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, искусственный интеллект и системы управления**»

КАФЕДРА «**Программное обеспечение ЭВМ и информационные технологии**»

Отчет по лабораторной работе №1, часть 2

по курсу «Операционные системы»

«Обработчик прерывания системного таймера и пересчет динамических приоритетов»

Студент группы **ИУ7-54Б** _____ **Чепиго Д.С.**

(Подпись, дата) (И.О. Фамилия)

Преподаватель _____ **Рязанова Н.Ю.**

(Подпись, дата) (И.О. Фамилия)

***2022** г.*

Содержание

1 Функции обработчика прерывания от системного таймера

1.1 Операционный системы семейства UNIX

1.1.1 По тику

Функции:

— инкремент счётчика реального времени;

— декремент кванта текущего потока;

— декремент счетчиков времени до отправления на выполнение отложенных вызовов, при достижении счётчиков нуля происходит установка флага обработчика отложенных вызовов;

— инкремент счётчика процессорного времени, полученного процессом в режиме задачи и в режиме ядра.

1.1.2 По главному тику

Функции:

— инициализация отложенных вызовов функций, относящихся к работе планировщика;

— пробуждение системных процессов *swapper* и *pagedaemon*;

— декремент счетчика времени до отправления одного из следующих сигналов:

  – *SIGALRM* - сигнал, посылаемый процессу по истечении времени, которое предварительно задано функцией *alarm()*;

  – *SIGPROF* - сигнал, посылаемый процессу по истечении времени, которое задано в таймере профилирования;

  – *SIGVTALRM* - сигнал, посылаемый процессу по истечении времени, которое задано в «виртуальном» таймере.

1.1.3 По кванту

Посылка сигнала *SIGXCPU* текущему процессу, если он превысил выделенный ему квант использования процессора.

1.2 Операционные системы семейства Windows

1.2.1 По тику

Функции:

— инкремент счетчика реального времени;

— декремент кванта текущего потока;

— декремент счетчиков отложенных задач.

1.2.2 По главному тику

Освобождение объекта «событие», который ожидает диспетчер настройки баланса, который, по событию от таймера, сканирует очередь DPC (Deferred procedure call, отложенный вызов процедуры) и повышает приоритет процессов, которые находились в состоянии ожидания дольше 4 секунд.

1.2.3 По кванту

Инициализация диспетчеризации потоков – постановка соответствующего объекта в очередь DPC.

2 Пересчет динамических приоритетов

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

2.1 UNIX

Планирование процессов в UNIX основано на приоритете процесса. Планировщик всегда выбирает процесс с наивысшим приоритетом. Приоритеты планирования пользовательских процессов изменяются с течением времени, то есть динамически системой, в зависимости от использования вычислительных ресурсов, времени ожидания запуска и текущего состояния процесса.

В современных системах UNIX ядро является вытесняющим – то есть процесс в режиме ядра может быть вытеснен более приоритетным процессом в режиме ядра. Это сделано, 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 по правилу чем меньше число, тем выше приоритет. При этом числа в диапазона от 0 до 49 являются зарезервированными для ядра, то есть они фиксированы. Числа в диапазона от 50 до 127 являются приоритетами прикладных задач (приоритеты которых могут изменяться во времени).

Изменение приоритета прикладных задач зависит от двух факторов: фактор «любезности» и результат последней измеренной величины использования процессора. Фактор «любезности» – целое число в диапазоне от 0 до 39 со зна-

чением 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 этого фактора.

Дескриптор процесса UNIX BSD struct 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 (повышено) планировщиком для того, чтобы выполнить процесс в режиме ядра. В таком случае **p_usrpri** будет использоваться для хранения приоритета, который будет назначен процессу, когда тот вернется в режим задачи.

p_cpu инициализируется нулем при создании процесса (и на каждом тике обработчик таймера увеличивает это поле 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 **p_pri**, равное приоритету сна события или ресурса, по которому произошла блокировка (значение приоритета сна для некоторых событий в системе 4.3 BSD представлены в таблице 1).

Таблица 1 – Приоритеты сна в операционной 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	ый 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
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 1)

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

где *load_average* - это среднее количество процессов, находящихся в состоянии готовности к выполнению, за последнюю секунду.

Также процедура schedcpu() пересчитывает приоритеты для режима задачи всех процессов по формуле 2.

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

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

Таким образом, если процесс в последний раз использовал большое количество процессорного времени, то его p_cpu будет увеличен, следовательно будет рост значения p_usrpri и понижение приоритета. Чем дольше процесс простаивает в очереди на выполнение, тем больше фактор полураспада уменьшает его p_cpu и следовательно повышение 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 динамический пересчёт приоритетов низкоприоритетных процессов позволяет избежать бесконечного откладывания.

2.2 Windows

В Windows процессу при создании назначается базовый приоритет. Процесс по умолчанию наследует свой базовый приоритет у того процесса, который его создал. Относительно базового приоритета процесса потоку назначается относительный приоритет.

Планирование осуществляется только на основании приоритетов потоков, готовых к выполнению: если поток с более высоким приоритетом становится готовым к выполнению, поток с более низким приоритетом вытесняется планировщиком. По истечению кванта времени текущего потока, ресурс передается самому приоритетному потоку в очереди готовых к выполнению.

В Windows используется 32 уровня приоритета - целое число от 0 до 31:

— 0 – зарезервирован для процесса обнуления страниц;
— от 0 до 15 – динамически изменяющиеся уровни;
— от 16 до 31 – уровни реального времени;
— 31 – наивысший приоритет.

Уровни приоритета потоков назначаются в зависимости от двух разных позиций: Windows API и 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);
— уровень простоя (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);
— самый низший (Lowest) - (–2);
— уровень простоя (Idle) - (–15)

В таблице 2 показано соответствие между приоритетами Windows API и ядра системы.

Планировщик может повысить текущий приоритет потока в динамическом диапазоне (от 1 до 15) вследствие следующих причин:

— повышение приоритета владельцем блокировки;
— повышение приоритета после завершения ввода/вывода (таблица 3);
— повышение приоритета вследствие ввода из пользовательского интерфейса;
— повышение приоритета вследствие длительного ожидания ресурса исполняющей системы;
— повышение вследствие ожидания объекта ядра;
— повышение приоритета в случае, когда поток, готовый к выполнению, не был запущен в течение длительного времени;

Таблица 2 – Соответствие между приоритетами Windows API и ядра Windows

Класс приоритета	Real-time	High	Above	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 приоритета проигрывания для мультимедийных приложений (MMCSS).

Таблица 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

Диспетчер настройки баланса

В Windows включен общий механизм ослабления загруженности центрального процессора, который называется **диспетчером настройки баланса** и является частью системного потока.

Один раз в секунду этот поток сканирует очередь готовых потоков в поиске тех из них, которые находятся в состоянии ожидания около 4 секунд. Если

такой поток будет найден, диспетчер настройки баланса повышает его приоритет до 15 единиц и устанавливает квантовую цель эквивалентной тактовой частоте процессора при подсчете 3 квантовых единиц. Как только квант истекает, приоритет потока тут же снижается до обычного базового приоритета. Если поток не был завершен и есть готовый к запуску поток с более высоким уровнем приоритета, поток с пониженным приоритетом возвращается в очередь готовых потоков.

Для минимизации времени своей работы, диспетчер настройки баланса 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, заслуживающих именно этого повышения, он прекратит сканирование на этом месте и начнет его с этого же места при следующем проходе.

MMCSS

Для того, чтобы мультимедийные потоки могли выполняться с минимальными задержками, драйвер *MMCSS* (*MultiMedia Class Scheduler Service*) временно повышают приоритет потоков до уровня, соответствующего их категориям планирования (таблица 4). Для того, чтобы другие потоки могли получить ресурс, приоритет снижается до уровня, соответствующего категории *Exhausted*.

Категории планирования – первичный фактор, определяющий приоритет потоков, зарегистрированных в MMCSS:

Таблица 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 – Interrupt Request Level.

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 – Interrupt Service Routine.

После выполнения ISR диспетчер прерывания понижает IRQL процессора до исходного уровня 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.

В ядре IRQL-уровни представлены в виде номеров от 0 до 31, где более высоким номерам соответствуют прерывания с более высоким приоритетом.

Уровни IRQL:

- 31 – наивысший (например, ошибка шины) ;
- 30 – питание;
- 29 – межпроцессорное прерывание;
- 28 – таймер;
- 27 – устройство n;
- .....;
- 3 – устройство 1;
- 2 – DPC (deferred procedure call)/dispatch);
- 1 – APC (asynchronous procedure call);
- 0 – низший.

3 Вывод

Функции обработчика прерывания от системного таймера для операционных систем семейства UNIX и для семейства Windows схожи, так как они являются системами разделения времени. Схожие задачи обработчика прерывания от системного таймера:

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

Пересчет динамических приоритетов осуществляется только для пользовательских процессов, чтобы избежать бесконечного откладывания. Обе операционные системы (UNIX и Windows) – это системы разделения времени с динамическими приоритетами и вытеснением.

В UNIX приоритет пользовательского процесса (процесса в режиме задач) может динамически пересчитываться, в зависимости от трех факторов. Приоритеты ядра – фиксированные величины.

В 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.