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 по курсу "Операционные системы"

Тема функции прерывания от системного таймера в системах разделения времени

Студент Шацкий Р.Е.

Группа ИУ7-55Б

Оценка (баллы) _____

Преподаватели Рязанова Н.Ю.

Москва — 2021 г.

1 Функции обработчика прерывания от системного таймера

Обработчик прерываний от системного таймера имеет наивысший приоритет. Никакая другая работа в системе не может выполняться во время обработчика прерывания от системного таймера. Обработчик прерывания от системного таймера должен завершаться как можно быстрее, чтобы не влиять на отзывчивость системы (отзывчивость – как быстро система отвечает на запросы пользователей).

Тик — период времени между двумя последующими прерываниями таймера.

Основной тик — период времени равный n тикам таймера (число n зависит от конкретного варианта системы).

Квант времени (quantum, time slice) — временной интервал, в тече- ние которого процесс может использовать процессор до вытеснения другим процессом.

1 Unix, системный таймер

По тику

- инкремент счетчика тиков аппаратного таймера;
- инкремент счетчика использования процессора текущим процессом(инкремент поля p_cpu структуры proc до максимального значения — 127);
- инкремент часов и других таймеров системы;
- декремент кванта текущего потока;
- декремент счетчика времени до отправления на выполнение отложенных вызовов, при достижении счетчиком нуля происходит выставление флага для обработчика отложенного вызова.

По главному тику

- пробуждение в нужные моменты системных процессов, таких как swapper и pagedaemon. (”пробуждение” тут понимается так: регистрация отложенного вызова процедуры wakeup, которая перемещает дескрипторы процессов из списка ”спящие” в очередь ”готовых к выполнению”);

- регистрация отложенных вызовов функции, которые относятся к работе планировщика;

- декремент счетчик времени, которое осталось до отправления одного из следующих сигналов:

  – SIGVTALRM — сигнал, посылаемый процессу по истечении времени, заданного в “виртуальном” таймере;

  – SIGPROF — сигнал, посылаемый процессу по истечении времени заданного в таймере профилирования;

  – SIGALRM — сигнал, посылаемый процессу по истечении времени, предварительно заданного функцией alarm().

По кванту

- сигнал SIGXCPU текущему процессу, если он превысил выделенную для него квоту использования процессора. По получении сигнала обработчик сигнала прерывает выполнение процесса.

2 Windows, системный таймер

По тику

- инкремент счетчика системного времени;

- декремент счетчиков времени отложенных задач;

- декремент кванта текущего потока;

- если активен механизм профилирования ядра, то инициализация отложенного вызова обработчика ловушки профилирования ядра с помощью постановки объекта в очередь DPC (обработчик ловушки профилирования регистрирует адрес команды, выполнявшейся на момент прерывания).

По главному тику

- освобождение объекта «событие», которое ожидает диспетчер настройки баланса. Диспетчер настройки баланса по событию от таймера сканирует очередь готовых процессов и повышает приоритет процессов, которые находились в состоянии ожидания дольше 4 секунд.

По кванту

- инициация диспетчеризации потоков (добавление соответствующего объекта в очередь DPC (Deferred procedure call — отложенный вызов процедуры)).

2 Пересчет динамических приоритетов

В ОС семейства UNIX и Windows динамически могут пересчитываться приоритеты **только пользовательских процессов.**

1 Unix

Планирование процессов в UNIX основано на приоритете процесса. Планировщик всегда выбирает процесс с наивысшим приоритетом. Приоритеты процессоров изменяются с течением времени (динамически) системой в зависимости от использования вычислительных ресурсов, времени ожидания запуска и текущего состояния процесса. Если процесс готов к запуску и имеет наивысший приоритет, планировщик приостановит выполнение текущего процесса с более низким приоритетом, даже если тот не «выработал» свой временной квант.

Традиционное ядро UNIX является строго невытесняющим, однако в современных системах UNIX ядро является вытесняющим – то есть процесс в режиме ядра может быть вытеснен более приоритетным процессом в режиме ядра. Ядро сделано вытесняющим для того, чтобы система могла обслуживать процессы реального времени, например видео и аудио.

Очередь процессов, готовых к выполнению, формируется согласно приоритетам и принципу вытесняющего циклического планирования, то есть сначала выполняются процессы с большим приоритетом, а процессы с одинаковым приоритетом выполняются в течении кванта времени друг за другом циклически. В случае, если процесс с более высоким приоритетом поступает в очередь процессов, готовых к выполнению, планировщик вытесняет текущий процесс и предоставляет ресурс более приоритетному процессу.

Приоритет процесса задается любым целым числом, которое лежит в диапазоне от 0 до 127 (чем меньше число, тем выше приоритет):

- 0 – 49 — зарезервированы для ядра (приоритеты ядра фиксированы);
- 50 – 127 — прикладные (приоритеты прикладных задач могут изменяться во времени).

Изменение приоритета прикладных задач зависит от следующих факторов:

- **фактор "любезности" (nice)** – это целое число в диапазоне от 0 до 39 со значением 20 по умолчанию. Увеличение значения приводит к уменьшению приоритета. Пользователи могут повлиять на приоритет процесса при помощи изменения значений этого фактора, но только суперпользователь может увеличить приоритет процесса. Фоновые процессы автоматически имеют более высокие значения фактора любезности;

- последняя измеренная величина использования процессора.

Структура proc содержит следующие поля, которые относятся к приоритетам:

- **p_pri** – текущий приоритет планирования;

- **p_usrpri** – приоритет режима задачи;

- **p_cpu** – результат последнего измерения использования процессора;

- **p_nice** – фактор 'любезности', который устанавливается пользователем.

p_pri используется планировщиком для принятия решения о том, какой процесс отправить на выполнение.

p_pri и **p_usrpri** равны, когда процесс находится в режиме задачи.

Значение **p_pri** может быть изменено (увеличено) планировщиком для того, чтобы выполнить процесс в режиме ядра. В таком случае **p_usrpri** будет использоваться для хранения приоритета, который будет назначен процессу при возврате в режим задачи.

p_cpu инициализируется нулем при создании процесса (и на каждом тике обработчик таймера увеличивает это поле текущего процесса на 1, до максимального значения равного 127).

Ядро системы связывает приоритет сна с событием или ожидаемым ресурсом, из-за которого процесс может блокироваться (приоритет сна определяется для ярда, поэтому лежит в диапазоне 0 - 49). Когда процесс "просыпается", ядро устанавливает в поле **p_pri** приоритет сна – значение приоритета из диапазона системных приоритетов, зависящее от события или ресурса по которому произошла блокировка (событие и связанное с ним значение приоритета сна в системе 4.3BSD представлены в таблице 2.1).

Таблица 2.1: Таблица приоритетов в системе **4.3BSD**

Приоритет	**Значение**	**Описание**
PSWP	0	Свопинг
PSWP + 1	1	Страничный демон
PSWP + 1/2/4	1/2/4	Другие действия по обработке памяти
PINOD	10	Ожидание освобождения inode
PRIBIO	20	Ожидание дискового ввода-вывода
PRIBIO + 1	21	Ожидание освобождения буфера
PZERO	25	Базовый приоритет
TTIPRI	28	Ожидание ввода с терминала
TTOPRI	29	Ожидание вывода с терминала
PWAIT	30	Ожидание завершения процесса потомка
PLOCK	35	Консультативное ожидание блокированного ресурса
PSLEP	40	Ожидание сигнала

Каждую секунду ядро системы инициализирует отложенный вызов процедуры schedcpu(), которая уменьшает значение **p_pri** каждого процесса, исходя из фактора "полураспада" (в системе 4.3BSD считается по формуле 2.1).

$$decay = \frac{2 \cdot load\_average}{2 \cdot load\_average + 1} \tag{2.1}$$

где **load_average** – среднее количество процессов, находящихся в состоянии готовности к выполнению (за последнюю секунду).

Процедура schedcpu() пересчитывает приоритеты для режима задачи всех процессов по формуле 2.2.

$$p\_usrpri = PUSER + \frac{p\_cpu}{2} + 2 \cdot p\_nice \tag{2.2}$$

где **PUSER** – базовый приоритет в режиме задачи, который равен 50.

Таким образом, если процесс в последний раз использовал большое количество процессорного времени, то его **p_cpu** будет увеличен. Это приведет к росту значения **p_usrpri**, то есть к понижению приоритета. Чем дольше процесс простаивает в очереди на выполнение, тем больше фактор полураспада уменьшает его **p_cpu**, что приводит к повышению его приоритета.

Такая схема предотвращает бесконечное откладывание низкоприоритетных процессов. Применение данной схемы предпочтительно процессам, осуществляющим много операций ввода-вывода, в противоположность процессам, производящим много вычислений. То есть, если процесс большинство времени выполнения тратит на ожидание ввода-вывода, то он остается с высоким приоритетом и, таким образом, быстрее получает процессор при необходимости.

В то же время вычислительные приложения обычно обладают более высокими значениями **p_cpu** и работают на значительно более низких приоритетах.

Динамический пересчет приоритетов процессов в режиме задачи позволяет избежать бесконечного откладывания.

Таким образом, приоритет процесса в режиме задачи может быть динамически пересчитан по следующим причинам:

- изменения фактора любезности процесса системным вызовом nice;
- в зависимости от степени загруженности процессора процессом p_cpu;
- вследствие ожидания процесса в очереди готовых к выполнению процессов;
- приоритет может быть повышен до соответствующего приоритета сна вследствие ожидания ресурса или события.

2 Windows

В Windows реализуется приоритетная, вытесняющая система планирования, при которой всегда выполняется хотя бы один работоспособный (готовый) поток с самым высоким приоритетом, с той оговоркой, что конкретные, имеющие высокий приоритет и готовые к запуску потоки могут быть ограничены процессами, на которых им разрешено или предпочтительнее всего работать.

В Windows планировка потоков осуществляется на основании приоритетов готовых к выполнению потоков. Поток с более низким приоритетом вытесняется планировщиком, когда поток с более высоким приоритетом становится готовым к выполнению.

Код Windows, отвечающий за планирование, реализован в ядре. Нет единого модуля или процедуры с названием "планировщик", так как этот код рассредоточен по ядру.

Совокупность процедур, выполняющих обязанности планировщика, называется диспетчером ядра. Диспетчеризация потоков может быть вызвана:

- поток готов к выполнению (только что создан или вышел из состояния "ожидания");
- поток выходит из состояния "выполняется", т.к. его квант истек, либо поток завершается, либо переходит в состояние "ожидание";
- поменялся приоритет потока;
- изменилась привязка к процессорам => поток больше не может работать на процессоре, на котором он выполнялся.

Windows использует 32 уровня приоритета:

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

Уровни приоритета потоков назначаются исходя из двух разных позиций: одной от Windows API и другой от ядра Windows.

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

- реального времени — Real-time (4);
- высокий — High (3);
- выше обычного — Above Normal (6);
- обычный — Normal (2);
- ниже обычного — Below Normal (5);
- простой — Idle (1).

После назначается относительный приоритет отдельных потоков внутри этих процессов:

- 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).

Исходный базовый приоритет потока наследуется от базового приоритета процесса. Процесс по умолчанию наследует свой базовый приоритет у того процесса, который его создал. Соответствие между приоритетами Windows API и ядра системы приведено в таблице 2.2.

Текущий приоритет потока в динамическом диапазоне – от 1 до 15 может быть повышен планировщиком вследствие следующих причин:

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

- **Повышение вследствие событий планировщика или диспетчера (сокращение задержек)** При наступлении события диспетчера вызываются процедуры с целью проверить не должны ли на локальном процессоре быть намечены какие-либо потоки, которые не должны быть спланированы. При каждом наступлении такого события вызывающий код может также указать, какого типа повышение должно быть применено к потоку, а также с каким приращением приоритета должно быть связано это повышение;

- **Повышения приоритета, связанные с завершением ожида- ния** В общем случае поток, пробуждающийся из состояния ожидания, должен иметь возможность приступить к выполнению как можно скорее;

- **Повышение приоритета владельца блокировки** Так как блокировки ресурсов исполняющей системой и блокировки критических разделов используют основные объекты диспетчеризации, то в результате освобождения эих блокировок осуществляются повышения приоритеты, связанные с завершением ожидания. Но сдругой строны, так как высокоуровневые реализации этих объектов отслеживают владельца блокировки, то ядро может принять решение о том, какого вида повышение должно быть применено с помощью AdjustBoost;

- **Повышение вследствие завершения ввода-вывода** Windows дает

временное повышение приоритета при завершении определенных операций ввода/вывода, при этом потоки, которые ожидали ввода/вывода имеют больше шансов сразу же запуститься. Подходящее значение для увеличения зависит от драйвера устройств (представлены в таблице 2.3);

Таблица 2.3: Рекомендуемые значения повышения приоритета.

Устройство	**Приращение**
Диск, CD-ROM, параллельный порт, видео	1
Сеть, почтовый ящик, именованный канал, последовательный порт	2
Клавиатура, мышь	6
Звуковая плата	8

- **Повышение при ожидании ресурсов исполняющей системы** Если поток пытается получить ресурс исполняющей системы, который уже находится в исключительном владении другого потока, то он должен войти в состояние ожидания до тех пор, пока другой поток не освободит ресурс. Для ограничения риска взаимных исключений исполняющая система выполняет это ожидание, не входя в бесконечное ожидание ресурса, а интервалами по 500 мс. Если по окончании этих 500 мс ресурс все также находится во владении, то исполняющая система пытается предотвратить зависание центрального процессора путем получения блокировки диспетчера, повышения приоритета потока (потоков), владеющих ресурсом до 15 в случае если исходный приоритет владельца был меньше, чем у ожидающего, и не был равен 15), перезапуска их квантов и выполнения еще одного ожидания;

- **Повышение приоритета потоков первого плана после ожидания** Смысл такого повышения заключается в улучшении скорости отклика интерактивных приложений, то есть если дать приложениям первого плана небольшое повышение приоритета при завершении ожидания, то у них повышаются шансы сразу же приступить к работе, осо-

бенно когда другие процессы с таким же базовым приоритетом могут быть запущены в фоновом режиме;

- **Повышение приоритета после пробуждения GUI-потока** Потоки — владельцы окон получают при пробуждении дополнительное повышение приоритета на 2 из-за активности при работе с окнами, например, при поступлении сообщений от окна. Система работы с окнами Win32k.sys применяет это повышение приоритета, когда вызывает функцию KeSetEvent для установки события, используемого для пробуждения GUI-потока. Смысл такого повышения схож со смыслом предыдущего повышения — содействие интерактивным приложениям;

- **Повышения приоритета, связанные с перезагруженностью центрального процессора** Диспетчер настройки баланса (механизм ослабления загруженности центрального процессора) сканирует очередь готовых потоков раз в секунду и, если обнаружены потоки, ожидающие выполнения более 4 секунд, то диспетчер настройки баланса повышает их приоритет до 15. Как только квант истекает, приоритет потока снижается до базового приоритета. Если поток не был завершен за квант времени или был вытеснен потоком с более высоким приоритетом, то после снижения приоритета поток возвращается в очередь готовых потоков. Диспетчер настройки баланса сканирует лишь 16 готовых потоков и повышает приоритет не более чем у 10 потоков (если найдет) за один проход. При следующем проходе сканирование возобновляется с того места, где оно было прервано в прошлый раз;

- **Повышение приоритетов для мультимедийных приложений и игр** Потоки, на которых выполняются различные мультимедийные приложения, должны выполняться с минимальными задержками. В Windows такая задача решается с помощью повышения приоритетов таких потоков драйвером **MMCSS** (MultiMedia Class Scheduler Service). MMCSS работает с различными определенным задачи, например:

  – аудио;

  – аудио профессионального качества;

– игры;

– захват;

– воспроизведение;

– низкая задержка;

– задачи администратора многооконного режима;

– распределение.

Важное свойство для планирования потоков – категория планирования, это первичный фактор, который определяет приоритет потоков, зарегистрированных с MMCSS (категории планирования указаны в таблице 2.4).

Функции MMCSS временно повышают приоритет потоков, зарегистрированных с MMCSS до уровня, который соответствует категории планирования. Потом их приоритет снижается до уровня, соответствующего категории планирования Exhausted, для того, чтобы другие потоки тоже могли получить ресурс.

Таблица 2.4: Категории планирования

Категория	**Приоритет**	**Описание**
High (Высокая)	23-26	Потоки профессионального аудио (Pro Audio), запущенные с приоритетом выше, чем у других потоков на системе, за исключением критических системных потоков
Medium (Средняя)	16-22	Потоки, являющиеся частью приложений первого плана, например Windows Media Player
Low (Низкая)	8-15	Все остальные потоки, не являющиеся частью предыдущих категорий
Exhausted (Исчерпавших потоков)	1-7	Потоки, исчерпавшие свою долю времени центрального процессора, выполнение которых продолжиться, только если не будут готовы к выполнению другие потоки с более высоким уровнем приоритета

IRQL

Для обеспечения поддержки мультизадачности системы, когда исполняется код режима ядра, Windows использует приоритеты прерываний IRQL.

Прерывания обслуживаются в порядке их приоритета. При возникновении прерывания с высоким приоритетом процессор сохраняет информацию о состоянии прерванного потока и активизирует сопоставленный с данным прерыванием диспетчер ловушки. Последний повышает IRQL и вызывает процедуру обслуживания прерывания (ISR).

После выполнения ISR диспетчер прерывания понижает IRQL процессора до исходного уровня и загружает сохраненные ранее данные о состоянии машины. Прерванный поток возобновляется с той точки, где он был прерван. Когда ядро понижает IRQL, могут начать обрабатываться ранее замаскированные прерывания с более низким приоритетом. Тогда вышеописанный процесс повторяется ядром для обработки и этих прерываний.

3 Вывод

Функции обработчика прерывания от системного таймера в режиме long для ОС семейства UNIX и Windows схожи, так как эти ОС являются системами разделения времени. Общие основные функции:

- декремент кванта текущего процесса в UNIX и декремент текущего потока в Windows;
- инициализация отложенных действий, которые относятся к работе планировщика (например, пересчет приоритетов);
- декремент счетчиков времени (таймеров, часов, счетчиков времени отложенных действий, будильников реального времени).

Обе операционные системы (UNIX и Windows) – это системы разделения времени с вытеснением и динамическими приоритетами.

В ОС UNIX приоритет пользовательского процесса (процесса в режиме задач) может динамически пересчитываться, в зависимости от фактора "любезности", p_cpu (результат последнего измерения использования процессора) и

базового приоритета (PUSER). Приоритеты ядра – фиксированные величины.

В ОС Windows при создании процесса ему назначается базовый приоритет, относительно базового приоритета процесса потоку назначается относительный приоритет, таким образом, у потока нет своего приоритета. Приоритет потока пользовательского процесса может быть динамически пересчитан.