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"

Студент Сукочева Алис
Группа ИУ7-53Б
Название предприятия МГТУ им. Н. Э. Баумана, каф. ИУ7
Тема Функции таймера в защищенном режиме в ОС Unix и Windows

Студент: _____ Сукочева А.
подпись, дата Фамилия, И.О.
Преподаватель: _____ Рязанова Н.Ю.
подпись, дата Фамилия, И. О.

Москва — 2020 г.

Функции обработчика прерывания от системного таймера в защищенном режиме в Unix

а) по тику

— инкрементирует счетчик тиков аппаратного таймера;

— инкрементирует часы и другие таймеры системы;

— декрементирует счетчик времени до отправления на выполнение отложенного вызова;

— инкрементирует счетчик использования процессора текущим процессом;

— декременирует квант текущего потока.

б) по главному тику

— регистрирует отложенные вызовы (см. рисунок 0.1) функций , относящиеся к работе планировщика;

— регистрирует отложенный вызова (см. рисунок 0.1) процедуры wakeup, которая перемещает дескрипторы процессов из очереди «спящих» в очередь «готовых к выполнению»;

— декрементирует счетчик времени, оставшегося до отправления одного из сигналов:

SIGALARM - сигнал, посылаемый процессу по истечении времени;

SIGPROF - сигнал, посылаемый процессу по истечении времени заданном в таймере профилирования;

SIGVTALARM - сигнал, посылаемый процессу по истечении времени, заданного в «виртуальном» таймере.

в) по кванту

— посылает текущему процессу сигнала SIGXCPU, если он израсходовал выделенный ему квант процессорного времени.

ни. Например, в системе SVR4 любая подсистема ядра может зарегистрировать отложенный вызов следующим образом:

```
int to_ID = timeout (void (*fn)(), caddr_t arg, long delta);
```

где fn() — функция ядра, которую необходимо запустить, arg — аргумент, который следует передать fn(). delta — временной интервал, через который эта функция должна быть вызвана, выраженный в тиках процессора. Ядро выполняет функцию, определенную в отложенном вызове, в системном контексте. Следовательно, такая функция не должна переходить в режим ожидания или осуществлять доступ к контексту процесса. Возвращаемое значение to_ID может быть использовано для отмены выполнения отложенного вызова:

Рисунок 0.1 — Регистрация отложенных вызовов в SVR4.

Функции обработчика прерывания от системного таймера в защищенном режиме в Windows

а) по тику

— инкрементирует счётчик системного времени;

— декрементирует остаток кванта текущего потока;

— декрементирует счетчик отложенных задач;

— ставит в очередь DPC объект диспетчера настройки баланса (этот диспетчер активизируется каждую секунду для возможной инициации событий, связанных с планированием и управлением памятью).

б) по главному тику

— Возвращает задействованный в системе объект "событие", который ожидает диспетчер настройки баланса.

в) по кванту

— инициализирует диспетчеризацию потоков путем постановки соответствующего объекта в очередь DPC.

Пересчет динамических приоритетов.

Только **приоритеты пользовательских процессов** могут динамически пересчитываться.

Пересчет динамических приоритетов в Unix.

Вытесняемое ядро означает, что процесс в режиме ядра может быть вытеснен более приоритетным процессом в режиме ядра. Это сделано для того, чтобы система могла обслуживать процессы реального времени, такие как:

а) аудио;

б) видео.

В современных системах UNIX/Linux ядро является вытесняемым.

Приоритет задается любым целым числом, лежащим в диапазоне от 0 до 127. Чем меньше такое число, тем выше приоритет. Приоритеты от 0 до 49 зарезервированы для ядра, они а являются фиксированными величинами Прикладные процессы могут обладать приоритетом в диапазоне 50-127. В первую очередь выполняются процессы с большим приоритетом, а процессы с одинаковыми приоритетами выполняются в течении кванта времени циклически друг за другом. На рисунке 0.2 приведены поля структуры *proc*, относящиеся к приоритетам.

Планировщик использует p_pri для принятия решения о том, какой процесс направить на выполнение. У процесса, находящегося в режиме задачи, значения p_pri и p_usrpri идентичны. Значение текущего приоритета p_pri может быть

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

Рисунок 0.2 — Поля структуры *proc*, относящиеся к приоритетам

повышено планировщиком для выполнения процесса в режиме ядра. В этом случае p_usrpri будет использоваться для хранения приоритета, который будет назначен процессу при возврате в режим задачи. Фактор любезности – целое число в диапазоне от 0 до 39 со значением 20 по умолчанию. Увеличение фактора любезности приводит к уменьшению приоритета процесса. Фактор любезности процесса может быть изменен суперпользователем с помощью системного вызова nice. При создании процесса поле p_cpu инициализируется нулем. На каждом тике обработчик таймера увеличивает поле p_cpu текущего процесса на единицу, до максимального значения, равного 127.

Ядро системы связывает приоритет сна с событием или ожидаемым ресурсом, из-за которого процесс может блокироваться. Когда процесс просыпается после блокирования в системном вызове, ядро устанавливает в поле p_pri приоритет сна – значение приоритета из диапазона от 0 до 49, зависящее от события или ресурса по которому произошла блокировка. На рисунке 0.3 показано событие и связанное с ним значение приоритета сна 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

Рисунок 0.3 — Системные приоритеты сна

Каждую секунду, обработчик прерывания инициализирует отложенный вызов процедуры schedcpu(), которая уменьшает значение p_cpu каждого процесса исходя из фактора ”полураспада”, который рассчитывается по формуле 0.1, где

load_ average - это среднее количество процессов, находящихся в состоянии готовности к выполнению, за последнюю секунду.

$$decay = \frac{2 \cdot load\_average}{2 \cdot load\_average + 1} \tag{0.1}$$

Процедура schedcpu() пересчитывает приоритеты для режима задачи всех процессов по формуле 0.2, где $PUSER$ - базовый приоритет в режиме задачи, равный 50.

$$p\_usrpri = PUSER + \frac{p\_cpu}{2} + 2 \cdot p\_nice \tag{0.2}$$

В результате, если процесс в последний раз, т.е. до вытеснения другим процессом, использовал большое количество процессорного времени, его p_cpu будет увеличен. Это приведет к росту значения p_usrpri и, следовательно, 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 по вине операционной системы. Ее применения предпочтительно процессам, осуществляющим много операций ввода-вывода, в противоположность процессам, производящим много вычислений.

Пересчет динамических приоритетов в Windows.

В Windows при создании процесса, ему назначается базовый приоритет. Относительно базового приоритета процесса потоку назначается относительный приоритет.

Планирование осуществляется на основании приоритетов потоков, готовых к выполнению. В Windows поток с более низким приоритетом вытесняется планировщиком, когда поток с более высоким приоритетом становится готовым к выполнению. Диспетчер настройки баланса сканирует очередь готовых потоков раз в секунду и, 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. Диспетчер настройки баланса сканирует лишь 16 готовых потоков и повышает приоритет не более чем у 10 потоков за один проход. При следующем проходе сканирование возобновляется с того места, где оно было прервано в прошлый раз. Наличие 10 потоков, приоритет которых следует повысить, свидетельствует о необычно высокой загруженности системы.

Windows использует 32 уровня приоритета (рисунок 0.4), 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.

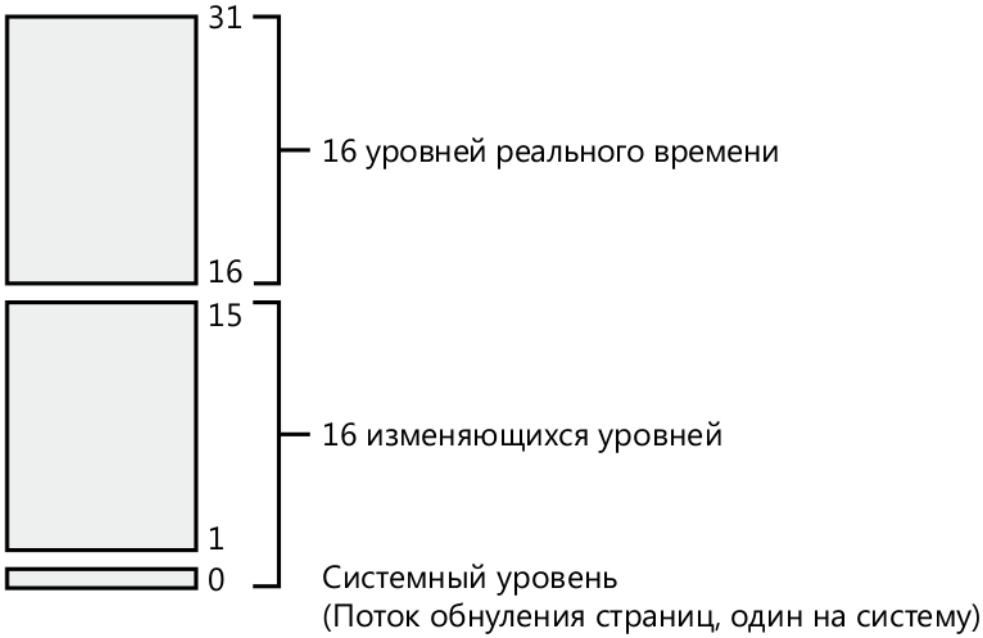

Рисунок 0.4 — Уровни приоритета потоков

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 (7), Обычный — Normal (2), Ниже обычного — Below Normal (5) и Простоя — Idle (1).

Затем назначается относительный приоритет отдельных потоков внутри этих 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) и 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 на рисунке 0.5.

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

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

Рисунок 0.5 — Соответствие между приоритетами Windows API и ядра Windows

— повышение вследствие событие планировщика или диспетчера(сокращение задержек);

— повышение приоритета владельца блокировки;

— повышение приоритета после завершения ввода/вывода (сокращение задержек) (рисунок 0.6);

— повышение приоритета вследствие ввода из пользовательского интерфейса(сокращение задержек и времени отклика);

— повышение приоритета вследствие длительного ожидания ресурса исполняющей системы(предотвращение зависания);

— повышение вследствие ожидания объекта ядра;

— повышение приоритета в случае, когда готовый к выполнению поток не был запущен в течение длительного времени (предотвращение зависания и смены приоритетов);

— 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

Рисунок 0.6 — Рекомендуемые значения повышения приоритета

MMCSS.

Повышение приоритета проигрывания мультимедиа обычно управляется службой пользовательского режима, которая называется службой планировщика

класса мультимедиа — MultiMedia Class Scheduler Service (MMCSS). MMCSS работает с вполне определенными задачами, включая следующие:

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

В свою очередь, каждая из этих задач включает информацию о свойствах, отличающих их друг от друга. Одно из наиболее важных свойств для планирования потоков называется категорией планирования — Scheduling Category, которое является первичным фактором, определяющим приоритет потоков, зарегистрированных с MMCSS. На рисунке 0.7 показаны различные категории планирования.

Механизм, положенный в основу MMCSS, повышает приоритет потоков внутри зарегистрированного процесса до уровня, соответствующего их 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

Рисунок 0.7 — Категории планирования

Затем он снижает категорию этих потоков до Exhausted, чтобы другие, не относящиеся к мультимедийным приложениям потоки, также могли получить ресурс.

Вывод

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

— инициализируют отложенные действия (такие как пересчет приоритетов);
— выполняют декремент счетчиков времени:
    а) часов;
    б) таймеров;
    в) будильников реального времени;
    г) счетчиков времени отложенных действий.
— уменьшает квант процессорного времени, выделенного процессу.

Обе операционные системы являются системами разделения времени с вытеснением и динамическими приоритетами.

Приоритет пользовательского процесса в ОС Unix/Linux может динамически пересчитываться, в зависимости от фактора любезности, p_cpu и базового приоритета. Приоритеты ядра являются фиксированными величинами.

При создании процесса в Windows, 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.

В любой системе у процесса базовый приоритет. Классическое ядро Unix не было многопоточным. Современные ядра и ядра Linux многопоточные.

Литература

1) Вахалия. UNIX изнутри.

2) Соломон, Руссинович. Внутреннее устройство Windows.