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 дисциплине "Операционные системы"

Тема Изучение функций прерывания от системного таймера

Студент Турчанинов А. М.

Группа ИУ7-54Б

Оценка (баллы) _____

Преподаватели Рязанова Н. Ю.

Москва — 2022 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. Поэтому он должен завершаться как можно быстрее, чтобы не влиять на отзывчивость системы (отзывчивость показывает, насколько быстро система отвечает на запросы пользователей).

1.1 UNIX

По тику:

- инкремент счетчика реального времени;
- инкремент счетчика использования процессора текущим процессом (т.е. инкремент поля `p_cpu` структуры `proc` до максимального значения);
- декремент таймера, который определяет оставшееся время до посылки сигнала процессу. Существует 3 типа таймеров:
  - Таймер реального времени. Этот таймер используется для отсчета реального времени. Когда значение таймера становится равным нулю, процессу отправляется сигнал SIGALRM;
  - Таймер профилирования. Этот таймер уменьшается только когда процесс выполняется в режиме ядра или задачи. Когда значение таймера становится равным нулю, процессу отправляется сигнал SIGPROF;
  - Таймер виртуального времени. Этот таймер уменьшается только когда процесс выполняется в режиме задачи. Когда значение таймера становится равным нулю, процессу отправляется сигнал SIGVTALRM.
- проверка необходимости выполнения отложенных вызовов. Реализация этой проверки может варьироваться (*). При установлении факта необходимости выполнения отложенного вызова происходит установка флага, указывающего на необходимость запуска обработчика отложенного вызова;
- декремент кванта текущего потока.

(*) Например, может быть реализован отсортированный по времени запуска двунаправленный список отложенных вызовов, в котором каждый элемент содержит разницу между временем своего запуска и временем запуска предыдущего отложенного вызова. Тогда ядро системы на каждом тике декрементирует счетчик времени до запуска первого в списке отложенного вызова и при достижении счетчиком нуля происходит установка флага, указывающего на необходимость запуска обработчика отложенного вызова. В другой реализации такого списка элементы хранят абсолютное время своего запуска. При совпадении текущего времени и времени первого в списке отложенного вызова происходит установка флага. Также для реализации механизма запуска отложенных вызовов может быть использован метод карусели.

По главному тику:

- регистрация отложенных вызовов функций, относящихся к работе планировщика, таких как пересчет приоритетов (под регистрацией понимается вставка в список отложенных вызовов);
- пробуждение в нужные моменты системных процессов, таких как `swapper` и `pagedaemon`. Под пробуждением понимается регистрация отложенного вызова процедуры `wakeup`, которая меняет статусы процессов со `sleeping` на `running`.

По кванту:

- посылка текущему процессу сигнала `SIGXCPU`, если он превысил выделенную ему квоту использования процессора. При получении сигнала обработчик прерывает выполнение процесса.

1.2 Windows

По тику:

- инкремент счетчика реального времени;
- декремент счетчиков времени отложенных задач;
- декремент кванта текущего потока на величину, равную количеству тактов процессора, произошедших за тик;
- если активен механизм профилирования ядра, инициализация отложенного вызова обработчика ловушки профилирования ядра путем постановки объекта в очередь DPC (обработчик ловушки профилирования регистрирует адрес команды, выполнявшейся на момент прерывания).

По главному тику:

- освобождение объекта "событие", которого ожидает диспетчер настройки баланса. (Диспетчер настройки баланса сканирует очередь готовых процессов и повышает приоритет процессов, находящихся в состоянии ожидания дольше 4 секунд).

По кванту:

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

2 | Пересчёт динамических приоритетов

В операционных системах семейства UNIX и семейства Windows могут пересчитываться приоритеты только пользовательских процессов.

2.1 UNIX

Планирование процессов в UNIX основано на приоритете процесса. Планировщик всегда выбирает процесс с наивысшим приоритетом. Приоритеты процессов изменяются с течением времени 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.

Очередь готовых к выполнению процессов формируется согласно их приоритетам: сначала выполняются процессы с большим приоритетом, а процессы с одинаковым приоритетом выполняются в течение кванта времени друг за другом. В случае, если процесс с более высоким приоритетом поступает в очередь процессов, готовых к выполнению, планировщик вытесняет текущий процесс и предоставляет ресурс более приоритетному.

Приоритет процесса задается целым числом из диапазона от 0 до 127 (чем меньше число, тем выше приоритет). Диапазон 0 – 49 зарезервирован для ядра (приоритеты ядра фиксированы). Диапазон 50 – 127 используется прикладными процессами (приоритеты прикладных процессов могут изменяться во времени).

Дескриптор процесса `proc` содержит следующие поля, относящиеся к приоритету:

- `p_pri` – текущий приоритет планирования;
- `p_usrpri` – приоритет в режиме задачи;
- `p_cpu` – результат последнего измерения использования процессора;
- `p_nice` – фактор “любезности”, устанавливаемый пользователем.

Для принятия решения о том, какой процесс отправить на выполнение, планировщик использует `p_pri`. Когда процесс находится в режиме задачи, `p_pri` = `p_usrpri`. Однако, когда процесс просыпается после блокировки, его приоритет будет временно повышен для выполнения в режиме ядра. Поэтому планировщик использует `p_usrpri` для хранения приоритета, который будет назначен процессу при возврате в режим задачи. Ядро системы связывает приоритет сна с событием или ожидаемым ресурсом, из-за которого процесс может заблокироваться (приоритет сна определяется для ядра, 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.

В таблице 2.1 приведены приоритеты сна в `ОС 4.3 BSD`.

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 блок. ресурса
`PSLEEP`	40	Ожидание сигнала

Приоритет в режиме задачи зависит от двух факторов: степени "любезности" `p_nice` и результата последнего измерения использования процессора `p_cpu`.

Степень "любезности" `p_nice` — 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 значения этой степени (увеличения степени любезности), но только суперпользователь может уменьшить степень любезности (то есть повысить приоритет процесса).

Результат последнего измерения использования процессора `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). Каждую секунду ядро системы инициализирует отложенный вызов процедуры `schedcpu()`, которая уменьшает значение p_cpu каждого процесса, исходя из фактора *"полураспада"*. Расчёт производится 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, равный 50.

Таким образом, если процесс в последний раз использовал большое количество процессорного времени, его `p_cpu` будет увеличен, что приведёт к росту значения `p_usrpri` т.е. 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. Её применение предпочтительнее процессам, осуществляющим много операций ввода-вывода.

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 тоже время вычислительные приложения обычно обладают более высокими значениями `p_cpu` и работают на значительно более низких приоритетах

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

- вследствие 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.

Традиционно ядро UNIX является непрерываемым, то есть процесс, выполняющийся в режиме ядра, не может быть прерван системой, а вычислительные ресурсы переданы более высокоприоритетному процессу. Однако в современных системах UNIX ядро является вытесняемым (процесс в режиме ядра может быть вытеснен более приоритетным процессом в режиме ядра). Ядро сделано вытесняемым для того, чтобы система могла обслуживать процессы реального времени, например, работу с видео и аудио.

2.2 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 более высокоприоритетным потоком, когда тот 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 эти обязанности, называется диспетчером ядра. Диспетчеризация потоков может быть вызвана в случаях:

- 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, следовательно, поток больше не может работать на процессоре, на котором он выполнялся.

Windows использует 32 уровня приоритета:

- от 0 до 15 — динамические уровни (уровень 0 зарезервирован для потока обнуления страниц).
- от 16 до 31 — уровни 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).

Затем 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);
- низший (lowest, -2);
- простой (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:

- **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.

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

- **Повышения связанные с завершением ожидания**

  Повышения приоритета, связанные с завершением ожидания, пытаются уменьшить время задержки между потоком, пробуждающимся по сигналу объекта, и потоком, фактически приступившим к своему выполнению в процессе, который не находился в состоянии ожидания. Поскольку событие, наступление которого ждал поток, может дать информацию того или иного сорта, важно, чтобы это состояние не изменялось. В противном случае эта информация может стать неактуальной или неверной, как только поток будет запущен.

- **Повышение приоритета владельца блокировки**

  Так как блокировки ресурсов исполняющей системой и блокировки критических разделов используют основные объекты диспетчеризации, то в результате освобождения этих блокировок осуществляются повышения приоритетов, связанные с завершением ожидания. Но с другой стороны, т.к 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, особенно 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 перегруженностью 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 задачи, например:

– воспроизведение медиа контента;

– игры;

– использование функции записи экрана;

– задачи администратора многоэкранного режима.

Важное свойство для планирования потоков — категория планирования -– 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

Вывод

Несмотря на то, что Windows и Unix разные операционные системы, обработчик системного таймера выполняет схожие основные функции:

- Регистрация отложеннных действий, относящихся к работе планировщика, такие как пересчет приоритетов;
- Декремент счетчиков времени: часов, таймеров, будильников реального времеени, счетчиков времени отложенных действий.
- Декремент кванта.

Схожееть функций системного таймера обусловлена тем, что и UNIX, и Windows, являются системами разделения времени с приоритетами и вытеснением. Тем не менее, эти системы отличаются подходами к реализации планирования и перерасчета приоритетов процессов и потоков.