МГТУ им. Баумана

Лабораторная работа №4

По курсу: "Операционные системы"

Отчет по лабораторной работе №1 (часть 2)

Работу выполнил: студент группы ИУ7-53Б
Наместник Анастасия

Преподаватель: Рязанова Н. Ю.

Москва, 2020

Оглавление

1 Функции обработчика прерывания от системного таймера в защищенном режиме

1.1 Функции обработчика прерывания от системного таймера в защищенном режиме для ОС семейства Windows

По тику обработчик прерываний от системного таймера выполняет следующие задачи:

1. Инкрементирует счетчик системного времени.
2. Декрементирует счетчики времени отложенных задач.
3. Декрементирует квант текущего потока.
4. Инициализирует отложенный вызов обработчика ловушки профилирования ядра путем постановки объекта в очередь DPC (если активен механизм профилирования ядра): обработчик ловушки профилирования регистрирует адрес команды, выполнявшейся на момент прерывания.

По главному тику обработчик прерываний от системного таймера выполняет следующую задачу: инициализирует диспетчер настройки баланса (Освобождает объект "событие" , которое ожидает диспетчер на-

стройки баланса. Таким образом, диспетчер настройки баланса по событию от таймера один раз в секунду сканирует очередь готовых потоков и повышает приоритет потоков, ожидающих выполнения более 4 секунд).

По кванту обработчик прерываний от системного таймера выполняет следующую задачу: инициализирует диспетчеризацию потоков добавлением соответствующего объекта в очередь DPC.

1.2 Функции обработчика прерывания от системного таймера в защищенном режиме для ОС семейства Unix/Linux

По тику обработчик прерываний от системного таймера выполняет следующие задачи:

1. Инкрементирует счетчик тиков аппаратного таймера.
2. Декрементирует квант текущего потока.
3. Инкрементирует часы и другие таймеры системы.
4. Инкрементирует счетчик использования процессора текущим процессом.
5. Декрементирует счетчик времени до отправления отложенных вызовов на выполнение (в случае обнуления счетчика для обработчика отложенного вызова устанавливается флаг).

По главному тику обработчик прерываний от системного таймера выполняет следующие задачи.

1. Регистрирует отложенные вызовы функций, относящихся к работе планировщика (например, пересчет приоритетов) Пояснение: регистрация (в системе SVR4) осуществляется с помощью вызова timeout(), которому передается функция, чей запуск откладывается: int to_ID = timeout(void (*fn)(), caddr_t arg, long delta).
2. Декрементирует квант текущего потока.

3. В нужные моменты пробуждает системные процессы (такие как swapper и pagedaemon), а именно инициализирует отложенный вызов процедуры wakeup, которая перемещает дескрипторы процессов из списка «спящих» в очередь готовых к выполнению.

4. Декрементирует счетчик времени, оставшегося до посылки ядром одного из сигналов: SIGALRM (посылается процессу по истечении заданного промежутка действительного времени), SIGPROF (используется будильником профиля процессора), SIGVTALRM (используется будильником виртуального времени).

По кванту обработчик прерываний от системного таймера выполняет следующую задачу: посылает текущему процессу сигнал SIGXCPU, в случае если тот превысил выделенную ему квоту использования процессора.

2 | Пересчет динамических приоритетов

В операционных системах семейства UNIX/Linux и семейства Windows динамически пересчитываются только приоритеты пользовательских процессов. В ОС разных семейств пересчет приоритетов выполняется поразному.

2.1 Пересчет динамических приоритетов для ОС семейства Windows

В Windows при создании процесса ему назначается приоритет – это базовый приоритет. Потокам процесса назначается относительный приоритет (относительно базового приоритета процесса).

Планирование осуществляется на основании приоритетов потоков, готовых к выполнению. Поток с более низким приоритетом вытесняется планировщиком, когда поток с более высоким приоритетом становится готовым к выполнению. По истечении кванта времени текущего потока ресурс передается первому – самому приоритетному - потоку в очереди готовых на выполнения. Один раз в секунду системный поток, называемый диспетчером настройки баланса (balance set manager), сканирует очередь готовых потоков в поиске тех из них, которые находятся в состоянии ожидания. Если обнаружены потоки, ожидающие выполнения более 4 секунд, 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.

Однако, диспетчер набора балансировки сканирует не все потоки, находящиеся в состоянии готовности. Для минимизации времени центрального процессора, затрачиваемого на его работу, он сканирует только 16 готовых потоков. Если на данном уровне приоритета имеется больше потоков, он запоминает то место, на котором остановился, и начинает с него при следующем проходе очереди. Кроме того, он за один проход повысит приоритет только 10 потоков. Если найдет 10 потоков, приоритет которых нужно повысить (что свидетельствует о необычно высоко загруженной системе), он прекратит сканирование на этом месте и начнет его с этого же места при следующем проходе.

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.

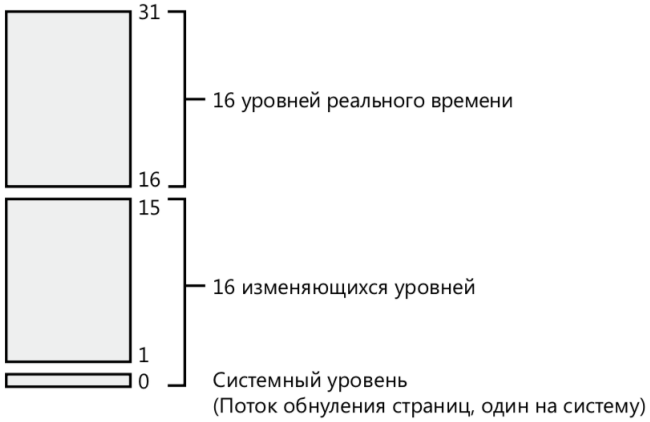

Рис 2.1: Уровни приоритета потоков в Windows

Уровни приоритета потоков назначаются Windows API и ядром Windows.

При создании процесса ему присваивается базовый приоритет (по классу которого Windows API систематизирует процессы):

- реального времени — Real-time (4);
- высокий — High (3);

- выше обычного — Above Normal (7);
- обычный — Normal (2);
- ниже обычного — Below Normal (5);
- простоя — Idle (1).

После того как Windows API систематизирует процессы по классу приоритета, присвоенного им при создании, отдельным потокам внутри процесса назначается относительный приоритет:

- 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).

Числа в списке выше представляют собой приращение, применяемое к базовому приоритету процесса.

Таким образом, в Windows API каждый поток имеет базовый приоритет, являющийся функцией класса приоритета процесса и его относительного приоритета процесса.

Отображение базовых приоритетов процессов на относительные приоритеты отдельных потоков внутри этих процессов приведено на рис. 2.2, где *насыщение* - это уровень, критичный по времени, и уровень простоя.

Класс приоритета/ Относительный приоритет	**Realtime**	**High**	**Above**	**Normal**	**Below Normal**	**Idle**
Time Critical (+ насыщение)	31	15	15	15	15	15
Highest (+2)	26	15	12	10	8	6
Above Normal (+1)	25	14	11	9	7	5
Normal (0)	24	13	10	8	6	4
Below Normal (–1)	23	12	9	7	5	3
Lowest (–2)	22	11	8	6	4	2
Idle (– насыщение)	16	1	1	1	1	1

Рис 2.2: Отображение приоритетов ядра Windows на Windows API

Текущий приоритет потока (в динамическом диапазоне от 1 до 15) может быть повышен по причинам, приведенным ниже.

1. Событие планировщика или диспетчера.
2. Повышение приоритета владельца блокировки.
3. Завершение ввода/вывода (рис. 2.3).
4. Ввода из пользовательского интерфейса.
5. Длительное ожидание ресурса исполняющей системы.
6. Ожидания объекта ядра.
7. Готовый к выполнению поток не был запущен в течение длительного времени.
8. Повышение приоритета проигрывания мультимедиа службой планировщика MMCSS.

Устройство	**Повышение приоритета**
Жесткий диск, привод компакт-дисков, параллельный порт, видеоустройство	1
Сеть, почтовый слот, именованный канал, последовательный порт	2
Клавиатура, мышь	6
Звуковое устройство	8

Рис 2.3: Рекомендуемые значения повышения приоритета

Текущий приоритет потока в динамическом диапазоне может быть понижен до базового приоритета путем вычитания всех повышений.

Мультимедийные потоки должны выполняться с минимальными задержками. Эта задача решена в Windows путем повышения приоритетов мультимедийных потоков драйвером MultiMedia Class Scheduler Service (**MMCSS**). Приложения, реализующие воспроизведение мультимедийного контента, указывают драйверу MMCSS такие задачи, как:

- аудио;

- захват;
- распределение;
- игры;
- проигрывание;
- аудио профессионального качества;
- задачи администратора многооконного режима.

Одно из наиболее важных свойств для планирования потоков называется категорией планирования — Scheduling Category, которое является первичным фактором, определяющим приоритет потоков, зарегистрированных с MMCSS. На рис. 2.4 представлены различные 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 продолжится, только если не будут готовы к выполнению другие потоки с более высоким уровнем приоритета

Рис 2.4: Рекомендуемые значения повышения приоритета

Механизм, положенный в основу MMCSS, повышает приоритет потоков внутри зарегистрированного процесса до уровня, соответствующего их категории планирования и относительного приоритета внутри этой категории на гарантированный срок. Затем он снижает категорию этих потоков до Exhausted, чтобы другие, не относящиеся к мультимедийным приложениям потоки, также получили шанс на выполнение.

2.2 Пересчет динамических приоритетов для ОС семейства Unix/Linux

Ядро современных ОС семейства Unix/Linux является вытесняющим для поддержки процессов реального времени, таких как аудио и видео. Это означает, что процесс в режиме ядра может быть вытеснен более приоритетным процессом в режиме ядра.

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.

Приоритет представляет собой целое число из диапазона от 0 до 127. Чем меньше число, тем выше приоритет. В диапазоне *0-49* находятся приоритеты ядра, являющиеся фиксированными величинами. В диапазоне *50-127* – приоритеты прикладных задач, которые могут изменяться во времени в зависимости от факторов, описанных ниже.

1. Фактор любезности - целое число в диапазоне от 0 до 39 со значением 20 по умолчанию. Чем меньше значение фактора любезности, тем выше приоритет процесса. Фактор любезности процесса может быть изменен суперпользователем с помощью системного вызова nice.

2. Величина использования процессора, измеренная в момент последнего обслуживания им процесса.

Поля структуры *proc*, относящиеся к приоритетам, представлены на рис. 2.5.

p_pri	Текущий приоритет планирования
p_usrpri	Приоритет режима задачи
p_cpu	Результат последнего измерения использования процессора
p_nice	Фактор «любезности», устанавливаемый пользователем

Рис 2.5: Структура proc

Когда процесс находится в режиме задачи, значения p_pri и p_usrpri равны. При пробуждении после блокирования в системном вызове значение текущего приоритета p_pri будет повышено для выполнения процесса в режиме ядра. Тогда планировщик использует p_usrpri для хранения приоритета, который будет назначен при возврате в режим задачи, а p_pri - для хранения временного приоритета для выполнения в режиме ядра.

Ядро системы связывает приоритет сна с событием или ожидаемым ресурсом, из-за которого процесс может блокироваться. Приоритет сна - это число в диапазоне от 0 до 49, так как он является величиной, определяемой для ядра. Когда процесс просыпается после блокирования в системном вызове, ядро устанавливает в поле p_pri приоритет сна события или ресурса. Значения приоритета сна в зависимости от события в системе 4.3BSD продемонстрированы на рис. 2.6.

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

Рис 2.6: Приоритеты сна в системе 4.3BSD

Поле *p_cpu* структуры proc, инициализируемое нулем при создании процесса, содержит величину результата последнего сделанного измерения использования процессора процессом. На каждом тике p_cpu инкрементируется обработчиком таймера на единицу для текущего процесса до максимального значения (127). Процедуру schedcpu(), вызываемая

ядром каждую секунду, уменьшает значение p_cpu каждого процесса в зависимости от *фактора "полураспада"* (decay factor). В системе 4.3 BSD значение этого фактора не фиксировано, и для его расчета применяется формула 2.1.

$$decay = (2 * load\_average)/(2 * load\_average + 1) \tag{2.1}$$

где load_average - это среднее количество процессов, находящихся в состоянии готовности к выполнению, за последнюю секунду. Процедура schedcpu() также пересчитывает приоритеты для режима задачи всех процессов по формуле 2.2.

$$p\_usrpri = PUSER + (p\_cpu/4) + (2 * p\_nice) \tag{2.2}$$

где PUSER - это базовый приоритет в режиме задачи, равный 50.

Очевидно, что чем больше процессорного времени использовал процесс, тем больше будет его p_cpu, а, следовательно, и p_usrpri, что приведет к снижению приоритета. С другой стороны, чем дольше процесс находится в очереди на выполнение, тем меньше будет его p_cpu благодаря фактору полураспада, и его приоритет будет повышаться. Такая схема позволяет предотвратить зависание низкоприоритетных процессов по вине ОС.

3 | Вывод

Среди функций обработчика прерывания от системного таймера в защищенном режиме в ОС семейств Unix/Linux и Windows можно выделить следующие пункты, являющиеся общими для обоих семейств:

- инициализирует отложенные действия, относящиеся к работе планировщика (например, пересчет приоритетов);
- декрементирует счетчики времени (например: часы, счетчик времени отложенных действий);
- декрементирует квант процессорного времени.

Как ОС семейства Unix/Linux, так и ОС семейства Windows являются системами разделения времени с вытеснением и динамическими приоритетами, пересчет которых осуществляется следующим обрзаом:

- в Windows при создании процесса ему назначается базовый приоритет. Потокам процесса назначается относительный приоритет относительно базового приоритета процесса. Приоритет потока пользовательского процесса может быть пересчитан динамически.
- в Unix/Linux приоритет процесса описывается текущим приоритетом и приоритетом процесса в режиме задачи. Приоритет процесса в режиме задачи может быть динамически пересчитан исходя из фактора любезности и величины использования процессора. Приоритеты ядра, наоборот, являются фиксированными величинами.