Реализации многозадачности

«Операционные системы»

Д.В. Иртегов

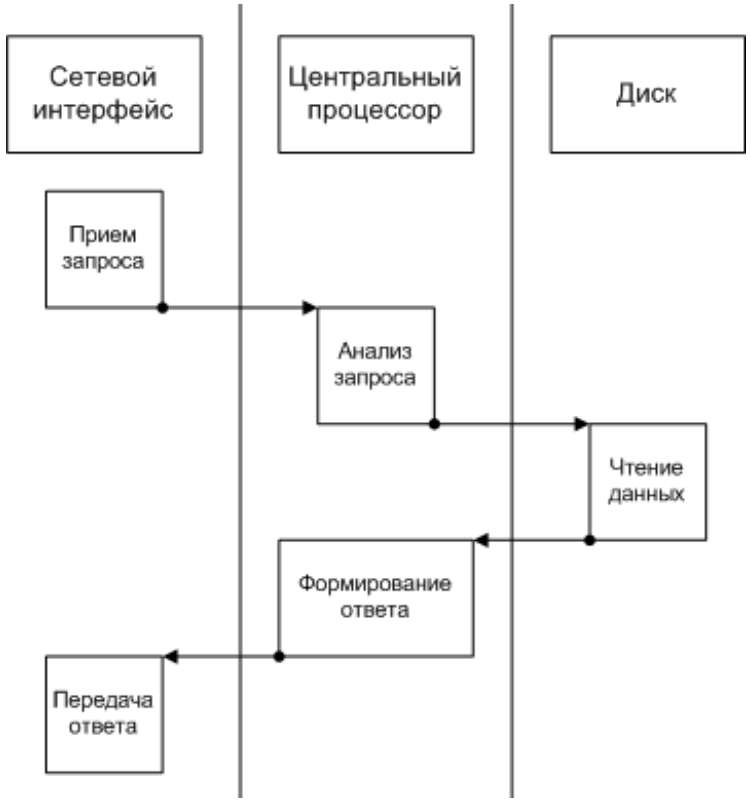
ФИТ/ФФ НГУ

2020

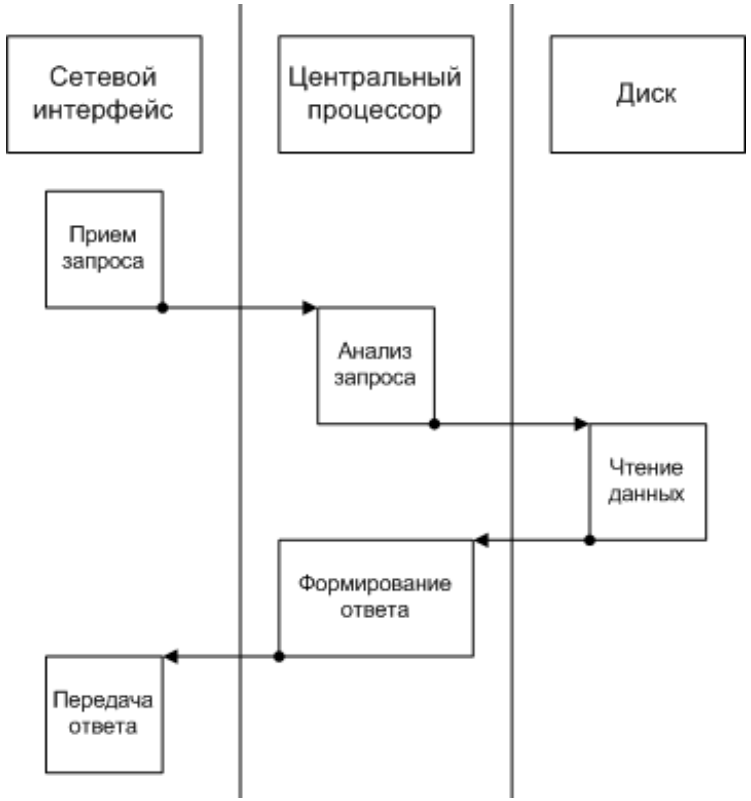
Зачем нужна многозадачность на однопроцессорной машине?

- Большая часть прикладных программ ориентирована на ввод-вывод
  - пользовательские интерфейсы
  - сеть
  - реальное время
  - базы данных
- Устройства ввода-вывода медленные
- Устройства ввода-вывода могут работать параллельно с ЦПУ
- ЦПУ можно чем-то занять, пока приложение ждет ввода

Ввод-вывод


Сетевой интерфейс
Центральный процессор
Диск
Прием запроса
Анализ запроса
Чтение данных
Формирование ответа
Передача ответа

Ввод-вывод


Сетевой интерфейс
Центральный процессор
Диск
Прием запроса
Анализ запроса
Чтение данных
Формирование ответа
Передача ответа


Сетевой интерфейс
Центральный процессор
Диск
Прием запроса 1
Прием запроса 2
Прием запроса 3
Анализ запроса 1
Анализ запроса 2
Чтение данных 1
Чтение данных 2
Формирование ответа 1
Формирование ответа 2
Передача ответа 1
Передача ответа 2
Анализ запроса 3
...

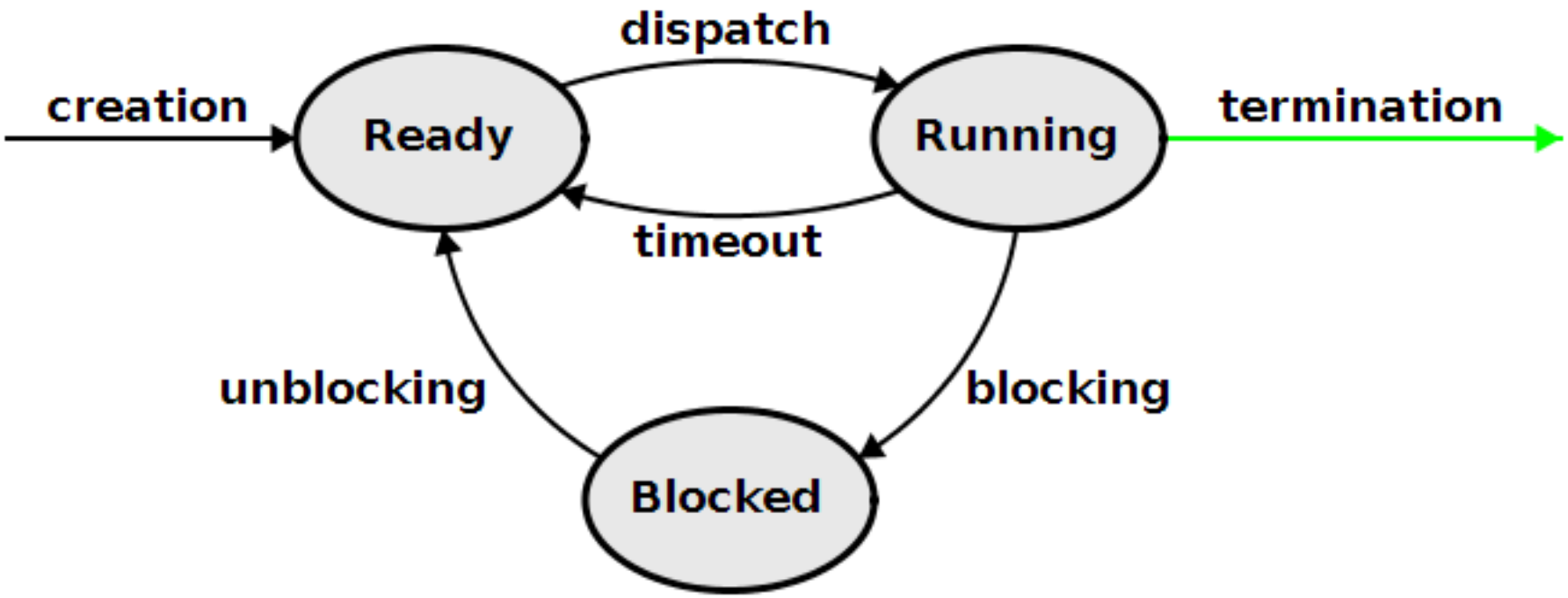
Как такое реализовать?

- Вы привыкли, что операция ввода-вывода выглядит как обычный вызов функции
- Вы знаете, что такая операция блокирующаяся
- Блокирующийся системный вызов означает, что в это время могут исполняться другие процессы
- Но при возврате вам создают иллюзию, что ваш процесс не прерывался

Как создать иллюзию непрерывности?

- Процессор хранит состояние в регистрах и памяти
- Память у каждого процесса своя
- Надо при блокировке процесса сохранить регистры, а при разблокировке – восстановить
- *Набор регистров, которые нужно сохранить и восстановить так, чтобы процесс не заметил переключения*, называется **контекстом процесса**

Butterfly diagram



Кооперативная многозадачность

- Переключаем процессы только по блокирующимся вызовам
- Можно добавить вызов для переключения без блокировки.
  В Linux это pthread_yield(3), в Solaris shed_yield(2)
  В моей книжке это называется TaskSwitch
- Главное преимущество: детерминизм
- Ошибки соревнования ловятся тестированием
  - если вы зовете блокировку или yield в критической секции, вы будете ее там звать всегда
- В старых книжках говорили, что реальное время можно реализовать только кооперативно
  - без детерминизма вы не можете давать гарантий

Недостатки кооперативной многозадачности

- Детерминизм только на однопроцессорной машине
- Что делать с плохо себя ведущими программами, которые не блокируются и не зовут yield?
- Программы, рассчитанные на однозадачные ОС, нужно переделывать (расставлять по коду yield)

Вытесняющая многозадачность

- Разрешить системе переключать задачи по своей инициативе
- Стоп. ОС – это программа. Чтобы программа могла что-то сделать, она должна получить управление

Вытесняющая многозадачность

- Разрешить системе переключать задачи по своей инициативе
- Стоп. ОС – это программа. Чтобы программа могла что-то сделать, она должна получить управление
- Прерывания
- Используя прерывания таймера, можно реализовать переключение задач по времени (timeslice, кванты времени)
- Используя прерывания от других устройств, можно передавать управление проснувшейся задаче

Преимущества вытесняющей многозадачности

- Понятно, что делать с плохо себя ведущими программами (отобрать у них процессор – не проблема)
- Поскольку разработчики вынуждены защищать критические секции, программы для вытесняющих ОС могут работать на многопроцессорных машинах без переделки
- Для вычислительных задач, таймслайс - это удобно
- Программы для однозадачных ОС можно запускать почти без переделок: поскольку такие задачи не взаимодействуют с другими, в них не может быть критических секций
  - Подождите. А как быть с взаимодействием через файлы?

Недостатки вытесняющей многозадачности

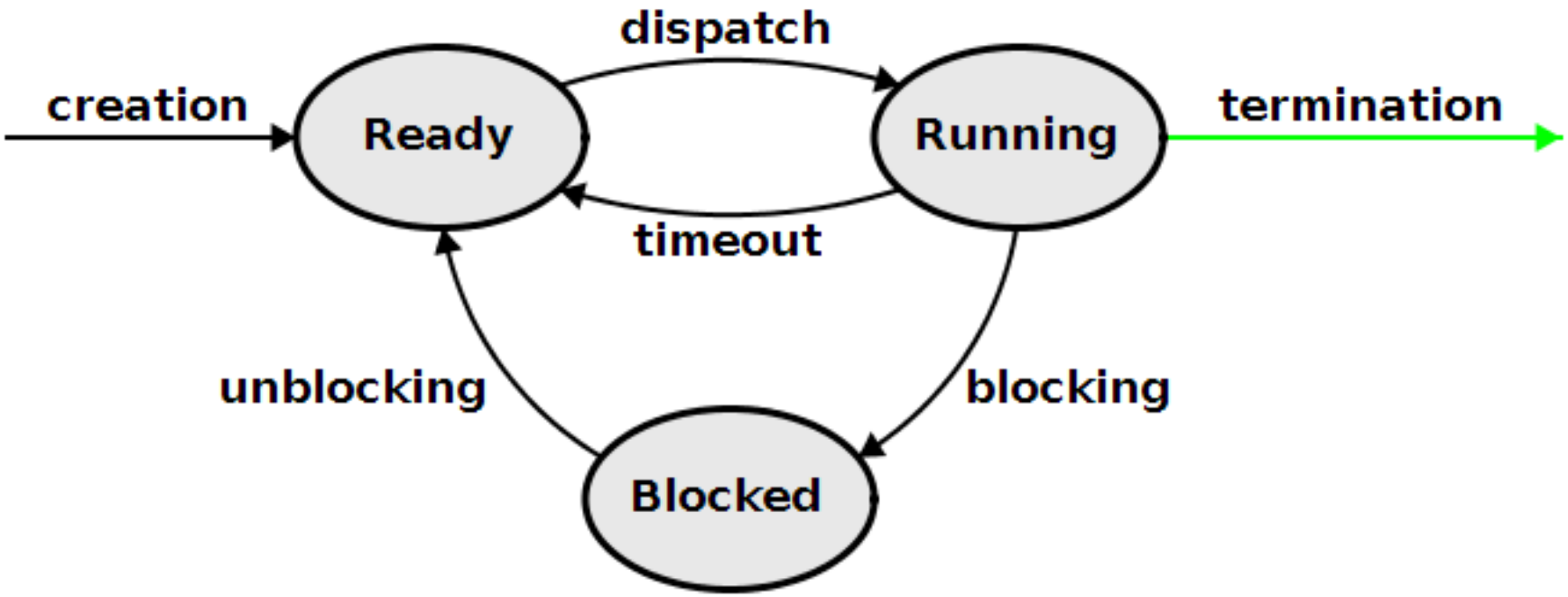
- Прерывания прилетают асинхронно
- Необходимы блокировки для защиты критических секций
- Программы для кооперативных ОС необходимо переделывать
- Индетерминизм: ошибки не ловятся тестированием

Вытесняющее планирование в Unix

- Основные понятия – процесс и нить
- Процесс – единица изоляции (виртуальная память, контекст доступа)
- Нить – единица планирования (сущность, которой создается иллюзия последовательного исполнения)
- В старых юникс-системах, у каждого процесса была ровно одна нить, поэтому процессы и нити иногда отождествляли
- В современных (~ с 1990х) юниксах, в процессе может быть несколько нитей
  (основной поток будет изучать в следующем семестре)

Планирование с приоритетами

- Если несколько задач Ready, какую выбрать?



Приоритеты (продолжение)

- Изменение порядка планирования не может влиять на функциональность программ (если в них корректно защищены критические секции)
- Но оно влияет на время исполнения и время реакции на внешние события
- Три разных цели для оптимизации:

1. Реальное время: гарантированное время реакции
2. Разделенное время: оптимизация среднего времени реакции
3. Справедливое планирование

Реальное время

- Обычно используют фиксированные приоритеты
- Расписываем требования к времени реакции для всех задач
- Задачам с меньшим временем даем более высокий приоритет
- При подсчете гарантий времени для низкоприоритетных задач нужно учитывать не только задержки в системе, но и время работы всех более приоритетных задач

Разделенное время

- Обычно используют динамические приоритеты
- Задачам, которые реагируют на внешние события (чего-то ждут) приоритет повышают
- Задачам, которые не реагируют, но занимают ЦПУ приоритет понижают
- Классический планировщик Unix:
  - Когда процесс снимают по кванту времени, ему начисляют штраф
  - Когда процесс сам освобождает процессор (блокируется), штраф снимают

Справедливое планирование

- Рассмотрим компьютер коллективного пользования
  - Разделяемый хостинг
  - Облачные инфраструктуры
- Пользователи платят за процессорное время
- Если им не выдать, будут спрашивать «за что платим?»
- Процессорное время нужно для обработки событий (например, запросов к веб-сайту)
- Поэтому, если вам все оплаченное время выдать одним куском в полночь по гринвичу, вас это вряд ли устроит

Справедливые планировщики

- Вводится понятие «периода справедливости», например, 1 секунда
- Процессы делятся на группы (в Linux – cgroup)
- Каждой группе выделяем долю времени в каждом периоде справедливости
- Планируем только процессы тех групп, которые еще не израсходовали свою долю в текущем периоде
- Если таких процессов нет, можно планировать любые готовые процессы.

Чем плох справедливый планировщик?

- У классического планировщика разделенного времени, решение об изменении приоритета и планировании принимается за константное время
- У справедливого планировщика, нужен поиск по группам время планирования зависит от количества процессов и групп