МИНОБРНАУКИ РОССИИ
Санкт-Петербургский государственный
электротехнический университет
«ЛЭТИ» им. В.И. Ульянова (Ленина)
Кафедра «Информационные системы»

РЕФЕРАТ №1

по дисциплине «Системы реального времени»

Тема: Диспетчеризация задач

Дата представления реферата на проверку: 04.03.2023

Евсиков А.О.

Студент гр. 8361 evsik.sasha@gmail.com

Преподаватель Сидельников В.В.

Санкт-Петербург

2023

Содержание

1. Планирование и диспетчеризация 3
1.1. Определения и различия процессов и потоков 3
1.2. Задачи диспетчеризации 4
2. Планирование выполнения задач 7
2.1. Причины активации потока 7
2.2. Состояния задачи 9
3. Общие принципы управления процессами 11
3.1. Планирование с переключением и без переключения 11
3.2. Приоритеты и приоритетное управление 11
3.3. Алгоритмы планирования 12
3.4. Многоуровневые очереди с обратными связями 12
3.5. Статические алгоритмы 14
3.6. Динамические алгоритмы 15
Список используемых источников 18

1. Планирование и диспетчеризация

1.1. Определения и различия процессов и потоков

Существует несколько понятий процесса:

«Процесс – действия машины при выполнении программы»

«Процесс – программа в стадии выполнения»

«Процесс – это адресное пространство вместе с выполняемыми в нем потоками управления, а также системными ресурсами, которые этим потокам требуются» [1]

А также в QNX (POSIX-совместимая операционная система реального времени) «Процесс – это контейнер потоков»

Один или несколько потоков выполняются в контексте процесса. Поток – это базовая единица, для которой операционная система выделяет время процессора.

Процессы и потоки связаны друг с другом, но при этом имеют существенные различия.

Процесс — экземпляр программы во время выполнения, независимый объект, которому выделены системные ресурсы (например, процессорное время и память). Каждый процесс выполняется в отдельном адресном пространстве: один процесс не может получить доступ к переменным и структурам данных другого. Если процесс хочет получить доступ к чужим ресурсам, необходимо использовать межпроцессное взаимодействие. Это могут быть конвейеры, файлы, каналы связи между компьютерами и многое другое. [2]

Поток использует то же самое пространства стека, что и процесс, а множество потоков совместно используют данные своих состояний. Как

правило, каждый поток может работать (читать и писать) с одной и той же областью памяти, в отличие от процессов, которые не могут просто так получить доступ к памяти другого процесса. У каждого потока есть собственные регистры и собственный стек, но другие потоки могут их использовать.

Поток — определенный способ выполнения процесса. Когда один поток изменяет ресурс процесса, это изменение сразу же становится видно другим потокам этого процесса.

1.2. Задачи диспетчеризации

На протяжении существования процесса выполнение его потоков может быть многократно прервано и продолжено (если ОС не поддерживает многопоточную обработку, все сказанное ниже о планировании и диспетчеризации относится к процессу).

Переход от выполнения одного потока к другому осуществляется в результате планирования и диспетчеризации. Работа по определению того, в какой момент необходимо прервать выполнение текущего активного потока и какому потоку предоставить возможность выполняться, называется планированием. Планирование потоков осуществляется на основе информации, хранящейся в описателях процессов и потоков. При планировании могут приниматься во внимание приоритет потоков, время их ожидания в очереди, накопленное время выполнения, интенсивность обращений к вводу-выводу и другие факторы. ОС планирует выполнение потоков независимо от того, принадлежат ли они одному или разным процессам. Так, например, после выполнения потока некоторого процесса ОС может выбрать для выполнения другой поток того же процесса или же назначить к выполнению поток другого процесса.

Планирование потоков, по существу, включает в себя решение двух следующих задач:

1. определение момента времени для смены текущего активного потока;
2. выбор для выполнения потока из очереди готовых потоков.

Существует два типа планирования:

1. динамическое планирование;
2. статическое планирование.

В большинстве операционных систем универсального назначения планирование осуществляется динамически (on-line), т.е. решения принимаются во время работы системы на основе анализа текущей ситуации.

Другой тип планирования — статический — используется в специализированных системах, в которых весь набор одновременно выполняемых задач определен заранее, например в системах реального времени. Планировщик в этом случае называется статическим (или предварительным планировщиком), так как он принимает решения о планировании не во время работы системы, а заранее (offline).

Диспетчеризация заключается в реализации найденного в результате планирования (динамического или статического) решения, т.е. в переключении процессора с одного потока на другой. Прежде чем прервать выполнение потока, ОС запоминает его контекст, с тем чтобы впоследствии использовать эту информацию для последующего возобновления выполнения данного потока. Контекст отражает, во-первых, состояние аппаратуры компьютера в момент прерывания потока: значение счетчика команд,

содержимое регистров общего назначения, режим работы процессора, флаги, маски прерываний и другие параметры. Во-вторых, контекст включает параметры операционной среды, а именно ссылки на открытые файлы, данные о незавершенных операциях ввода- вывода, коды ошибок выполняемых данным потоком системных вызовов и т.д.

Диспетчеризация сводится к следующему:

- сохранение контекста текущего потока, который требуется сменить;
- загрузка контекста нового потока, выбранного в результате планирования;
- запуск нового потока на выполнение.

Поскольку операция переключения контекстов существенно влияет на производительность вычислительной системы, программные модули ОС выполняют диспетчеризацию потоков совместно с аппаратными средствами процессора. [3]

2. Планирование выполнения задач

2.1. Причины активации потока

Активация потока может быть вызвана несколькими факторами:

1. Синхронизация (synchronization). T1 запрашивает мьютекс, и если он занят потоком T2, то T1 встаёт в очередь, тем самым давая возможность другим потокам запуститься.
2. **Вытеснение** (**preemption**). Происходит событие, в результате которого высокоприоритетный поток T2 может запуститься. Тогда поток T1 с низким приоритетом вытесняется, и T2 запускается.
3. Уступание (yielding). Программист явно вызывает sched_yield() во время работы T1, и тогда планировщик ищет другой поток T2, который может запуститься, и запускает его.
4. **Квантование** (**time-slicing**). Квант времени для потока T1 истёк. Тогда поток T2 получает квант и запускается

К примеру, так выглядит планирование на основе квантования

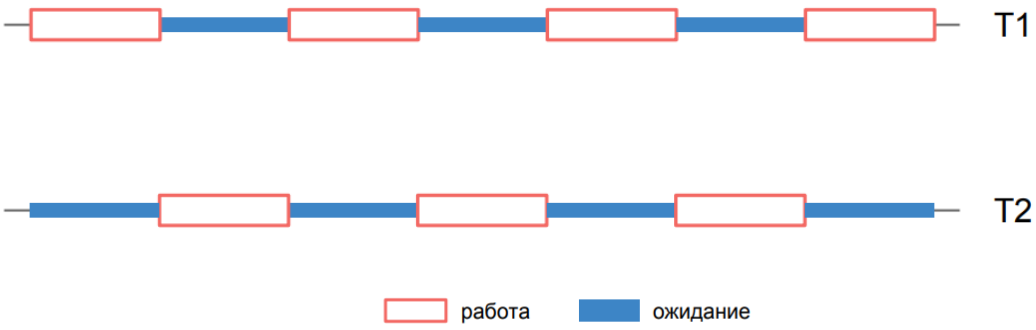

Рисунок 1 – Планирование на основе квантования

Операции по переключению процессов критичны по времени и должны осуществляться с максимальной эффективностью. На процессорах,

первоначально не разработанных для многозадачного режима, процедура сохранения и восстановления контекста – переключение процессов – реализуется длинной последовательностью стандартных инструкций процессора. В набор команд процессора, спроектированного для работы в многозадачном режиме, входят специальные инструкции для сохранения и восстановления контекста. Переменные процесса не входят в состав контекста, и сохранять их специально нет необходимости, поскольку они хранятся в памяти, выделенной процессу и защищенной операционной системой от доступа других процессов.

Планировщик вызывается обычно после обработки каждого прерывания. Если используется стратегия переключения процессов на основе квантования времени, необходимо иметь внешний по отношению к процессору интервальный таймер, вырабатывающий прерывания по истечении кванта времени (time slice), выделенного процессу. При возникновении прерывания исполнение текущего процесса приостанавливается и проверяется, должен ли быть прерван текущий процесс и загружен новый. Принудительная приостановка текущего процесса для передачи управления другому процессу называется вытеснением (preemption).

Продолжительность кванта времени влияет на производительность системы. При коротком кванте улучшается общее время обслуживания коротких процессов. Если величина кванта сопоставима с временем, необходимым для переключения процессов, то большая часть ресурсов процессора будет уходить на планирование и переключение. Если величина кванта слишком большая, увеличивается время ожидания процессов и, соответственно, время реакции. [4]

Процесс исполняется до тех пор, пока не произойдет одно из следующих событий:

- истек выделенный ему квант времени;

- процесс заблокирован, например, ждет завершения операции ввода/вывода;
- процесс завершился;
- вытеснен другим процессом, имеющим более высокий приоритет, например обработчиком прерываний.

2.2. Состояния задачи

В многозадачной среде процесс может находиться в одном из трех состояний (рис. 2).

• Готов (ready). Процесс может начать исполнение, как только освобо- дится процессор.

• Исполнение (running, executing). Процесс исполняется в данный мо- мент, т. е. процессор исполняет его код.

• Ожидание, заблокирован (waiting, locked). Для продолжения работы процессу не хватает какого-либо ресурса, за исключением ЦП, либо он ждет наступления внешнего события.

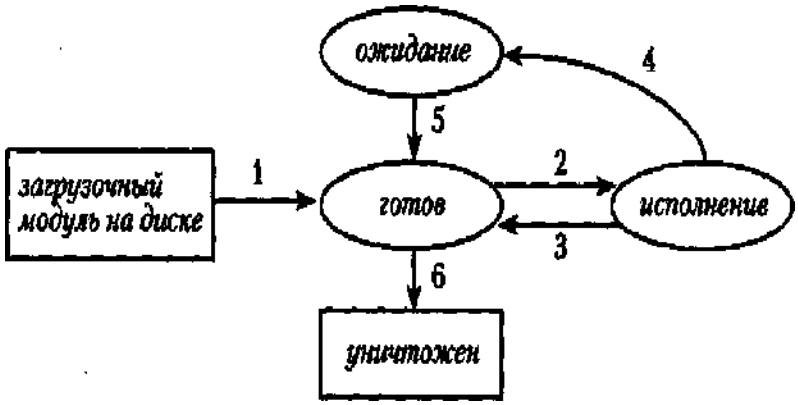

Рисунок 2 – Состояния процесса

На рисунке 2 также показаны возможные переходы из одного состояния в другое:

1. От "Загрузочный модуль на диске" к "Готов". Программа загружается (load) в оперативную память, настраиваются

относительные адреса (relocation), выдёляются рабочие области для данных, кучи и стека с соответствующими указателями и создается контекст процесса.

2. От "Готов" к "Исполнение". Планировщик выбирает первый в очереди готовых процессов и передает ему управление.
3. От "Исполнение" к "Готов". Процесс либо исчерпал свой квант времени, либо появился готовый для исполнения процесс с более высоким приоритетом.
4. От "Исполнение" к "Ожидание". Для дальнейшего развития процесс должен ждать наступления какого-либо внешнего события (завершения операции ввода/вывода или освобождения ресурса, например доступа к памяти, заблокированной другим процессом, или сигнала от другого процесса и т. п.).Иногда процесс переводится в состояние ожидания до истечения некоторого интервала времени с помощью явной инструкции в его программе.
5. От "Ожидание" к "Готов". Когда ожидаемое событие произошло или истекло заданное время, процесс переводится в состояние "Готов" и помещается в очередь готовых процессов, откуда затем выбирается планировщиком.

После выполнения последней инструкции программы операционная система удаляет процесс из памяти и освобождает все выделенные ему ресурсы, включая память. [4]

3. Общие принципы управления процессами

3.1. Планирование с переключением и без переключения

Планирование без переключения предусматривает, что после предоставления ресурсов ЦП какому-либо процессу, отобрать ЦП у этого процесса нельзя. Если же ресурсы ЦП можно отобрать, то говорят о планировании с переключением.

При использовании планирования без переключения коротким заданиям приходится больше ждать из-за выполнения длительных заданий, с другой стороны, для всех процессов создаются как бы равные условия, а времена ответа здесь более предсказуемы.

Планирование с переключением необходимо в системах, где процессы высокого приоритета требуют немедленного внимания, например в интерактивных системах разделения времени этот способ планирования позволяет гарантировать приемлемые времена ответа.

3.2. Приоритеты и приоритетное управление

Система может присваивать процессам приоритеты автоматически или они могут назначаться извне. Приоритеты могут быть заслуженными или купленными. Они могут быть статическими или динамическими. Они могут назначаться по какому-то рациональному принципу или присваиваться в ситуациях, когда системе просто необходимо каким-либо образом различать процессы.

Статические приоритеты не изменяются, такой механизм установки приоритетов достаточно прост и не сопряжен с большими издержками. Однако следует учитывать, что такой механизм недостаточно гибок, т.к. не реагирует на изменение окружающей ситуации.

Динамические приоритеты позволяют повысить реактивность системы, т.к. реагируют на изменения ситуации, и начальное значение приоритета процесса может быть изменено на новое, более подходящее значение.

Покупаемые приоритеты дают возможность пользователю повысить приоритет задания и получить более высокий уровень обслуживания за "дополнительную плату" (например, уменьшение кванта времени).

3.3. Алгоритмы планирования

Планирование по принципу FIFO (first-in-first-out)

Принцип FIFO, «первый пришедший обслуживается первым», является наиболее простой дисциплиной планирования. ЦП предоставляется процессам в порядке их прихода в очередь готовности.

После того, как процесс получает ЦП в свое распоряжение, он выполняется до завершения, т.е. это дисциплина планирования без переключения, поэтому ее не рекомендуют использовать в системах с разделением времени.

Как правило, принцип FIFO редко используется самостоятельно в качестве основной дисциплины обслуживания, чаще он комбинируется с другими дисциплинами, например, диспетчирование процессов может выполняться согласно их приоритетам, однако процессы с одинаковыми приоритетами диспетчируются по принципу FIFO. [3]

3.4. Многоуровневые очереди с обратными связями

Механизм планирования должен оказывать предпочтение коротким заданиям с лимитируемым вводом-выводом, чтобы обеспечить хороший коэффициент использования устройств ввода-вывода; как можно быстрее определять характер задания, чтобы соответствующим образом ппланировать

его выполнение. Многоуровневые очереди с обратными связями позволяют достичь этих целей. [5]

При таком алгоритме планирования, новый процесс входит в сеть очередей с конца верхней очереди и перемещается по ней по принципу FIFO пока не получит в свое распоряжение ЦП. Если процесс не успевает завершиться по истечении отведенного ему кванта времени, он перемещается в конец очереди более низкого уровня и получит в свое распоряжение ЦП, когда достигнет начала этой очереди и не будет ожидающих процессов в верхней очереди. Обычно в такой многоуровневой системе предусматривается нижняя очередь, организованная по принципу RR, где процесс циркулирует до окончательного завершения. Как правило, в таких структурах квант времени при переходе в каждую очередь более низкого уровня увеличивается.

Многоуровневые очереди с обратными связями представляют собой идеальный механизм, позволяющийцессы на категории в соответствиии с их потребностями в ресурсах ЦП. В системе с разделением времени, каждый раз, когда процесс выходит из сети очередей, он может быть отмечен признаком очереди самого низкого уровня, где он побывал, что дает возможность впоследствии, когда он снова войдет в сеть очередей, направлять его прямо в ту очередь, в которой он в последний раз завершал свою работу. Заметим, что при таком алгоритме планирования, чем дольше процесс занимает ЦП, тем ниже становится его приоритет, пока процесс не опускается в очередь самогго приоритета. Размер же кванта времени, выделяемого процессу, как правило увеличивается по мере перехода процесса в каждую следующую очередь.

Сеть мноуровневых очередей с обратными связями — это пример адаптивного механизма планирования, который реагирует на изменение поведения контролируемой им системы.

3.5. Статические алгоритмы

Примером такого алгоритма является алгоритм планирования со статическим расписанием. Он подразумевают, что расписание запуска задач составляется заранее, до старта системы. Планировщик лишь просто следует этому расписанию и не составляет его в ходе работы. Строго это относится, разумеется, только к периодическим задачам. Если же имеются еще и спорадические задачи, то планировщик, естественно, должен выбрать моменты времени запуска таких задач по ходу работы.

К преимуществам данного класса алгоритмов относят следующие обстоятельства:

Исключительная простота, обусловленная отсутствием понятия "процесс"/"поток". Передача управления задаче — вызов подпрограммы

Как следствие, результаты тестирований и поверок весьма надежны.

Благодаря этим качествам, алгоритмы именно этого типа чаще всего применяются там, где требуется высокая надежность (автопилоты и т.п.)

К недостаткам — следующие:

Негибкость. Любое изменение (числа задач, времен исполнения и т.д.) требует останова системы и пересчета расписания.

Планировщик фактически "отвязан" от внешного мира, так как работает по прерываниям от таймера. Учет спорадических задач довольно сложен.

Размер таблицы с расписанием может оказаться большим при соответствующих соотношениях между периодами задач. [5]

3.6. Динамические алгоритмы

Принцип работы планировщиков на основе приоритета задач состоит в том, что в каждый момент момент времени исполняется та задача, которая имеет наивысший приоритет. Различные виды планировщиков различаются правилами, в соответствии с которыми назначается приоритет. У работ одной и той же задачи может быть разный приоритет. В этом случае планировщик (или алгоритм) называется динамическим алгоритмом с динамическими приоритетами.

Рассмотрим 2 алгоритма динамического планирования с динамическими приоритетами

EDF (earliest deadline first) — приоритет задачам назначается по принципу "в каждый момент времени наивысший приоритет имеет та задача, у которой осталось меньше всего времени до крайнего срока".

LLF (least laxity first) — приоритет задачам назначается по принципу "в каждый момент времени наивысший приоритет имеет задача с наименьшим резервом времени (laxity)".

Резервом (запасом) времени называется разность между временем, оставшимся до крайнего срока и временем, которое задаче еще нужно проработать

При этом имеются 2 модификации алгоритма EDF:

- с вытесненением задач
- без вытеснения задач

Под вытеснением имеется ввиду то, что если во время работы какой-то задачи возникает работа другой задачи с приоритетом выше, чем у работающей в данный момент, то управление передается вновь возникшей задаче.

При невытесняющем EDF задача всегда доделывает свою очередную работу до конца, независимо от того, появились ли во время работы этой задачи другие задачи с более высоким приоритетом или не появились.

Алгоритмы EDF и LLF считаются оптимальными для любого набора задач (периодические с любым соотношением между периодами и крайними сроками, спорадические) если:

- все задачи независимы
- возможно вытеснение
- вытеснение не требует временных затрат (что не соответствует реальному положению вещей)

Другим классом динамических алгоритмов являются динамические алгоритмы планирования со статическими приоритетами. Напомним, что принцип работы любого планировщика на основании приоритета состоит в том, что в каждый момент времени исполняется та задача, которая имеет наивысший приоритет. В случае динамических планировщиков со статическими приоритетами задач приоритет задачи, будучи однажды ей назначен, не изменяется с течением времени. Приводимые далее способы назначения приоритетов ориентированы на системы периодических задач.

Существует 2 распространенных способа назначения приоритетов:

RMS (rate monotonic scheduling). Правило назначения приоритетов таково: чем меньше период задачи, тем выше у нее приоритет. Иными словами, чем чаще (отсюда rate) задача переходит в состояние готовности, тем ее приоритет выше.

DMS (deadline monotonic scheduling). В этом алгоритме приоритеты назначаются по немного другому правилу: чем меньше относительный крайний срок задачи, тем выше ее приоритет.

Эти алгоритмы неоптимальны, так как при определенных условиях могут привести к превышению дедлайна, однако ввиду их относительной простоты именно они чаще используются в системах реального времени [5]

Список используемых источников

1. Лекционные слайды (Лекция 7 «Процессы») vec.etu.ru
2. McDowell G.L. Cracking the Coding Interview 6 издание в переводе Е. Матвеева / Питер, 2016
3. Рудаков А.В. Операционные системы и среды / ИНФРА-М, 2019
4. Лекции по дисциплине «Системы реального времени» vec.etu.ru
5. Анатольев А.Г Управление процессами. Диспетчеризация / 2012