спотворення прототипу кожного конкретного предметної середовища. Але сучасна дійсність така, що кожна проектна ситуація – це нова проблема не схожа на іншу. І кожен раз потрібно розробляти цілісну проектно-художню модель проблемної ситуації, що майже завжди пов'язано з виявленням цілісних ансамблів предметного світу.

ЛІТЕРАТУРА

1. Мереняшева М.А. Метод художественно-композиционного моделирования в обучении профессии дизайнера / М.А. Мереняшева // Перспективы науки и образования 2014. – № 2.

2. Мелодинский Д.Л. Школа архитектурно-дизайнерского формообразования – М.: Архитектура, 2004. – 312 с.

3. Логінова А.О., Логвіненко С.О. Про оцінку ефективності дизайнерських рішень // Науково-технічна конференція студентів, аспірантів і молодих вчених «Наукова весна 2019». – Дніпро: НТУ «ДП». – 2019.

4. Логінова А.О., Вернер І.В., Кожушкіна Т.Л. Морфологічна трансформація як засіб дизайну // Сборник научных трудов международной конференции «Современные инновационные технологии подготовки инженерных кадров для горной промышленности и транспорта 2019». – Днепр: НТУ «ДП», 2019. – С. 201-205.

5. Логінова А.О., Литовченко А.А., Щелкунов А.Ю. Економічна ефективність художньо-конструкторських розробок технологічного обладнання // Сборник научных трудов международной конференции «Современные инновационные технологии подготовки инженерных кадров для горной промышленности и транспорта 2019». – Днепр: НТУ «ДП», 2019. – С. 205-211.

УДК 004.451

ОБЗОР И АНАЛИЗ СЕМЕЙСТВ ОПЕРАЦИОННЫХ СИСТЕМ ОБЩЕГО НАЗНАЧЕНИЯ И ИХ ВОЗМОЖНОСТЕЙ

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 работе проанализировано четыре семейства операционных систем общего назначения и их возможности. По результатам анализа были составлены краткие характеристики используемых алгоритмов планирования и диспетчеризации задач.

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

OVERVIEW AND ANALYSIS OF GENERAL-PURPOSE OPERATING SYSTEM FAMILIES AND THEIR CAPABILITIES

A.I. Martyshkin[1], D.E. Ilyichov[2]

[1]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

[2]Student of group 17iv1bp, Faculty of automated information technologies, FGBOU VO "Penza State Technological University", Penza, Russia, e-mail: biberlink@mail.ru

Abstract. The paper analyzes four families of General-purpose operating systems and their capabilities. Based on the results of the analysis, brief characteristics of the used algorithms for planning and dispatching tasks were compiled.

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

Введение. В современном мире на повестке дня довольно остро стоит вопрос производительности вычислений. Если для обычного пользователя ПК оптимизация выполняемой им задачи, сократит время её выполнения с одной минуты до 59 секунд, разницу он скорее всего не заметит. Иначе дело обстоит с большими объёмами вычислений [1]. Уменьшение времени выполнения многократно повторяемой задачи с 60 до 59 секунд позволяет выполнить весь объём работ на 1,67% быстрее или выполнить на 1,67% больший объём работ за то же время. Таким образом можно сэкономить около шести суток вычислений в год.

Цель работы. В статье рассматриваются проблемы, связанные с планированием и диспетчеризацией задач на исполнение в операционных системах (ОС). Цель работы состоит в максимизации эксклюзивного использования процессорных ресурсов ОС общего назначения.

Материал и результаты исследований. ОС общего назначения применяются в персональных компьютерах и серверных системах, как правило для работы с данными (вычисления, обработка, хранение). Основным требованием являются широкие функциональные возможности для работы с различными пакетами программ [2].

Низкая стоимость и широкий спектр решаемых задач поспособствовали широкому распространению персональных компьютеров, а вместе с ними и ОС разделения времени.

Системы такого типа предназначены решать следующие проблемы:

- обеспечивать загрузку компонентов компьютера;
- гарантировать каждой задаче возможность выполнения на процессоре;
- предоставлять задачам объём вычислительных ресурсов в зависимости от их приоритета.

Системы разделения времени используют квантование времени исполнения задач на процессоре. В общем случае, по истечению кванта, задача снимается с процессора, на выполнение выбирается следующая задача. Детали зависят от используемого алгоритма планирования задач на выполнение.

В контексте данной работы имеет смысл обозначить алгоритмы планирования, имеющие реализацию, самостоятельную или в составе более сложного алгоритма:

Первым пришел первым обслужен (First In First Out, FIFO) – выполнение в порядке поступления [3]. Первая готовая задача выполняется на процессоре, каждая следующая встаёт в очередь и ожидает завершения всех предстоящих. Гарантирует исполнение на процессоре в случае, если ни одна из задач не займёт его навечно. Этот алгоритм не относится к разделению времени, так как задачи выполняются последовательно без прерываний.

Циклический (Round Robin, RR) — задачи выполняются циклически фиксированные промежутки времени [4]. По окончанию очередного промежутка, задача встаёт в конец очереди. Для каждой задачи этот процесс повторяется пока она не завершится. Циклический алгоритм гарантирует каждой задаче 1/N долю времени на процессоре, где N – общее число задач.

С наименьшим временем исполнения вперёд - на выполнение каждый раз выбирается задача с наименьшим временем, необходимым на её исполнение [4]. Для работы алгоритма требуется знание о времени, которое задаче потребуется.

Вытесняющий алгоритм со статическими приоритетами (Fixed priority preemptive) — на процессоре в любой момент времени выполняется приоритетная задача [5]. При появлении задачи с наивысшим приоритетом, выполняющаяся задача прерывается и встаёт в очередь, освобождая процесс приоритетной. Выполнение низкоприоритетных задач не гарантируется.

Алгоритм планирования по ближайшему крайнему сроку (Earliest Deadline First) — для каждой задачи должен быть определён крайний срок её выполнения, превышение которого лишает смысла дальнейшее её выполнение [6].

Приоритетная очередь — очередь, в которой порядок обработки задачи не зависит от времени её поступления [4]. На выполнение каждый раз выбирается первая задача с наивысшим приоритетом из всей очереди. При

этом возможны две реализации: вытесняющий алгоритм, когда поступающая в очередь задача с приоритетом, большим чем обрабатываемая, вытесняет последнюю с процессора и не вытесняющий алгоритм, когда задача новая задача с наивысшим приоритетом дожидается завершения задачи, которая уже на процессоре.

Многоуровневая очередь (Multilevel Queue) – задачи помещаются в одну из очередей, отличающихся приоритетами, и закрепляется за ней до завершения [5]. На выполнение выбирается задача из приоритетной очереди. Внутри очередей могут реализовываться различные алгоритмы планирования. Например, могут использоваться две очереди для разных классов задач, очередь с первым приоритетом, работающая по принципу FCFS, очередь с вторым приоритетом организована внутри по принципам приоритетной. Задачи первой очереди могут вытеснять с процессора задачи второй очереди. Таким образом возможно организовать совместную работу двух классов задач: обычного приоритета и повышенного, вытесняющего. Multilevel feedback queue – в отличие от многоуровневой очереди, задачи могут перемещаться из одной очереди в другую, обычно после исчерпания своей квоты на процессоре [4].

Накладные расходы складываются из времени, необходимого на планирование, диспетчеризацию задач и сопутствующих им процессов. Время работы планировщика зависит от вычислительной сложности его реализации и, в общем случае, количества задач в системе. Диспетчеризация, как правило, хорошо оптимизирована и сама по себе занимает крайне мало времени, однако сопутствующее переключение контекста влечёт за собой охлаждение кэшей, инвалидацию TLB и, в худшем случае, свопинг. Охлаждение кэшей означает потерю актуальности записей в кэш-памяти и частые кэш-промахи до его прогрева. За кэш-промахом следует обращение к нижестоящему более медленному уровню памяти. TLB содержит адреса виртуальных страниц процесса. Инвалидация TLB выливается в удвоенное число обращений к основной памяти компьютера [7]. Свопинг страниц процесса на диск является способом борьбы с нехваткой памяти. Ненужные в настоящий момент страницы процесса выгружаются на диск, а необходимые загружаются в основную память. Происходит активный обмен с самой медлительной памятью в ПК. При активном свопинге, производительность вычислений снижается многократно [2].

Для рассмотрения функциональных возможностей и используемых алгоритмов планирования/диспетчеризации задач выбраны следующие ОС: FreeBSD 10; GNU\Linux 4; Oracle Solaris 11; Microsoft Windows 7/10.

Выбор ОС для анализа продиктован их распространенностью для всевозможных вычислений, а также желанием охватить различные семейства

ОС с целью выявления наиболее подходящих реализаций требуемого функционала.

Для обозначения самостоятельной единицы диспетчеризации в статье используется понятие “задача”. В терминологии ОС Windows, Solaris и FreeBSD процессом является системная сущность, содержащая необходимое для выполнения программы окружение, в том числе программный код. Потоком называется сущность, хранящая состояние исполнения последовательности инструкций в контексте процесса. У процесса может быть один или несколько потоков [8, 9, 10]. В Linux все потоки фактически являются процессами, разделяющими общие системные ресурсы [11]. Отсутствие необходимости выделять эти ресурсы сокращает расходы на создание потока. В результате получается сущность, диспетчеризуемая как потоки в других ОС. Таким образом, все рассматриваемые ОС в качестве минимальной единицы диспетчеризации используют поток (thread).

Планировщик ядра FreeBSD использует несколько приоритетных очередей выполнения (runqueues, рис. 1а): itqueues, rtqueues, queues, idqueues – в порядке снижения приоритета [12].

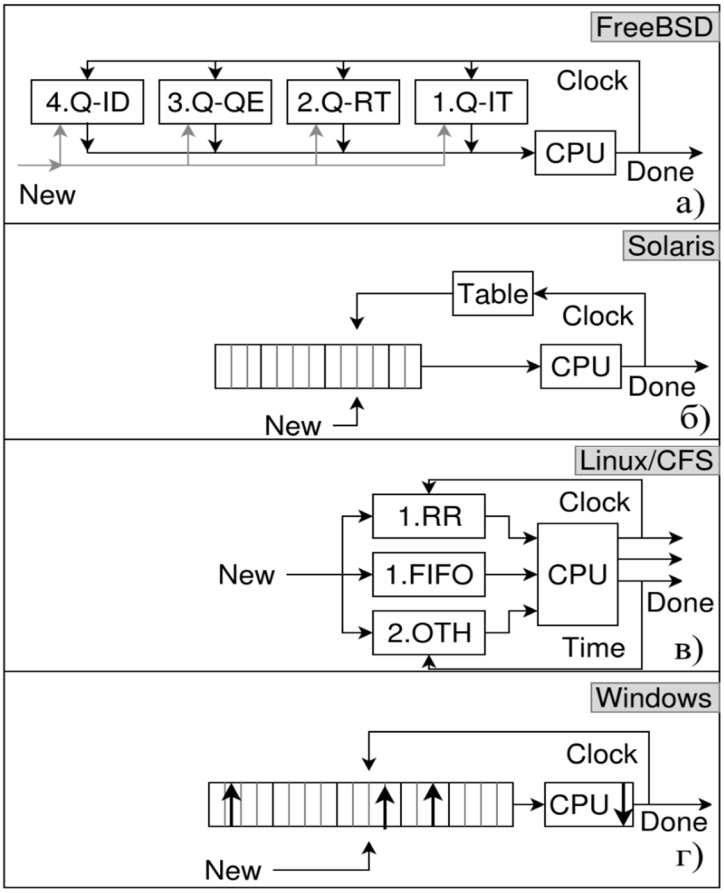

Рисунок 1 – Планировщики операционных систем

Для выполнения на процессоре выбирается приоритетная задача из старшей непустой очереди. Задача принудительно снимается с процессора по истечении выделенного ей кванта времени, её динамический приоритет пересчитывается. Задача помещается в конец списка равноприоритетных задач. Задача, заблокировавшая некоторый общий ресурс, получает приоритет задачи, ожидающей этот ресурс, если он выше.

Планировщик ОС Solaris использует количество списков по количеству приоритетов и таблицу их преобразования (рис. 1б). Приоритет задачи, израсходовавшей свой квант времени, снижается, однако вместе с тем, увеличивается следующий её квант. Наоборот, задача, освободившая процессор до исчерпания кванта, получает повышенный приоритет и меньший квант. Таким образом, время ожидания небольших периодических задач снижается, они быстрее попадают на процессор, а большие вычислительные задачи исполняются более эффективно, потому что реже снимаются с процессора.

Задачи, блокирующие системные ресурсы, получают повышенный приоритет, что снижает время исполнения таких задач, приближает время освобождения ресурса и уменьшает время простоя ожидающих ресурс задач.

Планировщик, находящийся в основной ветке разработки ядра Linux - CFS, использует несколько различных структур данных (рис. 1в) [13]. Задачи класса Deadline имеют чётко обозначенную продолжительность исполнения и крайний срок исполнения. Они планируются на процессор в соответствии с алгоритмом Earliest Deadline First.

Задачи реального времени имеют статические приоритеты и диспетчеризуются по одному из двух алгоритмов: FIFO или Round Robin. Во втором случае используются фиксированные кванты времени, после окончания которого, задача помещается в конец списка для своего приоритета. В первом случае, приоритетная задача исполняется целиком, не прерываясь.

Для обычных задач CFS реализует алгоритм взвешенной справедливой очереди. Назначаясь на процессор, задача получает промежуток времени на исполнение, пропорциональный времени ожидания в очереди и обратно пропорциональный числу готовых к исполнению задач.

Планировщик задач ОС семейства Windows использует несколько классов и уровней приоритетов: низкий, средний, высокий и по два промежуточных (см. рис 1г). Для одной задачи программным способом назначается базовый приоритет. На выполнение всегда выбирается наиболее приоритетная задача. Планировщик ядра, может увеличивать её приоритет в одном из четырёх случаев:

- появляется событие, такое как пользовательский ввод, ожидающее обработки задачей;

- задача выполняется в интерактивном режиме на переднем плане;
- происходит разблокировка задачи;
- периодически случайным образом повышается приоритет готовых к выполнению задач.

После исчерпания очередного кванта времени на процессоре, динамический приоритет задачи снижается на единицу, но не ниже базового. Таким образом, планировщик борется с эффектом “голодания”, повышая приоритеты ожидающих задач и приводя их в норму, по мере снижения этого эффекта. Также поощряются интерактивные процессы.

ОС FreeBSD, Linux и Solaris обеспечивают поддержку Pthreads – POSIX совместимой библиотеки, предоставляющей единый интерфейс для работы с многопоточными приложениями.

Solaris также предоставляет собственный интерфейс Solaris Threads, реализованный ещё до Pthreads. Solaris threads предоставляют такие возможности как управление уровнем параллелизма, усыпление/пробуждение потоков, механизм блокировок по чтению/записи, возможность создать поток-демон. В противовес они не поддерживают отмену потоков и стратегии планирования POSIX.

Windows имеет собственную реализацию потоков - Windows Threads. Интерфейс имеет множество функциональных возможностей, не предусмотренных в POSIX, таких как: планирование в контексте пользователя, волокна (fiber), вручную диспетчеризуемые в контексте потока, критические секции в качестве примитива синхронизации.

В таблице 1 приведены краткие сведения об интерфейсах работы с потоками, планировщиками и их конфигурации.

Таблица 1 – Интерфейсы операционных систем

Семейство ОС	FreeBSD	GNU/Linux	Solaris	Windows
Управление потоками	POSIX.1 pthreads			Windows Threads
Интерфейс планировщика	sched_setscheduler(SCHED_FIFO)			SetPriorityClass(REAL TIME_PRIORITY_ CLASS)
Привязка к вычислительному ядру	pthread_ setaffinity_ np	sched_ setaffinity	processor_ bind	SetThreadAffinityMask

Выводы. Как видно из таблицы, ОС FreeBSD, Linux, Solaris поддерживают интерфейсы, стандартизированные POSIX [14], а потому код, реализующий работу через эти интерфейсы, будет легко переносим между системами. Стандартом POSIX не описан функционал привязки потока к вычислительному

ядру, однако соответствующие возможности есть во всех рассмотренных ОС, хотя совместимость отсутствует.

ЛИТЕРАТУРА

1. Филиппенко П. Н., Шашелов А. А., Сеитова С. В. Построение систем, обрабатывающих большие вычисления: проблемы и тенденции // Известия Южного федерального университета. Технические науки. – 2010. – Т. 113. – №. 12.

2. Таненбаум Э. С., Херберт Б. Современные операционные системы. 4-е изд. – " Издательский дом "Питер" ", 2015.

3. Zhao W., Stankovic J. A. Performance analysis of FCFS and improved FCFS scheduling algorithms for dynamic real-time computer systems //Real Time Systems Symposium, 1989., Proceedings. – IEEE, 1989. – С. 156-165.

4. Silberschatz A. et al. Operating system concepts. – Reading : Addison- wesley, 1998. – Т. 4.

5. Unice W. K. Deterministic preemption points in operating system execution: пат. 6802024 США. – 2004.

6. Chiussi F. M., Sivaraman V. Guaranteeing data transfer delays in data packet networks using earliest deadline first packet schedulers: пат. 6532213 США. – 2003.

7. Chen J. B., Bershad B. N. The impact of operating system structure on memory system performance //ACM SIGOPS Operating Systems Review. – ACM, 1994. – Т. 27. – №. 5. – С. 120-133.

8. McDougall R., Mauro J. Solaris internals: Solaris 10 and OpenSolaris kernel architecture. – Pearson Education, 2006.

9. Baldwin J. H. Locking in the Multithreaded FreeBSD Kernel //BSDCon. – 2002. – С. 27-35.

10. Cohen A., Woodring M. Win32 Multithreaded Programming. – Oreilly & Associates Incorporated, 1998.

11. Drepper U., Molnar I. The native POSIX thread library for Linux //White Paper, Red Hat Inc. – 2003.

12. McKusick, Marshall Kirk, Keith Bostic, Michael J Karels, and John Quarterman. The Design and Implementation of the 4.4BSD Operating System. Reading, Mass.: Addison-Wesley, 1996.

13. Бовет Д., Чезати М. Ядро LINUX, 3-е изд. – БХВ-Петербург, 2007.

14. Butenhof D. R. Programming with POSIX threads. – Addison-Wesley rofessional, 1997.