помещение семафоров в разделенную память с защитой в виде команд TSL может исключить состояния состязания. Эти примитивы будут неприменимы в распределенной системе, состоящей из нескольких процессоров с собственной памятью у каждого, связанных локальной сетью.

Вывод из всего вышесказанного следующий: семафоры являются примитивами слишком низкого уровня, а мониторы могут использоваться только в некоторых языках программирования.

Список литературы

1. Таненбаум Э., Бос Х. Современные операционные системы. – СПб.: Питер, 2015. – 1120 с.

2. Бершадская Е.Г. Анализ технологий поддержки научных исследований // XXI век: итоги прошлого и проблемы настоящего плюс. 2015. – № 3 (25). – С. 11–17.

ПРИМЕНЕНИЕ ОБЛАЧНЫХ ТЕХНОЛОГИЙ

Кашицин И.М., Сальников И.И.

Пензенский государственный технологический университет, Пенза, e-mail: alexey314@ya.ru

Суть концепции облачных технологий заключается в предоставлении конечным пользователям удаленного динамического доступа к услугам, вычислительным ресурсам и приложениям через Интернет. Большинство сервис-провайдеров предлагают облачные вычисления в форме VPS-хостинга, виртуального хостинга, и ПО-как-услуга(SaaS). Облачные услуги долгое время предоставлялись в форме SaaS, например, Microsoft Hosted Exchange и SharePoint.

В 2011 году WINDOWS AZURE была объявлена коммерческой системой. Как и традиционная ОС, WINDOWS AZURE позволяет запускать приложения и хранить данные, но происходит это не на компьютере пользователя, а в вычислительных облаках.

Существуют два типа рабочих версий облачного приложения: веб-роль (Web role) и рабочая роль (Worker role). Первая умеет обрабатывать HTTP– или HTTPS-запросы, и на ее виртуальной машине (ВМ) запущен сервер Internet Information Services (IIS). Программист имеет возможность создать версию веб-роли с помощью ASP.NET либо Windows Communication Foundation (WCF), а также воспользоваться любой другой технологией .NET, работающей с IIS. Приложение может быть создано на любом языке программирования.

Приложения, созданные на основе WINDOWS AZURE, предоставляются как сервис физическим лицам, корпоративным пользователям или и тем, и другим одновременно. С помощью WINDOWS AZURE независимый разработчик программного обеспечения может создавать приложения для бизнес-пользователей, применяя принципы программного обеспечения как сервиса.

Примером может послужить решение, разработанное американской компанией Alinean, Inc. Ее сфера деятельности – предоставление по запросу аналитических средств в области анализа продаж и маркетинга. Системы Alinean позволяют оценить нужды и возможности бизнеса в будущем, предложить решение для наращивания мощностей и подсчитать, когда начнут окупаться инвестиции. Пользователями Alinean являются корпоративные клиенты, находящиеся в разных уголках земного шара. Среди них IBM, HP, Microsoft, Intel, AT&T, VMware, Oracle, Siemens, Symantec и др. В дата-центре Alinean, находящемся в Орландо (Флорида, США), сервис по запросу предоставляли 20 серверов, работающих 24 часа в сутки семь дней в неделю. Объем бизнеса рос, и мощностей стало не хватать, да и содержание внутреннего ЦОД становилось все дороже.

Благодаря масштабируемости WINDOWS AZURE позволяет вести учет огромного количества пользователей. Создавая облачноерешение, компания-разработчик может рассчитывать не только на корпорации, но и на физических лиц. Такое приложение было сделано новозеландской компанией TicketDirect International, которая, работая в онлайновом режиме, осуществляет 45 % всех продаж билетов на культурные и спортивные мероприятия Новой Зеландии. Предыдущая, традиционная, система продажи билетов, функционировавшая на базе Microsoft SQL Server 7 и SQL Server 2000, была написана на Visual Basic 6. Приложение без проблем обслуживало несколько сотен продаж в течение часа. Но в дни распродаж, когда объявлялась скидка на посещение популярного мероприятия, до системы пытались одновременно «достучаться» тысячи людей [1]. Неудивительно, что компьютерный парк продавца билетов не выдерживал такого наплыва пользователей.

WINDOWS AZURE предоставила TicketDirect масштабируемую инфраструктуру как сервис с возможностью оплаты по факту. В результате в момент распродаж приложение начинает использовать дополнительные мощности. Теперь компании TicketDirect не потребуется закупать оборудование только для того, чтобы покрыть временные всплески