УДК 519.872

ИССЛЕДОВАНИЕ И АНАЛИЗ МАТЕМАТИЧЕСКИХ МОДЕЛЕЙ ДЛЯ ОЦЕНКИ ПРОИЗВОДИТЕЛЬНОСТИ ПРИМИТИВОВ СИНХРОНИЗАЦИИ ПРОЦЕССОВ В РЕКОНФИГУРИРУЕМЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ

[1]Мартышкин А.И., [2]Здерев С.Г.

[1]ФГБОУ ВО «Пензенский государственный технологический университет», Пенза, e-mail: Alexey314@yandex.ru;

[2]ФГБОУ ВО «Пензенский государственный университет», Пенза, e-mail: Serg.zderev@gmail.com

В статье представлены результаты исследования математических моделей для оценки производительности примитивов синхронизации процессов в реконфигурируемых вычислительных системах. Проведен анализ известных примитивов синхронизации многопоточных приложений, таких как алгоритм Деккера, семафор, мьютекс. В статье исследуется семафор, модель применения взаимоисключения, являющаяся самым распространенным и часто используемым примитивом синхронизации. В представленном исследовании рассматриваются модели и методы планирования «живых» семафоров, часто применяемых для синхронизации взаимодействующих параллельных процессов при их доступе к общему ресурсу в параллельных вычислительных системах, среди которых можно выделить и реконфигурируемые вычислительные системы. В настоящем исследовании для оценки временных потерь процессов из-за образования перед семафором очередей ожидающих процессов предлагаются модели на основе систем и сетей массового обслуживания. Описываются бесприоритетные и приоритетные модели взаимодействия с семафором. Приводится математическая модель оценки потери времени для процессов, ожидающих на семафоре. Основные результаты исследования и расчетов приведены на рисунках в виде графиков. Полученные результаты говорят о том, что применение приоритетного планирования в отличие от бесприоритетного дает положительный эффект по ожиданию в очереди к семафору примерно на 30–38 % в зависимости от процессоров. В заключение приводятся основные выводы по проведенному исследованию.

Ключевые слова: аналитическое моделирование, математическая модель, общий ресурс, планирование процессов, примитив синхронизации, приоритет, реконфигурируемая вычислительная система, семафор

RESEARCH AND ANALYSIS OF MATHEMATICAL MODELS FOR EVALUATING THE PERFORMANCE OF PROCESS SYNCHRONIZATION PRIMITIVES IN RECONFIGURABLE COMPUTING SYSTEMS

[1]Martyshkin A.I., [2]Zderev S.G.

[1]Penza State Technological University, Penza, e-mail: Alexey314@yandex.ru;

[2]Penza State University, Penza, e-mail: Serg.zderev@gmail.com

The article presents the results of a study of mathematical models for evaluating the performance of process synchronization primitives in reconfigurable computing systems. The article analyzes the well-known synchronization primitives of multithreaded applications, such as the Dekker algorithm, semaphore, mutex. The article examines the semaphore as a model for the application of mutual exclusion, which is the most common and frequently used synchronization primitive. In the presented study, models and methods of planning “live” semaphores are considered, which are often used to synchronize interacting parallel processes when they access a shared resource in parallel computing systems, among which reconfigurable computing systems can be distinguished. In this study, models based on queuing systems and networks are proposed to estimate the time losses of processes due to the formation of queues of waiting processes before the semaphore. Non-priority and prioritized models of interaction with a semaphore are described. A mathematical model for estimating time loss for processes waiting on a semaphore is given. The main results of the study and calculations are shown in the figures in the form of graphs. The results obtained suggest that the use of priority scheduling, in contrast to non-priority scheduling, has a positive effect on waiting in the semaphore queue by about 30-38%, depending on the processors. At the article concludes, the conclusions of the study are presented.

Keywords: analytical modeling, mathematical model, shared resource, process planning, synchronization primitive, priority, reconfigurable computing system, semaphore

В настоящее время известны два способа синхронизации процессов взаимодействия: методы взаимного исключения и методы синхронизации. Стремление более эффективно использовать ресурсы вычислительной машины стало толчком к развитию многозадачных операционных систем (ОС). Естественным продолжением многозадачности является многопоточность – способность ОС поддерживать выполнение нескольких потоков в рамках одного процесса, каждый из которых обладает такими характеристиками, как состояние выполнения потока, контекст, стек выполнения и т.п. [1]. Итак, использование потоков может значительно повысить производительность программы, так как их взаимодействие между собой не требует участия ядра ОС,

а создание потоков и переключение контекста происходят быстрее, чем планирование их выполнения. Например, в ОС Linux используется модель потоков (легковесные процессы), которые могут совместно использовать определенные ресурсы, а также немедленно отслеживать произведенные над ними изменения. Для реализации многопоточности с каждым потоком ассоциируется легковесный процесс. Поэтому, с одной стороны, потоки могут использовать общее адресное пространство и файловые дескрипторы, а с другой – за планирование их выполнения отвечает ядро. Тем не менее одновременная попытка обращения нескольких потоков к одному общему ресурсу (ОР) может обратиться негативными последствиями [2].

Целью исследования является синтез математических моделей для оценки производительности методов синхронизации процессов в реконфигурируемых вычислительных системах (РВС). Для систем, имеющих ОР (критические секции (КС)), следует использовать модели и методы, синхронизирующие процессы, которые могут конфликтовать при использовании КС. Основная задача, решаемая в настоящем исследовании – проблема производительности РВС с параллельными процессами.

Материалы и методы исследования

Примитивы синхронизации многопоточных приложений. Синхронизация между потоками, как и между процессами, имеет колоссальное значение. Для организации возможности работы с ОР, к которым необходим эксклюзивный доступ, в коде каждого потока выделяют некоторую часть, в которой используются эти ресурсы – КС. При этом требуется, чтобы во время выполнения КС одного потока ни один другой поток не выполнялся в своей КС, использующей те же ОР. Для этого необходимо реализовать один из механизмов взаимного исключения [3]. Важно избегать ситуаций, когда какие-либо два потока ждут друг друга для доступа к КС (взаимоблокировка) или когда разделяемые ресурсы недоступны одному из потоков из-за использования их другими потоками. Для решения данной задачи известно множество алгоритмов. Их можно разделить на программные (например, алгоритм Деккера) и аппаратные решения, основанные на поддержке процессорами (ЦП) таких атомарных инструкций, как Test-and-Set и Compare-and-Swap [3]. Далее рассмотрим наиболее перспективные из них.

Алгоритм Деккера. Является первым известным корректным решением задачи взаимного исключения. Контроль входа в КС осуществляется с помощью двух переменных-флагов и переменной «turn», показывающей, очередь какого потока подошла [4]. Поток сообщает о намеренности войти в КС, установив соответствующий флаг. Если флаг второго потока не установлен, можно безопасно войти в КС. Иначе поток снимает свой флаг и ожидает, пока переменная «turn» не укажет, что подошла его очередь. Преимущество алгоритма Деккера: не требуется наличия специальных атомарных команд «Test-and-Set». Недостатки: использование «busy-wait» цикла, возможность синхронизации между собой лишь двух потоков.

Семафор. Понятие семафора впервые сформулировано Дейкстрой [4]. Семафор – объект, ограничивающий количество потоков, которые могут одновременно находиться в КС. Семафоры можно разделить на считающие и бинарные. Считающие семафоры управляются двумя операциями, исторически называемыми V и P. Операция V увеличивает счетчик семафора на единицу, освобождая, таким образом, место для другого потока в КС. Если счетчик семафора положителен, операция P уменьшает его, а поток может безопасно войти в КС, иначе поток ожидает освобождения семафора. Для реализации семафора необходима аппаратная поддержка атомарных операций, например, «Compare-and-Swap».

Мьютекс. Является аналогом бинарного семафора, который может предоставлять дополнительные возможности, среди которых запоминание идентификатора, заблокировавшего процесса для проверки того, какой процесс пытается его освободить. Также мьютекс может быть рекурсивным, т.е. предоставлять возможность одному и тому же процессу или потоку заблокировать его несколько раз. Если потоку не удалось захватить мьютекс, он засыпает, используя соответствующий системный вызов.

Использование примитивов синхронизации часто приводит к возникновению «узких мест» в коде программы, что связано с продолжительным ожиданием освобождения КС. Данное явление принято называть «конкуренцией за блокировку» [3]. Еще одно понятие, тесно связанное с примитивами синхронизации и описывающее то, насколько хорошо может быть расширена система, – масштабируемость. В случае с блокировками наибольшее значение имеет в контексте увеличения числа потоков, которым требуется доступ к КС, и увеличения количества ЦП или ядер. Третья характеристика блокировок – структурность. Она описывает объем данных, которые защищены примитивом синхронизации. Так, например, один примитив синхронизации

может использоваться для всей структуры данных, или же множество примитивов для каждого элемента этой структуры [5]. Таким образом, возникает целый ряд проблем, связанных с производительностью блокировок, которые зависят как от аппаратных особенностей системы и непосредственной реализации примитивов синхронизации, так и от способа их применения в конкретной программе.

С целью анализа производительности используют утилиты-профилировщики. К примеру, профилировщик mutrace работает только с мьютексами из библиотеки pthread, оценивает время ожидания потоком доступа к КС, количество захватов мьютекса, а также количество потоков, которые не смогли получить доступ к КС с первого раза [6]. Другой профилировщик, входящий в HPCToolkit, основан на вычислении времени, которое потоки тратят на ожидание блокировки, и «blame shifting» в создании конкуренции за КС на поток, находившийся в это время внутри нее, также происходит суммирование времени по всем потокам и вычисление процента для каждого примитива синхронизации от общего простоя [7].

В настоящей статье исследуется модель применения взаимоисключения – семафор, являющийся самым распространенным и часто используемым примитивом синхронизации. Когда несколько процессов запрашивают ресурс, запрашивающий процесс ставится в очередь в соответствии с принципом семафора. В этом случае доступ к запрашиваемому ресурсу осуществляется через очередь. Процесс может занимать ОР в течение длительного времени, пока другой процесс ждет освобождения ОР, так что быстрый процесс может остановиться, ожидая запуска медленного процесса [8].

Можно определить два типа семафоров: живой и неживой. Наиболее часто используемый семафор называется живым, и в системе с N процессами и M ресурсами очередь к семафору может содержать не более N-M процессов. Итак, T – временной отрезок, в течение которого семафор занят выполнением запроса некоторого процесса; ς – интервал между поступлением запросов к семафору от процессов для получения доступа к ОР. Есть некий процесс p1, который занимает семафор S. Другие процессы в этот момент стоят в ожидании освобождения семафора S и затем также занимают его; так как семафор S занят, они ожидают его освобождения. Причем если N-M больше, чем ς / T, то чаще всего при очередном запросе процесса p1, ОР занят другим процессом [8]. Таким образом, ЦП в РВС тратит большую часть времени на планирование и диспетчеризацию процессов. И если происходит ситуация, при которой время переключения примерно равно T, в таком случае к семафору S выстраивается очередь из процессов, запрашивающих ОР. Только в идеализированном случае все процессы имеют одинаковые параметры ς и T. В реальных же системах ς и T чаще всего сильно различаются для каждого отдельного процесса.

Доступ к ОР предоставляет диспетчер задач (ДЗ), который производит распределение ресурса (рис. 1). Предположим, что сразу несколько процессов обращаются к семафору с равной интенсивностью поступления запросов, но с разным временем, затрачиваемым на их обработку. Получим среднее время ожидания в очереди к семафору одной заявки, которое можно рассчитать выражением [9]:

$$t_S = \frac{\sum_{i=1}^{M} \rho_i \tau_i}{1 - R}, \qquad (1)$$

где $\rho_i = \lambda_i \tau_i$ – нагрузка на семафор, создаваемая i-м процессом, R – суммарная нагрузка на семафор от всех процессов. Если каждый ЦП формирует один тип процесса, то M = n.

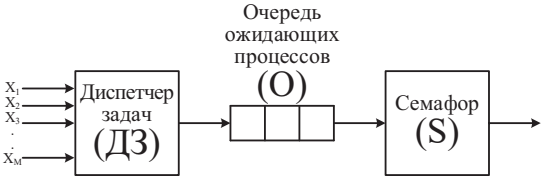


Рис. 1. Модель обслуживания на основе бесприоритетной дисциплины

Критический вопрос управления семафорами придерживается концепции: предоставление ОР в порядке поступления запросов. Чтобы решить текущую проблему, следует сократить среднее время, которое процессы тратят на выполнение действий с использованием ОР [10].

Зачастую, используя стратегию планирования «кратчайшая работа – первой», добиваются сокращения времени выполнения процессов. Для случая с семафором такая стратегия заключается в следующем: есть некие процессы, которые во время своего выполнения запрашивают, как в случае классической задачи «производители-потребители» [3], взаимодействие с семафором чаще одного раза, можно допустить нахождение в очереди ожидающих ОР процессов в любой момент времени уже обслуженных семафором процессов. В системе, удовлетворяющей такому принципу, процессы помещаются в очередь и получают

соответствующий приоритет: чем выше номер очереди, тем ниже приоритет (рис. 2). ДЗ выполняет сортировку на основе измеренного времени обработки i-го семафора. Каждый ЦП обрабатывает поток запросов X_i (i = 1,...,n), и время обработки предполагается экспоненциально распределенным. Пусть ДЗ формирует потоки запросов с интенсивностями $\lambda_1 \ldots \lambda_n$, каждый из которых имеет приоритет k_i (i = 1,…,n).

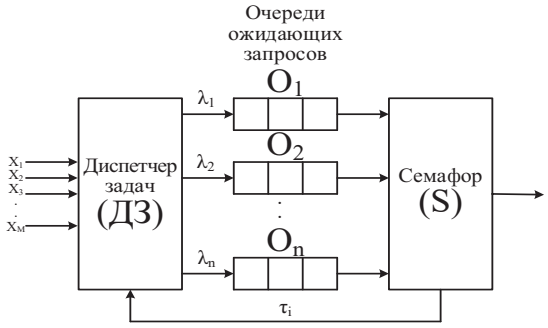


Рис. 2. Модель приоритетного обслуживания

Среднее время ожидания в очереди к семафору одной заявки k-го приоритета составит [9]:

$$t_S^k = \frac{\sum_{i=1}^{M} \rho_i \tau_i}{\left(1 - R_{k-1}\right)\left(1 - R_k\right)}, \qquad (2)$$

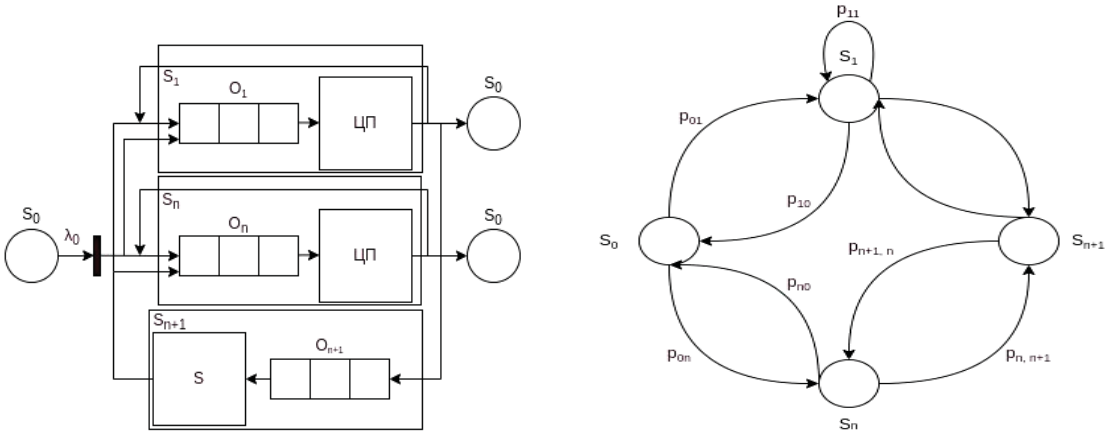
где $\rho_i = \lambda_i \tau_i$ – нагрузка на семафор, создаваемая i-м процессом, R_{k-1} – суммарная нагрузка на семафор от процессов, создаваемых потоками $\lambda_{1,\ldots,} \lambda_{k-1}$, R_k – суммарная нагрузка на семафор от процессов, создаваемых потоками $\lambda_{1,\ldots,} \lambda_k$.

Математическая модель оценки потери времени для процессов, ожидающих на семафоре. Предположим, что в системе есть ОР, защищенный семафором S, к которому могут обращаться несколько процессов. При функционировании вычислительной системы один или несколько процессов формируют свои запросы на занятие семафора S. Модель РВС для оценки производительности из-за конкурирующих процессов, обращающихся к семафору S, показана на рис. 3, а. Она представляет собой сеть, состоящую из n СМО ($S_1 \ldots S_n$), описывающих функционирование ЦП, и СМО S_{n+1}, описывающей работу семафора. Фиктивная СМО S_0 является источником задач на обработку. В РВС на выполнение поступает поток процессов с интенсивностью $\lambda_0 = 1 / t$, где t – период поступления запросов на обслуживание, который с вероятностью $p_{01,\ldots,} p_{0n}$ (рис. 3, б) распределяется по ЦП.

Процессы, циркулирующие в РВС, обращаются с запросами к семафору с некоторой интенсивностью $\lambda_i = 1 / \varsigma$, где i = 1,...,n. Предположим, вероятность распределения запросов для каждого ЦП одинакова и составляет $p_{01} = \ldots = p_{0n} = 1/n$. Каждый ЦП ($S_{1,\ldots,} S_n$) посылает запрос в семафор (S_{n+1}) с определенной вероятностью $p_{1,n+1} = \ldots = p_{n,n+1}$. Обработанные на семафоре запросы возвращаются в ЦП с вероятностью $p_{n+1,1} = \ldots = p_{n+1,n} = 1/n$.

Примем, что вероятность полностью обслуженного на узле i запроса равна p_{i0}, а вероятность того, что запрос останется обслуженным на узле S_i, равна p_{ij}, где i,j = 1,...,n.

Среднее время выполнения i-го процесса семафором должно быть аналогично выполнению узла процесса, что характеризует живой семафор [3].



а) *б)*

Рис. 3. Модельное представление (а) и граф (б) РВС с n-процессорами

Рис. 4. Время ожидания заявок в сети (а) и время ожидания в очереди к семафору (б) при использовании бесприоритетной и приоритетной стратегий планирования

С целью упрощения расчетов примем, что среднее время обслуживания в ЦП $v_i = 1$ ($i = 1,...,n$), а среднее время обслуживания в семафоре принято равным 0,8. Определим, что процесс, вызывающий семафор, выполняет в среднем 1000 циклов. В исследовании принято, что запрос покидает систему, с вероятностью $p_{i0} = 0{,}001$, следовательно, с вероятностью 0,999 запрос останется в системе. Предположим, что на каждые 8 единиц времени обработки приходится одна единица обслуживания в семафоре. Таким образом, вероятность того, что запрос останется в ЦП, и вероятность того, что он будет перенаправлен на семафор, составляет в среднем 8 единиц времени. Это означает, что каждый ЦП генерирует запрос семафора каждые 9 единиц времени ($\varsigma = 9$). Исходя из сказанного, можно получить вероятность запроса семафора некоторым процессом, выполняющимся i-м ЦП, следующим образом $p_{i,n+1} = 1/\varsigma = 0{,}111$ ($i = 1,\ldots,n$), а вероятность того, что процесс останется на обслуживании в ЦП, можно получить так: $p_{ij} = 1 - (p_{i,n+1} + p_{i0}) = 0{,}888$, где ($i,j = 1,\ldots,n$). Ожидание исполнения задачи в сети определится в соответствие с выражением

$$T_{ожс} = T_{ЦП} + T_S = \alpha_1 tw_1 + \alpha_2 tw_2 + \alpha_{n+1} tw_{n+1}, \quad (3)$$

где $T_{ЦП}$ – время ожидания ЦП, T_S – время ожидания семафора; $\alpha_i = \lambda_i / \lambda_0$ – коэффициент передачи ($i = 1,\ldots,n+1$); tw_i – время ожидания в i-й СМО, $tw_i = \rho_i v_i / (1 - \rho_i)$ [11] ($i=1,\ldots,n$); tw_{n+1} – время ожидания в очереди к семафору.

Интенсивности поступающих входных потоков задач оценим выражением

$$\lambda_i = \sum p_{ji} \lambda_j, \quad (4)$$

где p_{ji} – вероятность трансляции задачи из СМО S_j в S_i, причем $i, j = 1,\ldots,n+1$.

Зададим интенсивность $\lambda_0 = 0{,}008$ заявок в единицу времени, пусть запросы на вход поступают с заданной интенсивностью. Далее построим график зависимости времени ожидания заявок в сети от числа потоков, обслуживаемых семафором. Параметр n в таком случае варьируется от 6 до 16 ЦП, а длительность обработки запроса семафором при постоянном среднем времени обработки запроса варьируется от 0,4 до 1,4 единиц. В итоге получим, что самый короткий процесс обрабатывается в семафоре за 0,4 единицы времени, а самый длинный – за 1,4 единицы времени.

Результаты расчетов приведены на графиках (рис. 4).

Из графиков видно, что применение приоритетного планирования дает временной выигрыш по ожиданию в очереди к семафору приблизительно на 30–38 % в зависимости от процессоров, в отличие от бесприоритетного планирования.

Заключение

Оценив результаты, приходим к выводу, что синтезированные и исследованные модели позволяют проводить количественные оценки времени ожидания исполнения процессов. Исследуемые в статье модели возможно использовать на практике при разработке новых типов реконфигурируемых вычислительных систем. Практические результаты, полученные в работе, возможно применить для разработки и новых ОС для РВС, где основополагающим является время исполнения процессов: чем оно меньше, тем производительнее будет ОС. Также по проведенному исследованию построены графики, визуально отражающие основные полученные результаты расчетов моделей. Анализируя полученные графики, выявлено, что приоритетное планирование дает возможность уменьшения времени ожидания в очереди к семафору выполняющихся процессов приблизительно на 30–38 %.

Исследование выполнено за счет гранта Российского научного фонда № 21-71-00110, https://rscf.ru/project/21-71-00110/.

Список литературы

1. Столлингс В. Операционные системы. М.: Вильямс, 2013. 848 с.

2. Лав Р. Ядро Linux: описание процесса разработки, 3-е изд.: пер. с англ. М.: Вильямс, 2013. 496 с.

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

4. Dijkstra E.W. Cooperating sequential processes. Technological University Eindhoven. The Netherlands. September 1965. Academic Press. New York. P. 43–112.

5. Лав Р. Разработка ядра Linux. М.: Вильямс, 2013. 448 с.

6. Уорд Б. Внутреннее устройство Linux. СПб.: Питер, 2016. 384 с.

7. Tallent N.R., Mellor-Crummey J.M., Porterfield A. Analyzing lock contention in multithreaded applications. ACM SIGPLAN Notices, 2010. Vol. 45. No. 5. P. 269–279. DOI: 10.1145/1837853. 1693489.

8. Столлингс В. Операционные системы. Внутренняя структура и принципы проектирования. 9-е изд.: пер. с англ. М.: Вильямс, 2020. 1264 с.

9. Алиев Т.И. Основы моделирования дискретных систем. СПб.: СПбГУ ИТМО, 2009. 363 с.

10. Мартышкин А.И. Математическое моделирование диспетчеров задач в многопроцессорных вычислительных системах на основе стохастических сетей массового обслуживания: автореф. дис. … канд. техн. наук. Пенза, 2013. 23 с.

11. Карасева Е.А. Исследование процесса управления множеством критических ресурсов в многопроцессорных системах // Информационные технологии. Радиоэлектроника. Телекоммуникации. 2016. № 6–1. С. 298–304.