**Планировщик и диспетчер процессов в системе разделения времени**

1 Введение

1.1 Когда компьютер работает в многозадачном режиме, на нем могут быть активными (находится в состоянии готовности) несколько процессов (от двух и более), пытающихся одновременно получить доступ к одному процессору. Поэтому необходимо выбирать, какой процесс запустить следующим. Отвечающая за это часть Операционной системы (ОС) называется планировщиком, а используемый алгоритм – алгоритмом планирования. Помимо правильного выбора следующего процесса, планировщик также должен заботится об эффективном использовании процессора, поскольку переключение между процессами требует затрат.

1.2 В многозадачном режиме процессы могут находиться в одном из трех основных состояний: исполнение, готовность или ожидание. Граф изменения состояний процессов проиллюстрирован на рисунке 1.1. В состоянии исполнения может находиться только один процесс. В состоянии готовности могут находиться несколько процессов. Для оперативной выборки процессов на исполнение ОС всегда поддерживает двусвязный список готовых процессов. В данном списке всегда находится хотя бы один элемент (процесс, запускаемый в случае «простаивания» системы). В состоянии ожидания также могут находиться несколько процессов. Для организации ожидающих процессов также используются двусвязные списки. Но, в отличие от списка готовых процессов, для ожидающих процессов используется один список для каждого конкретного ресурса. Сколько разделяемых ресурсов, столько и списков заблокированных процессов.

1

2

3

4

Рисунок 1.1 – Граф изменения состояния процессов

На рисунке 1.1 номерами обозначены ситуации, когда происходит данный переход: 1 – планировщик выбирает данный процесс из списка готовых процессов, 2 – квант данного процесса истек и планировщик выбирает другой процесс, 3 – процесс блокируется, ожидая входных данных, 4 – поступили ожидаемые входные данные.

2 Алгоритм Round Robin

2.1 Одним из наиболее старых, простых, справедливых и часто используемых алгоритмов планирования является алгоритм циклического планирования или Round Robin (RR). Он работает по следующему принципу. Каждому процессу предоставляется некоторый интервал времени процессора (квант времени). Если к концу кванта времени процесс все еще работает, он прерывается, а управление передается другому процессу. Если процесс блокируется или прекращает работу раньше отведенного ему кванта времени, то переход управления происходит в этот момент. Алгоритм работы проиллюстрирован на рисунке 2.1. Так как используется приоритетное планирование, то сначала из списка готовых процессов выбирается процесс с наивысшим приоритетом. Если в списке остались только процессы с одинаковым приоритетом, то выбирается самый первый. После того, как новый процесс попадает в очередь готовых процессов, он помещается в конец очереди. Когда процесс отработал свой квант или вышел из состояния блокировки, он также помещается в конец очереди.

А

Е

Б

В

Г

Д

0

2

2

3

1

0

Направление обхода списка процессов

Текущий
(работающий)
процесс

Приоритет
процесса

Г

А

Б

В

Д

Е

3

0

2

2

1

0

Направление обхода списка процессов

Текущий
(работающий)
процесс

Приоритет
процесса

Следующий
выбираемый
процесс

Следующий
выбираемый
процесс

t

Рисунок 2.1 – Схема работы алгоритма RR

Самым важным атрибутом данного алгоритма является длина кванта времени, отводимого процессу. Слишком малый квант времени приведет к частому переключению процессов и небольшой эффективности. Слишком большой квант времени может привести к медленному реагированию на короткие интерактивные запросы. «Значение кванта времени около 20-50 мс часто является разумным компромиссом [1].»

**3 Перепланировка процессов**

3.1 Перепланировка - это процесс выбора следующего запускаемого процесса и переключение на него. Перепланировка должна происходить только в строго определенные моменты времени. Пример выбора моментов перепланировки в ОС с относительным приоритетом и фиксированным квантом времени представлен в таблице 3.1.

Таблица 3.1 – Моменты перепланировки

|  |  |
| --- | --- |
| № п/п | Момент перепланировки |
| 1 | Завершение процессом своей работы (системные вызовы exit(), cancel() и др.). |
| 2 | Блокирование процесса на системном вызове (операции ВВ, семафоре, в ожидании сигнала и т.д.). |
| 3 | Добровольное блокирование процесса (системный вызов wait(), sleep() и др.). |
| 4 | Истечение кванта времени, отведенного процессу. |

Этапы переключения между процессами представлены в таблице 3.2.

Таблица 3.2 – Этапы переключения между процессами

|  |  |
| --- | --- |
| № этапа | Описание этапа |
| 1 | Переключиться из режима пользователяв режим ядра (через прерывание). |
| 2 | Сохранить контекст текущего процесса. |
| 3 | Сохранить карту памяти (биты-признаки обращения к страницам памяти) текущего процесса. |
| 4 | Запустить планировщик для выбора нового процесса. |
| 5 | Загрузить контекст нового процесса. |
| 6 | Загрузить карту памяти нового процессав блок управления памятью. |
| 7 | Запустить новый процесс. |

**4 Дескриптор и контекст процесса**

4.1 Для обеспечения многозадачности в ОС используется таблица процессов (список), в которой находятся указатели на дескрипторы процессов. Дескрипторы содержат статическую информацию о процессе. Кроме этого в ОС также используется динамическая информация о текущем (работающем) процессе. Совокупность этой информации называется контекстом процесса. Примеры дескриптора и контекста процесса представлены в таблицах 4.1 и 4.2 соответственно.

Таблица 4.1 – Дескриптор процесса

|  |  |  |
| --- | --- | --- |
| Названиеполя | Описание | Диапазондопустимыхзначений |
| Идентификатор процесса | Число, однозначно идентифицирующее процесс в ОС.В системе не должно быть процессов с одинаковыми идентификаторами. | 0 - 216 |
| Оставшийся квант времени | Назначается ОС при создании процесса. С каждым прерыванием от таймера данное значение уменьшается на едини цу (у активного процесса). | 0 - 255 |
| Названиеполя | Описание | Диапазон допустимых значений |
|  | Когда значение достигнет 0, процесс переносится в конец очереди готовых процессов. |  |
| Приоритет процесса | Условное значение, по которому планировщик определяет, какой процесс выбрать на обслуживание. От 0 до 4 – приоритеты, назначаемые ядром ОС для своих нужд. От 5 до 9 – приоритеты, назначаемые пользователем для своих нужд. 0 – самый высокий приоритет, 9 – самый низкий приоритет. | 0 - 9 |
| Базовый адрес контекста процесса | Содержимое регистра TR (содержит селектор дескриптора в GDT). Команда LTR загружает регистр TR. Кроме загрузки непосредственно селектора, процессор автоматически загружает базовый адрес, лимит и атрибуты из дескриптора, находящегося в GDT. Команда STR сохраняет содержимое регистра TR в РОН или ОЗУ. | 0 - 216 |
| Названиеполя | Описание | Диапазон допустимых значений |
| Информация о ресурсах | Список, содержащий информацию об открытых файлах, программных каналах, именованных каналах, общих областях ОП и т.д. | 0 - 216 |
| Идентификатор родительского процесса | Для ускоренной работы системного вызова getppid(). | 0 - 216 |
| Список идентификаторов процессов-потомков | Для ускоренной работы системного вызова wait(). | 0 - 216 |
| Идентификатор пользователя | Для обеспечения многопользовательского режима. | 0 - 216 |

Таблица 4.2 – Контекст процесса

|  |  |  |
| --- | --- | --- |
| Названиеполя | Описание | Диапазондопустимыхзначений |
| РОН | Содержимое всех регистры общего назначения. EAX, EBX, ECX, EDX. Данное поле должно интерпретироваться как 4 подряд сохраненных 4-байтных значений. | 4 \* (0 – 232) |
| Селекторы | Селектор кодового сегмента (CS), селектор сегмента данных (DS), селектор сегмента стека (SS) и селекторы ES, FS, GS дескрипторов в LDT. Данное поле должно интерпретироваться как 6 подряд идущих 2-байтных значений. | 6 \* (0 – 216) |
| Регистрысмещений | Содержимое всех регистров смещений. EIP, ESP, EBP, ESI и EDI. 5 подряд идущих 4-байтных значений. | 5 \* (0 – 232) |
| Регистр флагов | Содержимое регистра EFLAGS. | 0 - 232 |
| Названиеполя | Описание | Диапазондопустимыхзначений |
| Регистр LDTR | Селектор дескриптора LDT в GDT. | 0 - 247 |
| Регистр CR3 | Содержимое регистра, содержащего базовый адрес каталога страниц. | 0 - 232 |
| Базовый адрес битового массива разрешенных операций ввода/вывода | Используется для ограничения доступа процесса к определенным портам ВВ. 0 – доступ к порту запрещен, 1 – доступ к порту разрешен. | 0 - 255 |

**5 Спецификация на разработку программного компонента «Планировщик и диспетчер процессов (ПИДП)»**

5.1 Общее описание

5.1.1 ПИДП – это программный комплекс, который вызывается, когда требуется любое действие, связанное с администрированием процессов в системе (создание/удаление процесса, перенос процесса из очереди заблокированных в очередь готовых и т.д.). Данная программа должна максимально быстро выполнять свои действия, так как она вызывается достаточно часто.

5.2 Основные компоненты

5.2.1 Планировщик – часть комплекса ПИДП, предназначенная для принятия решения о выборе следующего процесса на исполнение и переноса процессов между очередями.

5.2.2 Диспетчер – это часть программного комплекса ПИДП, предназначенная для реализации решения, выбранного планировщиком.

5.3 Ответственность компонентов

5.3.1 Сначала происходит поиск решения, а потом его реализация, то есть сначала вызывается планировщик, а потом уже диспетчер. Также может вызываться только планировщик, а диспетчер – нет (например, когда требуется просто перенести процесс из очереди заблокированных процессов в очередь готовых, если он получил данные от ВУ). Алгоритмы работы планировщика и диспетчера процессов представлены в приложении А.

5.4 Правила коммуникации

5.4.1 Функции, обеспечивающие планировку процессов, обмениваются указателями на дескрипторы процессов. Функция, обеспечивающая переключение контекста, работает с указателями на контексты процессов.

5.5 Основные структуры данных

5.5.1 Дескриптор (представлен в таблице 4.1), контекст (представлен в таблице 4.2), список готовых процессов (организован по принципу алгоритма RR с относительными приоритетами), список заблокированных процессов (организован по принципу список списков, то есть внутри списка заблокированных процессов находятся списки процессов, ожидающих ответ от НЖМД, ожидающих ответ от НГМД, ожидающих определенный семафор, ожидающих определенную очередь сообщений, ожидающих определенного сигнала и т.д.).

5.5.2 Указатель на начало списка готовых процессов, указатель на конец списка готовых процессов.

5.5.3 Указатель на структуру-описатель списка списков заблокированных процессов. Данная структура хранит в себе информацию о начале и конце каждой очереди. В случае появления нового устройства в системе необходимо только создать новую очередь и добавить информацию об ее начале и конце в данную структуру.

**6 Системные вызовы «Создать процесс» и «Удалить процесс»**

6.1 Системный вызов «Создать процесс»

Системный возов «Создать процесс» служит для создания почти полной копии родительского процесса (процесса, в котором был инициирован системный вызов). Для создания почти полной копии вызывающего процесса ОС должна скопировать некоторые данные из процесса-родителя в процесс-потомок. Выполнение процессов разделяется после данного системного вызова. Имя системного вызова вызова: creat\_proc. Входные данные: отсутствуют. Выходные данные: идентификатор процесса. Сам системный вызов реализован в ядре ОС, к которому обращается программа-заглушка в системной библиотеке (через прерывание). Перечень действий, совершаемым ядром ОС, представлен в таблице 6.1.

Таблица 6.1 – Системный вызов «Создать процесс»

|  |  |
| --- | --- |
| № этапа | Описание этапа |
| 1 | Проверить возможность создания нового процесса (кол-во процессов < 65535). |
| 2 | Выделить память в области ОС для дескриптора процесса. |
| 3 | Создать дескриптор для нового процесса. |
| 4 | Назначить новому процессу идентификатор. |
| 5 | Записать в поле «Идентификатор родительского процесса» идентификатор процесса-родителя. |
| 6 | Скопировать содержимое полей (приоритет, информация о ресурсах и идентификатор пользователя, запустившего процесс) дескриптора процесса-родителя. |
| 7 | Выделить память в области пользователя для процесса. |
| 8 | Выделить память в области ОС для контекста процесса. |
| 9 | Настроить содержимое контекста нового процесса. |
| 10 | Полностью скопировать образ памяти из процесса-родителя. |
| 11 | Обновить информацию у процесса-родителя о потомках. |
| 12 | Добавить указатель о новом процессе список готовых процессов. |

6.2 Системный вызов «Удалить процесс»

Системный вызов «Удалить процесс» служит для удаления уже существующего процесса. Причем удаление совершается самим ядром в принудительном порядке. Имя системного вызова: kill\_proc. Входные данные: идентификатор процесса. Выходные данные: отсутствуют. Сам системный вызов реализован в ядре ОС, к которому обращается программа-заглушка в системной библиотеке (через прерывание). Также программа-заглушка проверяет допустимость входного параметра. То есть идентификатор процесса должен быть беззнаковым 2-байтным целым числом. Перечень действий, совершаемым ядром ОС, представлен в таблице 6.2.

Таблица 6.2 – Системный вызов «Удалить процесс»

|  |  |
| --- | --- |
| № этапа | Описание этапа |
| 1 | Проверить существование данного идентификатора в таблице процессов. |
| 2 | Удалить информацию о текущем процессе из процесса-родителя. |
| № этапа | Описание этапа |
| 3 | Удалить из всех очередей указатель на дескриптор текущего процесса. |
| 4 | Освободить память от дескриптора, контекста и ОП уровня пользователя текущего процесса. |
| 5 | Вызвать планировщик. |

**7 Заключение**

7.1 В данном проекте была рассмотрена разработка программно-аппаратного комплекса «Планировщик и диспетчер процессов в системе разделения времени» с алгоритмом планирования RR и относительным приоритетом, а также некоторые системные вызовы. Проект показал, что программу планировщик надо разрабатывать очень тщательно, так как она является основой любой многозадачной ОС. В итоге получилось, что для нормальной работы планировщика и диспетчера процессов необходимо иметь в области ОП ОС как минимум дескриптор и контекст для каждого процесса, список готовых и заблокированных процессов. Также выяснилось, что переключение процессов – это длительная операция, так как приходится переключаться из режима пользователя в режим ядра, запускать процесс планировки, потом диспетчеризации, а потом снова переключаться обратно, на уровень пользователя. Системные вызовы создания и удаления процесса также требуют времени на обработку, так как им тоже нужно манипулировать данными в области ОЗУ ОС, для чего требуется также переключаться на уровень ядра.

Приложение А

Графические материалы

Начало

Конец

Выбрать первый элемент из списка в качестве максимального (по приоритету). Указатель на текущий элемент равен указателю на голову списка

Сдвинуть указатель текущего элемента на следующий элемент

Указатель на текущий элемент равен указателю на хвост списка

Нет

Да

Настроить указатель головы на максимальный элемент

Приоритет следующего элемента больше приоритета текущего элемента

Выбрать следующий элемент в качестве максимального элемента

Да

Нет

Рисунок А.1 – Блок-схема алгоритма работы планировщика

с очередью готовых процессов

Начало

Конец

Выбрать из параметров вызова нужную очередь заблокированных процессов

Настроить указатель головы на первый элемент списка

Рисунок А.2 – Блок-схема алгоритма работы планировщика

с очередью заблокированных процессов

Начало

Если планировщик вызван по прерыванию или системному вызову, связанному с разделяемым ресурсом, то запустить функцию работы с очередью заблокированных процессов

Если требуется планирование, то запустить функцию работы с очередью готовых процессов

Если требуется диспетчеризация процессов, то запустить диспетчер

Конец

Рисунок А.3 – Блок-схема алгоритма работы планировщика

Начало

Конец

Сохранить контекст текущей задачи

Загрузить контекст новой
задачи

Рисунок А.4 – Блок-схема алгоритма диспетчеризации

ПИДП

Планировщик

Диспетчер

Работа с очередью готовых процессов

Работа с очередью заблокированных процессов

Рисунок А.5 – Структурно-функциональная схема

планировщика и диспетчера процессов

**Список литературы**

Таненбаум Э.С. Современные операционные системы. 2-е изд. – М.: ПИТЕР, 2006.

Embedded X86 Programming: Protected Mode by Jean Gareau

Руководство по процессору Intel i80486.

http://www.brokensword.narod.ru/

http://asmdev.narod.ru/asmos/asmos.html

http://lowlevel.ru/

http://xkernel.excode.ru/

Исходный код ядра ОС Linux версии 0.01

http://www.citforum.ru/operating\_systems/bach/contents.shtml