Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет по лабораторной работе №1, часть 2 по курсу «Операционные системы»

Тема Функции обработчика прерываний системного таймера и пересчет динамических приоритетов

Студент Зайцева А.А.

Группа ИУ7-52Б

Оценка (баллы) _____

Преподаватели Рязанова Н.Ю.

Москва — 2021 г.

Оглавление

1 Функции обработчика прерывания от системного таймера

1.1 Unix

По тику:

- инкремент счётчика тиков аппаратного таймера;
- инкремент часов и других таймеров системы;
- обновление статистики использования процессора текущим процессом (инкрементмент поля p_cpu дескриптора текущего процесса до максимального значения, равного 127);
- декремент счетчика времени, оставшегося до отправления на выполнение отложенных вызовов, при достижении нулевого значения счетчика – выставление флага, указывающего на необходимость запуска обработчика отложенного вызова;
- декремент кванта текущего потока.

По главному тику:

- регистрация отложенных вызовов функций, относящихся к работе планировщика, таких как пересчет приоритетов;
- пробуждение (то есть регистрирация отложенного вызова процедуры wakeup, которая перемещает дескриптор процесса из списка “спящих” в очередь "готовых к выполнению") системных процессов `swapper` и `pagedaemon`;
- декремент счётчика времени, оставшегося до посылки одного из следующих сигналов:
  - `SIGALRM` – сигнал, посылаемый процессу по истечении времени, заданного функцией `alarm()` (будильник реального времени);

- `SIGPROF` – сигнал, посылаемый процессу по истечении времени, заданного в таймере профилирования (будильник профиля процесса);
- `SIGVTALRM` – сигнал, посылаемый процессу по истечении времени работы в режиме задачи (будильник виртуального времени).

По кванту:

- если текущий процесс превысил выделенный ему квант процессорного времени, отправка ему сигнала SIGXCPU.

1.2 Windows

По тику:

- инкремент счётчика системного времени;
- декремент остатка кванта текущего потока;
- декремент счётчиков отложенных задач;
- добавление процесса в очередь DPC (Deferred procedure call, отложенный вызов процедуры).

По главному тику:

- инициализация диспетчера настройки баланса путем освобождения объекта «событие», на котором он ожидает.

По кванту:

- инициация диспетчеризации потоков добавлением соответствующего объекта в очередь DPC.

2 Пересчёт динамических приоритетов

Динамические приоритеты могут быть только у пользовательских процессов. В операционных системах процесс является владельцем ресурсов, в том числе владельцем приоритета.

2.1 Unix

Очередь готовых к выполнению процессов формируется согласно приоритетам процессов и принципу вытесняющего циклического планирования: процессы с одинаковыми приоритетами выполняются в течении кванта времени циклически друг за другом. Если процесс, имеющий более высокий приоритет, поступает в очередь готовых к выполнению, планировщик вытесняет текущий процесс и предоставляет ресурс более приоритетному.

В современных системах `Unix` ядро является вытесняющим – процесс в режиме ядра может быть вытеснен более приоритетным процессом в режиме ядра.

Дескриптор процесса proc содержит следующие поля, относящиеся к приоритету:

2.1.1 Приоритеты процессов

Приоритет процесса в `UNIX` задаётся числом в диапазоне от 0 до 127, причём чем меньше значение, тем выше приоритет. Приоритеты 0–49 зарезервированы ядром операционной системы, прикладные процессы могут обладать приоритетом в диапазоне от 50 до 127. Приоритеты ядра являются фиксированными величинами.

Приоритеты прикладных задач могут изменяться во времени в зависимости от следующих двух факторов:

- фактор любезности – целое число в диапазоне от 0 до 39. Чем меньше значение фактора любезности, тем выше приоритет процесса. Фактор любезности процесса может быть изменён суперпользователем системным вызовом `nice`;

- степень загруженности процессора в момент последнего обслуживания им процесса.

Структура `proc` содержит следующие поля, относящиеся к приоритетам:

- `p_pri` – текущий приоритет планирования;
- `p_usrpri` – приоритет режима задачи;
- `p_cpu` – результат последнего измерения использования процессора;
- `p_nice` – показатель уступчивости, устанавливаемый пользователем.

Планировщик использует поле `p_pri` для принятия решения о том, какой процесс отправить на выполнение. Значения `p_pri` и `p_usrpri` одинаковы, когда процесс находится в режиме задачи. Когда процесс просыпается после блокировки в системном вызове, его приоритет временно повышается. Планировщик использует `p_usrpri` для хранения приоритета, который будет назначен процессу при переходе из режима ядра в режим задачи, а `p_pri` – для хранения временного приоритета для выполнения в режиме ядра.

Ядро связывает приоритет сна (0–49) с событием или ожидаемым ресурсом, из-за которого процесс может быть заблокирован. Когда блокированный процесс просыпается, ядро устанавливает `p_pri`, равное приоритету сна события или ресурса, на котором он был заблокирован, следовательно, такой процесс будет назначен на выполнение раньше, чем другие процессы в режиме задачи.

В таблице 2.1 приведены значения приоритетов сна для систем `4.3BSD UNIX` и `SCO UNIX`. Такой подход позволяет системным вызовам быстрее завершать свою работу. По завершении процессом системного вызова его приоритет сбрасывается в значение текущего приоритета в режиме задачи. Если при этом приоритет окажется ниже, чем приоритет другого запущенного процесса, ядро произведет переключение контекста.

Приоритет в режиме задачи зависит от уступчивости и последней измеренной величины использования процессора. Степень уступчивости – это число в диапазоне от 0 до 39 со значением 20 по умолчанию.

Таблица 2.1 – Системные приоритеты сна

Событие	**Приоритет 4.3BSD UNIX**	**Приоритет SCO UNIX**
Ожидание загрузки в память страницы	0	95
Ожидание индексного дескриптора	10	88
Ожидание ввода–вывода	20	81
Ожидание буфера	30	80
Ожидание терминального ввода	30	75
Ожидание терминального вывода	30	74
Ожидание завершения выполнения	30	73
Ожидание события	40	66

Системы разделения времени стараются выделить процессорное время таким образом, чтобы все процессы системы получили его в равных количествах, что требует слежения за использованием процессора. Поле `p_cpu` содержит величину последнего измерения использования процессора процессом. При создании процесса это поле инициализируется нулем. На каждом тике обработчик таймера увеличивает `p_cpu` на единицу для текущего процесса, вплоть до максимального значения – 127. Каждую секунду ядро вызывает процедуру `schedcpu`, которая уменьшает значение `p_cpu` каждого процесса исходя из фактора «полураспада».

В `4.3BSD` для расчета применяется формула

$$d = \frac{2 \cdot la}{2 \cdot la + 1},$$

где `la` – `load_average` – это среднее количество процессов в состоянии готовности за последнюю секунду.

Кроме того, процедура `schedcpu` также пересчитывает приоритеты режима задачи всех процессов по формуле

$$p_{usrpri} = PUSER + \frac{p\_cpu}{4} + 2 \cdot p\_nice,$$

где `PUSER` – базовый приоритет в режиме задачи, равный 50.

Если процесс до вытеснения другим процессом использовал большое

количество процессорного времени, его `p_cpu` будет увеличен, что приведет к увеличению значения `p_usrpri` и к понижению приоритета.

Чем дольше процесс простаивает в очереди на выполнение, тем меньше его `p_cpu`. Это позволяет предотвратить зависания низкоприоритетных процессов. Если процесс бóльшую часть времени выполнения тратит на ожидание ввода-вывода, то он остается с высоким приоритетом.

Системы разделения времени пытаются выделить процессорное время таким образом, чтобы конкурирующие процессы получили его примерно в равных количествах. Фактор полураспада обеспечивает экспоненциально взвешанное среднее значение использования процессора в течение функционирования процесса. Формула, применяемая в SVR3 имеет недостаток: вычисляя простое экспоненциальное среднее, она способствует росту приоритетов при увеличении загрузки системы.

2.2 Windows-системы

В Windows реализуется приоритетная, вытесняющая система планирования, при которой всегда выполняется хотя бы один работоспособный (готовый) поток с самым высоким приоритетом.

После того как поток был выбран для запуска, он запускается на время, называемое квантом. Но поток может и не израсходовать свой квант времени: если становится готов к запуску другой поток с более высоким приоритетом, текущий выполняемый поток может быть вытеснен.

Единого модуля под названием «планировщик» не существует. Процедуры, выполняющие обязанности по диспетчеризации, обобщенно называются диспетчером ядра. Диспетчеризации потоков могут потребовать следующие события:

- поток становится готовым к выполнению;
- поток выходит из состояния выполнения из-за окончания его кванта времени;
- поток завершается или переходит в состояние ожидания;
- изменяется приоритет потока;

- изменяется родственность процессора потока.

Windows использует 32 уровня приоритета, от 0 до 31. Эти значения разбиваются на части следующим образом:

- шестнадцать уровней реального времени (от 16 до 31);
- шестнадцать изменяющихся уровней (от 0 до 15), из которых уровень 0 зарезервирован для потока обнуления страниц.

Уровни приоритета потоков назначаются из двух разных позиций: от Windows API и от ядра Window.

Windows API систематизирует процессы по классу приоритета, который присваивается им при их создании:

- реального времени (real-time (4));
- высокий (high (3));
- выше обычного (above normal (7));
- обычный (normal (2));
- ниже обычного (below normal (5));
- простой (idle (1)).

Затем назначается относительный приоритет потоков в рамках процессов. Здесь номера представляют изменение приоритета, применяющееся к базовому приоритету процесса:

- критичный по времени (time-critical (15));
- наивысший (highest (2));
- выше обычного (above normal (1));
- обычный (normal (0));
- ниже обычного (below -normal (-1));
- самый низший (lowest (-2));

- простоя (idle (-15)).

Уровень, критичный по времени, и уровень простоя (+15 и –15) называются уровнями насыщения и представляют конкретные применяемые уровни вместо смещений.

Относительный приоритет – это приращение к базовому приоритету процесса.

Соответствие между приоритетами Windows API и ядра системы приведено в таблице 2.2.

Таблица 2.2 – Соответствие между приоритетами Windows API и ядра Windows

	real-time	**high**	**above normal**	**normal**	**below normal**	**idle**
time critical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1

Процесс обладает только базовым приоритетом, тогда как поток имеет базовый, который наследуется от приоритета процесса, и текущий приоритет. Процесс по умолчанию наследует свой базовый приоритет у того процесса, который его создал.

Приложения пользователя обычно запускаются с базовым приоритетом (normal), поэтому их исходный поток чаще всего выполняется с уровнем приоритета 8.

Повышение приоритета вступает в действие немедленно и может вызвать изменения в планировании процессора. Однако если поток использует весь свой следующий квант, то он теряет один уровень приоритета и перемещается вниз на одну очередь в массиве приоритетов. Если же он использует второй полный квант, то он перемещается вниз еще на один уровень, и так до тех пор, пока не дойдет до своего базового уровня. Повышение приоритета потока в Windows применяется только для потоков с

приоритетом динамического диапазона (0-15). Но каким бы ни было приращение, приоритет потока никогда не будет больше 15.

Сценарии повышения приоритета:

- Повышение приоритета владельца блокировки.
- Повышение вследствие завершения ввода-вывода (сокращение задержек – поток может вновь быстро запуститься и начать новую операцию ввода-вывода).
- Повышение вследствие ввода из пользовательского интерфейса (сокращение задержек и времени отклика).
- Повышение вследствие слишком продолжительного ожидания ресурса исполняющей системы (предотвращение зависания).
- Повышение в случае, когда готовый к запуску поток не был запущен в течение определенного времени (предотвращение зависания и смены приоритетов).
- Если поток ждал на семафоре, мьютексе или другом событии, то при его освобождении он получает повышение приоритета на два уровня, если находится в фоновом процессе, и на один уровен во всех остальных случаях;
- Повышение приоритета проигрывания мультимедиа, управляемое службой MMCSS (это не является настоящим повышением, служба устанавливает по необходимости новые базовые приоритеты для потоков).

Вывод

Функции обработчика прерывания от системного таймера для семейства ОС `Windows` и для семейства ОС `UNIX/Linux` схожи по своим действиям:

- инициализируют отложенные действия, относящиеся к работе планировщика, такие как пересчёт приоритетов;
- выполняют декремент счётчиков времени: часов, таймеров, будильников реального времени, счётчиков времени отложенных действий;
- выполняют декремент кванта: текущего процесса в `Linux`, текущего потока в `Windows`.

Такая схожесть объясняется тем, что обе системы являются системами разделения времени с динамическими приоритетами и вытеснением.