УДК 004.451

МОДУЛЬ ПРИНУДИТЕЛЬНОЙ ДИСПЕТЧЕРИЗАЦИИ ПРОЦЕССОВ ДЛЯ ОПЕРАЦИОННЫХ СИСТЕМ ОБЩЕГО НАЗНАЧЕНИЯ

А.И. Мартышкин
кандидат технических наук, доцент, доцент кафедры вычислительных машин и систем, ФГБОУ ВО «Пензенский государственный технологический университет», г. Пенза, Россия, e-mail: Alexey314@yandex.ru

Аннотация. В статье после проведения обзора существующих подходов и методов принято решение о подходе, используемом для решения рассматриваемой проблемы. Был выбран и реализован в виде модуля способ выделения ресурсов, основанный на общих для рассмотренных систем механизмов. Модуль запускает вычисления в конфигурации, соответствующей топологии системы, задает политику планирования реального времени для потоков, назначает их на доступные вычислительные ядра, самостоятельно диспетчеризует исполнение задач.

Ключевые слова: операционная система общего назначения, алгоритмы планирования, диспетчеризация процессов, вычислительный ресурс, планировщик, приоритет.

MODULE FOR FORCED PROCESS DISPATCHING FOR GENERAL-PURPOSE OPERATING SYSTEMS

A.I. Martyshkin
Ph.D., Associate Professor, Associate Professor of the Department of Computers and Systems, FGBOU VO 'Penza State Technological University', Penza, Russia, e-mail: Alexey314@yandex.ru

Abstract. In the article, after reviewing the existing approaches and methods, a decision was made on the approach used to solve the problem under consideration. A resource allocation method based on the mechanisms common to the systems considered was selected and implemented as a module. The module runs calculations in a configuration that corresponds to the system topology, sets a real-time scheduling policy for threads, assigns them to available computing cores, and independently dispatches task execution.

Keywords: General-purpose operating system, scheduling algorithms, process dispatching, computing resource, scheduler, priority.

Введение. Не всегда есть возможность оптимизировать сами вычисления [1]. Задача в процессе выполнения проходит несколько этапов. В том числе инициализацию, ожидание процессора, ожидание других ресурсов, выполнение на процессоре, завершение.

Инициализация включает создание необходимых ядру операционной системы (ОС) структур данных, загрузку образа программы в основную память, открытие необходимых файлов, выделение других ресурсов, первичное планирование исполнения на процессоре.

Ожидание процессора в очереди готовых задач является нормой для многозадачных систем с разделением времени. Как правило, на процессоре выполняется приоритетная задача, в то время как остальные ожидают в очереди. Ядро ОС оперирует динамическими приоритетами, не позволяя никакой задаче монополизировать процессор [2].

Выполнение задачи на процессоре в какой-то момент времени означает исполнение ее кода в этот момент процессором. Когда для дальнейшего исполнения задачи необходимо наступление некоторого события, задача встает в очередь, ожидающих этого события. Таким событием может быть окончание ввода-вывода без участия ЦП, освобождение системного ресурса (например, примитива синхронизации) и др. Время ожидания ввода-вывода зависит от нагрузки и пропускной способности подсистемы ввода-вывода. Время выполнения на процессоре зависит от оптимизации программного кода. Время ожидания в очереди готовых задач зависит от их количества в системе, их вычислительной мощности, соотношения приоритетов и конфигурации планировщика/диспетчера ОС. Время инициализации практически полностью зависит от реализации ОС [2].

Цель работы. Цель работы состоит в максимизации эксклюзивного использования процессорных ресурсов в ОС общего назначения. Поставленная задача: разработать программный модуль для запуска вычислений в приоритетном режиме, минимизации накладных расходов на планирование и диспетчеризацию.

Материал и результаты исследований. Для организации вычислительного процесса выбрана схема, использующая общие принципы, реализованные в ОС FreeBSD 10, GNU\Linux 4, Oracle Solaris 11, Microsoft Windows 7/10.

Для максимизации времени вычислительных задач на процессоре и минимизации количества переключений контекста выбрана стратегия планирования SCHED_FIFO. Это позволяет закрепить на процессоре необходимое число потоков, не вытесняемых по таймеру. Задачи реализуются в качестве функций, содержащихся в очереди в контексте приложения. Диспетчеризация задач состоит в сохранении результата, возвращаемого функцией, чтении и сдвиге указателя, вызове новой функции. Для исключения возможных миграций потоков между процессорами применяется явное указание на каком процессоре выполняться. Также запрещены программные прерывания.

При старте приложения определяется топология системы, количество процессоров, физических и логических ядер. В Linux эта информация извлекается

из унифицированной виртуальной файловой системы sysfs, служащей интерфейсом к структурам ядра. Виртуальный каталог /sys/devices/system/cpu/ содержит подкаталоги вида cpuN с файлами, описывающими топологию и состояние соответствующего логического ядра с номером N.

```
int topology (int ** cpus)
{
    struct dirent ** cpusdir; int c, count;
    count = scandir (SYSFS_CPU, &cpusdir, cpu_dir_filter, alphasort);
    if ((*cpus = malloc (sizeof(int) * count)) == NULL)
    {
        err (EXIT_FAILURE, "can't allocate memory for " \ "cpus array");
    }
    for (c=0; c < count; ++c)
    {
        assert (1 == sscanf (cpusdir[c]->d_name, "cpu%d", &(*cpus)[c]));
    }
    return count;
}
```

После определения топологии принимается решение за какими вычислительными ядрами следует закрепить вычислительные потоки.

```
cpu_set_t set;
/* ... */
if (tharg -> cpu >= 0) /* if < 0, do not bind */
{
    CPU_SET (tharg -> cpu, &set);
    if ( -1 == sched_setaffinity ( 0, sizeof(set), &set ) )
    {
        err (EXIT_FAILURE, "Can't set affinity mask");
    }
}
```

Для организации вычислений принято решение разработать модуль, подготавливающий потоки для вычислений и запускающий задачи. Каждый поток выполняется на отдельном ядре / процессоре, их количество равно количеству ядер минус один. Одно ядро выделяется под нужды различных системных и пользовательских задач. Таким образом достигается максимальная степень параллелизма, при сохранении отзывчивости системы. Так как стратегия разделения процессорного времени не позволяет монополизировать ядра для полезных вычислений, было принято решение счетные задачи перевести в класс realtime – наиболее приоритетный в системе. Realtime задачи получают максимальный приоритет, не меняющийся в зависимости от времени выполнения на процессоре, и никогда не вытесняются низкоприоритетными задачами. Счетные задачи получают максимум процессорного времени.

Выбран алгоритм диспетчеризации задач на процессор FIFO так как этот алгоритм влечет минимум накладных расходов и прост в реализации. Кванто-

вание времени не используется, задача не снимается с процессора принудительно, пока не будет выполнена до конца. Нет лишних переключений контекста, кэши “разогреты” подавляющую часть времени и работают эффективно.

Обработку прерываний возможно назначить на конкретное ядро, для этого используется интерфейс виртуальной файловой системы proc. Файл /proc/irq/M/smp_affinity (где M - номер прерывания) содержит маску, каждый единичный бит которой означает разрешение на обработку ядром с соответствующим номером. Разработанный модуль при старте осуществляет запись маски, соответствующей свободному ядру. Например, ядру с индексом 3 соответствует маска 8.

Прикладная очередь содержит указатели на функцию и передаваемый ей аргумент. Счетные потоки самостоятельно осуществляют выборку задачи, копируя указатели на функцию и аргумент в свой локальный контекст и сдвигая указатель текущего элемента очереди. Для совместного доступа используется примитив синхронизации мьютекс.

Измерения производились на основе счетчиков, встроенных в центральный процессор и ядро операционной системы Linux (начиная с версии 2.6.31) утилитой perf [17]. Измерялось количество переключений контекста, TLB-промахов, кэш-промахов первого и последнего уровня, количество миграций с процессора на процессор, суммарное время выполнения.

Тестовый запуск и замеры проводились на компьютере с процессором Intel ® Core™ i5 6200U, под управлением операционной системой GNU/Linux (архитектура x64, версия ядра 4.4.0-79-generic). Подробное описание процессора:

- Количество физических ядер: 2.
- Количество логических ядер: 2.
- Тактовая частота 2.30 ГГц.
- Кэш инструкций L1: 32 КБ на ядро.
- Кэш данных L1: 32 КБ на ядро.
- Кэш L2: 256 КБ на ядро.
- Кэш L3: 3 МБ на процессор.

Количество основной памяти, выделенное ядру операционной системы, составляет 2ГБ.

В качестве полезной нагрузки была выбрана задача, реализующая циклический обход массива с изменяющимся шагом и наращиванием значений элементов. Размер массива составляет 5 МБ. Внешняя нагрузка моделируется четырьмя процессами, активно выполняющими операции с памятью (для чего они выделяют суммарно 1,5 ГБ памяти).

Время вычислений (time) с предлагаемым модулем практически не зависит от нагрузки на систему, создаваемой сторонними программами. Сведены к минимуму такие негативные эффекты, как миграция на другое вычислительное

ядро (cpu-migration) во время выполнения, переключения программного контекста (context-switches). Количество промахов TLB (dTLB-load-misses, dTLB-store-misses) снижено в среднем до 1,45% от изначального количества.

Показатели кэш-промахов (cache-misses) практически не изменяются на протяжении эксперимента, что, по-видимому, связано с особенностью реализации тестовых задач: алгоритм циклического обхода массива в несколько мегабайт, с шагом значительно больше единицы, не позволяет разместить этот массив в кэше процессора.

Выводы. Действенность описываемого модуля подтверждена тестовыми запусками и измерениями таких показателей как время выполнения, количество переключений программного контекста и обращений во внешнюю память.

Работа выполнена при финансовой поддержке стипендии Президента РФ молодым ученым и аспирантам на 2018-2020 гг. (СП-68.2018.5).

ЛИТЕРАТУРА

1. Таненбаум Э. С., Херберт Б. Современные операционные системы. 4-е изд. – " Издательский дом "Питер" ", 2015.
2. Silberschatz A. et al. Operating system concepts. – Reading : Addison- wesley, 1998. – Т. 4.

УДК 303.732

ОПИСАНИЕ ПОНЯТИЯ И СТРУКТУРЫ СИСТЕМЫ И ЕЕ КОМПОНЕНТОВ

А.И. Мартышкин[1], Д.Э. Ильичов[2]

[1]кандидат технических наук, доцент, доцент кафедры вычислительных машин и систем, ФГБОУ ВО «Пензенский государственный технологический университет», г. Пенза, Россия, e-mail: Alexey314@yandex.ru

[2]студент группы 17ИВ1бп, Факультет автоматизированных информационных технологий, ФГБОУ ВО «Пензенский государственный технологический университет», г. Пенза, Россия, e-mail: biberlink@mail.ru

Аннотация. В статье определены понятия системы, описывается структура системы за счет определения ее компонентов. Приведены различные классификации систем. Отдельно рассмотрены недетерминированные системы, их сравнение с детерминированными. Раскрыта специфика исследования и моделирования недетерминированных систем.

Ключевые слова: система, исследование, классификация, элементы, внешняя среда, взаимодействие.