Лекция 6: Планирование – Scheduling

Алексей Линёв
Александр Мощук
Кирилл Погорельский

somes slides are adapted from the OS course at the University of Washington

Объявления

- Проект №1
  - сдача сегодня вечером
- Расписание:
  - Семинар по проекту №2 завтра (9:00)
  - Лекция №7 (синхронизация) в пятницу
- Сегодня:
  - Немного о проекте №0
  - Планирование

Планирование

- При обсуждении процессов и потоков мы говорили о переключении контекстов
- Мы обошли вопрос выбора следующего процесса или потока на исполнение
  - "выбирается поток из очереди готовых к выполнению"
- Принятие такого решения называется *планированием*
  - планирование – это стратегия (policy)
  - переключение контекстов – используемый механизм

Планирование

- Сегодня:
  - Стратегии планирования
    - Кого выбрать на исполнение и на какое время?
  - Объекты планирования: *задачи*
    - Процессы, потоки, люди, перемещение головок диска, ...
- *Планировщик* передвигает задачи из очереди в очередь
  - *Алгоритм планирования* определяет выбор задачи на исполнение, выбор очереди для ожидания
  - Планировщик запускается когда:
    - Исполняемая задача переходит в состояние ожидания
    - Происходит прерывание
      - особенно прерывание таймера
    - Создается или уничтожается новая задача
- Много классов планирования...
  - Пакетный (batch), интерактивный, реального времени, параллельный
  - В основном мы будем обсуждать интерактивные планировщики

Уровни принятия решений планирования

- Долговременное планирование
  - Нужно ли запустить очередную задачу на исполнение или нет?
  - Характерно для систем пакетной обработки
  - По какой причине вы можете решить не запускать задачу?
- Средневременное планирование
  - Нужно ли временно пометить выполняющуюся программу "не запускаемой" (например, при выгрузке ее на диск – swapped out)
- Краткосрочное планирование
  - Какому потоку передать ЦП? На какое время?
  - Какая операция ввода-вывода на диск должна быть выполнена следующей?

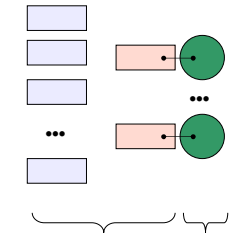
Критерии оценки I: Производительность

- Существуют различные метрики, оценивающие производительность (иногда – противоречивые)
  - эффективность, CPU utilization (→ MAX)
    - средний процент загруженности ЦП
  - пропускная способность, throughput (→ MAX)
    - среднее число выполненных задач за единицу времени
  - оборотное время, turnaround time (→ MIN)
    - среднее время от момента получения задачи до ее выполнения
  - время ожидания, waiting time (→ MIN)
    - среднее время, проведенное на очереди ожидания
  - время отклика, response time (→ MIN)
    - среднее время, проведенное на очереди готовности
  - потребляемая энергия (Дж/инструкция, → MIN)
    - причина для дополнительных ограничений

Критерии оценки II: Справедливость

- Не существует общепринятого определения "справедливости"
  - Как измерить справедливость?
    - Одинаковое время потребления ЦП? (по какой временной шкале?)
  - Честно на уровне пользователей? процессов? потоков?
  - Что если один процесс ограничен возможностями ЦП, а другой – возможностями ввода-вывода?
- Иногда нужно быть несправедливым:
  - Явно оказывать предпочтение некоторым определенным классам запросов/событий (иметь систему приоритетов), но…
  - при этом избегать *голодания* (*starvation*) – быть уверенным в том, что все запросы получат некоторое обслуживание

Типичная ситуация



Планирование:
- Кому предоставить каждый из ресурсов
- Когда пересмотреть свое решение

Объекты планирования Ресурсы

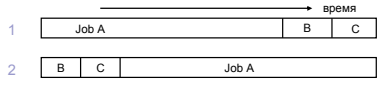
Когда принимать решение?

- Вытесняющее и невытесняющее планирование
  - Невытесняющее (non-preemptive)
    - когда ЦП выделяется кому-либо, он может использовать его до тех пор, пока не вернет его добровольно
      - например, выполнит запрос на ввод-вывод
  - Вытесняющее (preemptive)
    - вы можете пересмотреть решение
      - установка таймера позволяет получить ЦП у потока даже если тот не пожелал отдать его добровольно
    - пересмотр решения всегда связан с некоторыми дополнительными расходами
      - и эти расходы отрицательны при использовании любого критерия оценки алгоритма планирования
- Мы попытаемся найти сбалансированные стратегии
  - ресурсы не должны простаивать, если кто-то хочет их использовать

Алгоритм №1: FCFS/FIFO

- First-come first-served / First-in first-out (*FCFS/FIFO*)
  - выделение ресурса потокам производится в соответствии с порядком поступления запросов
  - совсем как общепринятое обслуживание людей в порядке очереди
    - кассиры в супермаркетах, банках, McDonald's,…
  - как правило, невытесняющий алгоритм
    - в супермаркетах нет переключений контекста!
  - все задачи равноправны, никакого голодания
    - в каком смысле это честно?
- Звучит великолепно!
  - в реальном мире – когда FCFS/FIFO работает хорошо?
    - даже в таких случаях – есть ли какие-либо ограничения?
  - а когда данный алгоритм работает плохо?

Пример FCFS/FIFO



- Предположим, что продолжительность A - 5, B и C – по 1
  - среднее оборотное время для расписания 1 (в предположении, что запросы A, B и C поступили в момент времени 0) равно (5+6+7)/3 = 18/3 = 6
  - среднее оборотное время для расписания 2 равно (1+2+7)/3 = 10/3 = 3.3
  - введем характеристику "относительная задержка" – субъективный фактор
    - Расписание 1: A: 5/5, B: 6/1, C: 7/1 (худшее - 7, среднее - 4.7)
    - Расписание 2: A: 7/5, B: 1/1, C: 2/1 (худшее - 2, среднее - 1.5)

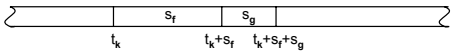
Недостатки FCFS/FIFO

- Среднее время ожидания может быть ужасно
  - быстрые обработчики ожидают завершения медленных
- Может привести к недостаточному использованию остальных ресурсов
  - если вы позволите мне двигаться, я смогу загрузить незанятые ресурсы
  - использование FCFS может привести к недостаточному перекрытию вычислительной активности и использования ввода-вывода

Алгоритм №2: SJF

- Shortest job first (*SJF*)
  - на выполнение выбирается запрос с наименьшими требованиями к обслуживанию
- Оптимальный с точки зрения среднего времени ожидания

Оптимальность SJF (доказательство)



- При любом планировании, отличном от SPT/SJF, существуют пары запросов f и g, для которых время обслуживания запроса f (равное s_f), превышает время обслуживания g (равное s_g)
  - т.е. $s_f > s_g$
- Полный вклад в среднее время ожидания от f и g равен $2t_k+s_f$
- Если поменять местами f и g, их суммарный вклад станет равен $2t_k+s_g$, что меньше, так как $s_g < s_f$

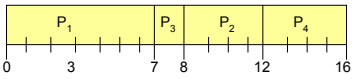
Недостатки SJF

- Невытесняющий алгоритм
- Но существует вытесняющая версия – SRPT (Shortest Remaining Processing Time first) – учитывающая появление новых запросов (а не предполагающая, что все запросы обрабатываются целиком)
  - Возможно, что только что появившийся запрос выполнится быстрее, чем оставшая часть текущего запроса
- Имеет ли место голодание?
- Можно ли узнать время обработки запроса?
- Можно ли его оценить/предсказать? как?

Пример: SJF

Process	Arrival Time	Burst Time
P1	0.0	7
P2	2.0	4
P3	4.0	1
P4	5.0	4

SJF (без вытеснения)

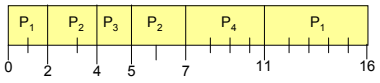


Среднее время ожидания = (0 + 6 + 3 + 7)/4 = 4

Пример: SJF

Process	Arrival Time	Burst Time
P1	0.0	7
P2	2.0	4
P3	4.0	1
P4	5.0	4

SRPT (SJF с вытеснением)



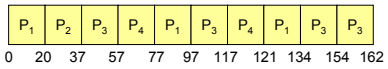
Среднее время ожидания = (9 + 1 + 0 +2)/4 = 3

Алгоритм #3: RR

- Round Robin (RR)
  - очередь готовых к выполнению обрабатывается циклически
  - каждому запросу выделяется интервал времени, называемый *квантом* (*quantum*)
    - запрос выполняется до истечения кванта или выполнения блокирующего вызова
      - какое событие уведомляет о завершении кванта?
    - мультиплексирование ЦП посредством разделения времени (квантования)
  - по определению подходит для систем разделения времени
    - и нет голодания!
- Звучит отлично!
  - в чем RR превосходит FCFS?
  - в чем RR превосходит SJF?
  - какие имеются недостатки?

Пример: RR (quantum = 20)

Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24



Недостатки RR

- Как выбирать размер кванта?
  - нет "правильного" значения
    - если квант мал, часто будут выполняться переключения контекста, давая дополнительные затраты
    - если квант велик – ухудшается время отклика
      - → FIFO
  - все задачи считаются равнозначными
    - если я запущу 100 экземпляров quake, качество обслуживания вашим сервисами существенно ухудшится
    - как это можно исправить?

Алгоритм №4: Приоритетное планирование

- Запросам присваиваются приоритеты
  - всегда на исполнение выбирается запрос с наивысшим приоритетом
    - для запросов одного приоритета используется другой алгоритм планирования (например RR)
  - для реализации SJF можно использовать приоритет, равный ожидаемому времени потребления ЦП
- Теоретически моделируется (и обычно практически реализуется) в виде множества "очередей приоритетов"
  - запрос, готовый к выполнению, помещается в конец очереди для своего приоритета

Недостатки приоритетного планирования

- Как вы собираетесь назначать приоритеты?
- Голодание
  - если постоянно существуют высокоприоритетные запросы в состоянии "готов к выполнению", низкоуровневые запросы могут никогда не выполниться
- Решение: организовать "устаревание" запросов
  - увеличивать приоритет в зависимости от накопленного времени ожидания
  - уменьшать приоритет в зависимости от количества уже потребленного времени ЦП
  - к настоящему моменту существует масса эвристических алгоритмов изменения приоритетов

Комбинированные алгоритмы

- На практике, каждая реальная система использует некоторый комбинированный подход с элементами FCFS, SJF, RR и приоритетного планирования
- Пример: многоуровневые очереди с обратной связью (multi-level feedback queues, *MLFQ*)
  - это иерархия очередей
  - очереди упорядочены по приоритетам
  - новый запрос поступает в очередь с наивысшим приоритетом
  - внутри каждой очереди реализован алгоритм RR
  - очереди имеют разный размер кванта
  - запросы перемещаются между очередями в зависимости от своей истории выполнения
  - В каких ситуациях MLFQ становится похож на SJF?

Планирование в UNIX

- Оригинальный алгоритм планирования сильно похож на MLFQ
  - 3-4 класса приоритетов, включающие ~170 уровней приоритетов
    - разделения времени: низшие 60 приоритетов
    - системные: средние 40 приоритетов
    - реального времени: высшие 60 приоритетов
  - приоритетное планирование между очередями, RR – внутри очередей
    - процесс с наивысшим приоритетом всегда выполняется первым
    - процессы с одинаковым приоритетом планируются по RR
  - у процессов динамически изменяются приоритеты
    - увеличиваются со временем, если процесс перешел в состояние ожидания до завершения своего кванта
    - уменьшаются, если процесс потребил свой квант целиком
- Цель:
  - дать интерактивным задачам преимущество перед задачами, просто потребляющим ЦП
    - интерактивные процессы, как правило, потребляют время ЦП малыми порциями

Выводы

- Существует несколько уровней планирования
- Выбор алгоритма планирования очень сильно отражается на производительности
  - и разница увеличивается при изменяющихся требованиях сервисов, обрабатывающих запросы
- Имеется набор критериев оценки алгоритмов планирования, в некоторых случаях – противоречащих друг другу
- Существует множество "чистых" алгоритмов, на практике обладающих рядом недостатков – FCFS, SJF, RR, приоритетный
- В реальных системах используются комбинированные алгоритмы