

Основы операционных систем

Лекции

2024–2025

2024-09-02

Обзор. Введение	3
Структура вычислительной системы	3
Определение операционной системы. Основные точки зрения	3
Эволюция вычислительных систем	3
1-ый период (1945 – 1955)	3
2-ой период (1955 – начало 60-ых)	4
3-период (начало 60-ых – 1980)	4

2024-09-09

4-ый период (1980 – 2005)	5
5-ый период (2005 – ??)	5
Основные функции ОС	5
Архитектурные особенности построения ОС	6
Внутреннее строение ОС	6
Понятие процесса. Операции над процессами	7
Процесс	7
Состояние процесса	7

2024-09-16

Process Control Block и контекст процесса	7
Одноразовые операции	8
Создание процесса	8
Завершение процесса	8
Запуск процесса	9
Приостановка процесса	9
Блокирование процесса	9
Разблокирование процесса	9
Пример цепочки операций	9
Кооперация процессов	9
Категории средств взаимодействия	9
Как устанавливается связь	9
Информационная валентность процессов и средств связи	10

2024-09-23

Особенности канальных средств связи	10
Буферизация	10
Модель передачи данных	10
Примеры средств связи	11

Pipe	11
FIFO (именованный pipe)	11
Надежность средств связи	11
Как завершить связь?	11
Нити исполнения (threads)	11
Алгоритмы синхронизации	13
Условие Бернштейна	13
Как чинить?	14

2024-09-02

Обзор. Введение

Карпов Владимир Ефимович

carpsov@mail.ru

Оценка:

$$O_{\text{теор}} \frac{O_{\text{кр}} + O_{\text{экз}}}{2}$$
$$O_{\text{итог}} \frac{2 * O_{\text{практ}} * O_{\text{теор}}}{O_{\text{практ}} + O_{\text{теор}}}$$

0 за $O_{\text{теор}}$ или $O_{\text{практ}}$ обнулит $O_{\text{итог}}$

Структура вычислительной системы

- Пользователь (человек или устройство)
- Программное обеспечение
 - Прикладные программы
 - Системные программы
 - Прочие системные программы
 - Операционная система
- Техническое обеспечение

Границы между прикладным и системными программами условны

Определение операционной системы. Основные точки зрения

- Распорядитель ресурсов
- Защитник (разграничитель) пользователей и программ
- Виртуальная машина (создание абстракций для работы с файлами, памятью и тд)
- «Кот в мешке»
- Постоянно функционирующее ядро (программа, которая поддерживает работу других программ)
- ...

Эволюция вычислительных систем

Факторы отбора эволюции выч. систем:

- Удобство пользования
- Стоимость
- Производительность

Развитие hard- и software нужно рассматривать совокупно

Периоды развития:

1-ый период (1945 – 1955)

- Ламповые машины
 - Огромные
 - Часто перегорали

- Нет разделения персонала
- Ввод программы коммутацией или перфокартами
- Одновременное выполнение только одной операции (те только либо программирование, либо расчеты, либо счет перфокарты и тд)
- Появление прообразов первых компиляторов
- Нет операционных систем

Фон Нейман имеет минимальное отношение к принципам Фон Неймана

ENIAC работал в 10-ой ссч

2-ой период (1955 – начало 60-ых)

- Полупроводниковые компьютеры
- Разделение персонала
- Бурное развитие алгоритмических языков
- Ввод задания колодой перфокарт
- Вывод результатов на печать
- Пакеты заданий и системы пакетной обработки (прообраз операционных систем)

Начало использования ЭВМ в коммерческих и исследовательских целях

3-период (начало 60-ых – 1980)

- Микросхемы, машины на интегральных схемах
 - Машины меньше
 - Сигнал идет быстрее, можно повысить тактовую частоту
 - Считыватели и принтеры становятся bottleneck
- Использование спулинга (отдельные процессоры для ввода, вывода и счета)
- Планирование заданий (из-за создания магнитных дисков)
- Мультипрограммные пакетные системы
- Системы разделения времени (time-sharing)

Изменения из-за мультипрограммирования:

Software	Hardware
<ul style="list-style-type: none"> • Планирование заданий • Управление памятью • Сохранение контекста • Планирование использования процессора • Системные вызовы • Средства коммуникации между программами • Средства синхронизации 	<ul style="list-style-type: none"> • Защита памяти • Сохранение контекста • Механизм прерываний • Привилегированные команды (в тч команды ввода/ вывода)

2024-09-09

Опр. Квант времени — время, пока программа работает подряд (без передачи управления другим программам)

Раньше клавиатура и дисплей стали независимы, потом превратились в терминалы, которые выводили данные во время выполнения программы

Появилась возможность **отладки**

Появляются **файловые системы** (много пользователей могут работать на одном устройстве хранения данных)

Программа обычно считается частями: всё программу хранить в оперативной памяти не обязательно

Появляется концепция **виртуальной памяти**: абстракция, иллюзия большой оперативной памяти

Появилась идея **обратной совместимости, полной совместимости, линеек устройств** (от слабых компьютеров до мэйнфреймов)

Популярные линейки:

- IBM
- PDP

Обратная совместимость имеет преимущества, но и заставляет «тащить» за собой недостатки

В опр момент IBM решили, что баги в системе править не будут, так как возникают новые баги

4-ый период (1980 – 2005)

- 1980 год — развитие больших интегральных схем: весь процессор мог быть на одном кристалле
- Первые персональные ЭВМ
- Дружественное программное обеспечение: программы пишутся для удобства пользователей
- Резкая деградация ОС: пропадает мультипроцессорность, защита памяти и т.д.
- Из-за роста мощности (в 90-е) деградация ОС прекращается
- Переосмысление роли сетей: из оборонки в пользовательские
- Сетевые и распределенные ОС

Сетевая ОС — пользователь явно использует возможности сети

Распределенная ОС — пользователь неявно использует возможности сети, используется абстракция

Период широкого использования ЭВМ в быту, в образовании, на производстве

5-ый период (2005 – ??)

- Появление многоядерных процессоров
- Мобильные компьютеры
- Высокопроизводительные вычислительные системы
- Облачные технологии
- Виртуализация выполнения программ: выполнение программы на любом из компьютеров распределительной сети

Период глобальной компьютеризации

—— Основные функции ОС ——

- Планирование заданий и использование процессора
- Обеспечение программ средствами коммуникации и синхронизации (межпроцессорные коммуникации)

- Управление памятью
- Управление файловой системой
- Управление вводом-выводом
- Обеспечение безопасности

Дальше в курсе будем изучать, как эти функции выполняются по отдельности и совместно

Архитектурные особенности построения ОС

Внутреннее строение ОС

- Монолитное ядро:
 - Каждая процедура может вызывать каждую
 - Все процедуры работают в привилегированном режиме
 - Ядро совпадает со всей операционной системой (вся ОС всегда сидит в оперативной памяти)
 - Точки входа в ядро — системные вызовы

+	-
- Быстродействие	- Нужно много памяти - Невозможность модификации без полной перекомпиляции

- Многоуровневая (Layered) система:
 - Процедура уровня K может вызывать только процедуры уровня $K - 1$
 - [Почти] все уровни работают в привилегированном режиме
 - Ядро [почти] совпадает со всей операционной системой
 - Точка входа — верхний уровень

+	-
- Легкая отладка (при удачном проектировании)	- Медленно - Нужно много памяти - Невозможность модификации без полной перекомпиляции

- Микроядерная (microkernel) архитектура:
 - Функции микроядра:
 - взаимодействие между программами
 - планирование использования процессора
 - ...
 - Микроядро — лишь малая часть ОС
 - Остальное — отдельные программы-"менеджеры", раб в пользовательском режиме
 - Всё общение через микроядро

+	-
- Только ядро — «особенное» - Легче отлаживать и заменять компоненты	- Ядро перезагружено — bottleneck - Всё очень-очень медленно работает

- Виртуальные машины
 - У каждого пользователя своя копия hardware
 - Пример:
 - Реальное hardware
 - Реальная ОС

- Виртуальное hardware - Linux - Пользователь 1
- Виртуальное hardware - Windows 11 - Пользователь 1
- Виртуальное hardware - MS-DOS - Пользователь 1

+	-
- Удобно	- Медленно из-за многоуровневости

- Экзоядерная (новая микроядерная) архитектура:
 - Функции экзоядра:
 - взаимодействие между программами
 - выделение и высвобождение физических ресурсов
 - контроль прав доступа
 - Весь остальной функционал выкидывается в библиотеки

Подходы **не** используются в чистом виде

Понятие процесса. Операции над процессами

Процесс

Термины «программа» и «задание» были придуманы для статических объектов

Для динамических объектов будем использовать «процесс»

Процесс характеризует совокупность:

- набора исполняющихся команд
- ассоциированных с ним ресурсов
- текущего момента его выполнения (контекст)

Процесс \neq программа, которая выполняется тк:

- одна программа может использовать несколько процессов
- один процесс может использовать несколько программ
- процесс может исполнять код, которого не было в программе

Состояние процесса

Процесс сам состояния не меняет, его переводит ОС, совершая «операцию»

2024-09-16

Набор (пар) операций:

- одnorазовые:
 - создание – завершение
- многоразовые:
 - запуск – приостановка
 - блокирование – разблокирование
 - изменение приоритета процесса

PROCESS CONTROL BLOCK и контекст процесса

Process Control Block — структура (или набор структур) с информацией о процессе; хранит:

- Регистровый контекст:
 - Программный счетчик
 - Содержимое регистров
- Системный контекст:

- Состояние процесса
- Данные для планирования использования процессора и управления памятью
- Учетная информация

PCB хранится в адресном пространстве ОС

Код и данные в адресном пространстве — пользовательский контекст

Контекст процесса — совокупность всех трех контекстов

Одноразовые операции

Медленные тк:

- Выполняют много действий
- Выделяют/ освобождают ресурсы
- Меняют число процессов

Первые несколько процессов создаются «хитро»

Всякий новый процесс рождается от другого процесса → процессы образуют генеалогическое дерево (или лес)

Создание процесса

1. Присваивание PID
2. Создание PCB с состоянием «рождение»
3. Выделение ресурсов.

Подходы:

1. из ресурсов родителя
 2. из ресурсов ОС (более частый вариант)
4. Занесение кода в адресное пространство и установка программного счетчика.

Подходы:

1. Клонирование родителя (в UNIX)
Есть системный вызов, который создает клона
Есть системный вызов, который полностью заменяет пользовательский контекст
 2. Из файла (в Windows)
5. Окончательное заполнение PCB
 6. Изменение состояния на «готовность»

Завершение процесса

Состояние «закончил исполнение» нужно, чтобы возможно было узнать причину завершения

1. Изменение состояния на «закончил исполнение»
2. Освобождение ресурсов
3. Очистка элементов PCB
4. Сохранение в PCB информации о причинах завершения

Процесс выкидывается из системы, когда родитель

- умирает

- интересуется причинами завершения

Если родитель умер раньше ребенка, PID=1 усыновляет ребенка

Zombie-процесс — процесс в состоянии «закончил исполнение»

· Запуск процесса ·

- Изменение состояния на «исполнение»
- Обеспечение наличия в оперативной памяти необходимой информации
- Восстановление значения регистров
- Передача управления по адресу программного счетчика

· Приостановка процесса ·

- Автоматическое сохранение программного счетчика и части регистров (работа hardware)
- Передача управления по специальному адресу (работа hardware)
- Сохранение динамической части
- Обработка прерывания
- Перевод процесса в состояние «Готовность»

· Блокирование процесса ·

- Сохранение контекста процесса в РСВ
- Обработка системного вызова
- Перевод процесса в состояние «ожидание»

· Разблокирование процесса ·

- Уточнение, какое событие произошло
- Проверка наличия процесса, ожидавшего события
- Перевод ожидающего процесса в состояние «готовность»
- Обработка произошедшего события

· Пример цепочки операций ·

———— Кооперация процессов ————

Кооперативные (взаимодействующие) процессы — процессы, которые влияют на поведения друг друга путем обмена информацией

Основные причины кооперации:

- Повышение скорости решения задач (для многоядерной системы)
- Совместное использование данных
- Модульная конструкция какой-то системы
- Для удобства работы пользователя

Взаимодействие между процессами происходит через ОС

—— Категории средств взаимодействия ——

- Сигнальные: передача бита
- Канальные: логический канал
- Разделяемая память: общее адресное пространство

—— Как устанавливается связь ——

- Нужна ли инициация?

- Обычно нужна для канальной и разделяемой памяти и не нужна для сигнальной
- Способы адресации:
 - Прямая
 - Симметричная: и отправитель, и получатель указывают ID друг друга
 - Асимметричная: только отправитель указывает ID получателя
 - Косвенная: есть вспомогательный объект для передачи

—— Информационная валентность процессов и средств связи ——

- Сколько процессов может одновременно ассоциировать с конкретным видом связи? — «It depends».
- Сколько идентичных средств связи может быть задействовано между двумя процессами? — «It depends».
- Направленность связи:
 - Симплексная связь: односторонняя
 - Полудуплексная связь: как в рации
 - Дуплексная связь: двусторонняя

2024-09-23

—— Особенности канальных средств связи ——

· Буферизация ·

Обладает ли канал внутренней памятью

Случаи:

- Буфера нет
Процесс-передатчик блокируется, пока процесс-получатель не считывает данные
- Буфер неограниченной емкости (физически не реализуем)
Процесс-передатчик никогда не ждет
- Буфер конечной емкости
Самый частый случай

· Модель передачи данных ·

- Потокковая модель
 - Операции приема/ передачи не интересуются содержанием данных и их происхождением
 - Данные не структурируются
 - Нет разделителей между записываемыми блоками
 - Можно считывать любое количество байт
- Модель сообщений
 - На данные накладывается некоторая структура
 - Отдельные сообщения явно разделены
 - Иногда сообщение хранит дополнительные данные, например, имя процесса-отправителя

- Считывать можно только сообщение целиком

—— Примеры средств связи ——

PIPE

- Потокковая модель
- Косвенная адресация
- Читать и писать может любое число процессов
- Однонаправленный

Через pipe могут общаться только процессы-родственники, так как вход и выход в pipe не видны остальной ОС.

FIFO (именованный PIPE)

Вход и выход именуются

Теперь общаться могут любые процессы, а не только родственники

—— Надежность средств связи ——

Система считается надежной:

- Нет потери информации
- Нет повреждения информации
- Нет нарушения порядка
- Не появляется лишняя информация

—— Как завершить связь? ——

Специальное действие для завершения нужно, если было нужно специальное средство для завершения

Если один из процессов больше не будет использовать средство связи, то система оповещает другой

—— Нити исполнения (threads) ——

```
Ввести массив A
Ожидания ввода A
Ввести массив B
Ожидания ввода B
Ввести массив C
Ожидания ввода C
A = A + B
C = A + C
Вывести массив C
Ожидание вывода C
```

Процессор много простаивает — хочется распараллелить: нужен второй процесс:

Процесс 1

Создание процесса 2

Переключение контекста

Процесс 2

Доступ к общей памяти

Ожидание ввода А и В

Переключение контекста

Доступ к общей памяти

TODO

Нужны дополнительные действия:

- Породить процесс 2
- Получить разделяемую память
- Нужны переключения контекста

На одноядерной системе это неэффективно из-за доп расходов

Аналогия с железной дорогой:

- Поезд — процессор с регистрами и данными в стеке
- Стрелки — условные переходы
- Склады — данные вне трека/ операции ввода-вывода

Если два поезда могут ехать одновременно, то получается мультипроцессорная система

Thread — каждый из поездов

В процессе могут быть несколько thread-ов:

- Общие:
 - Системный контекст (всего процесса)
 - Код
 - Данные вне стека
- Разные:
 - Регистровый контекст
 - Стек
 - Системный контекст thread-a

Для создания нового thread-a используется системный вызов

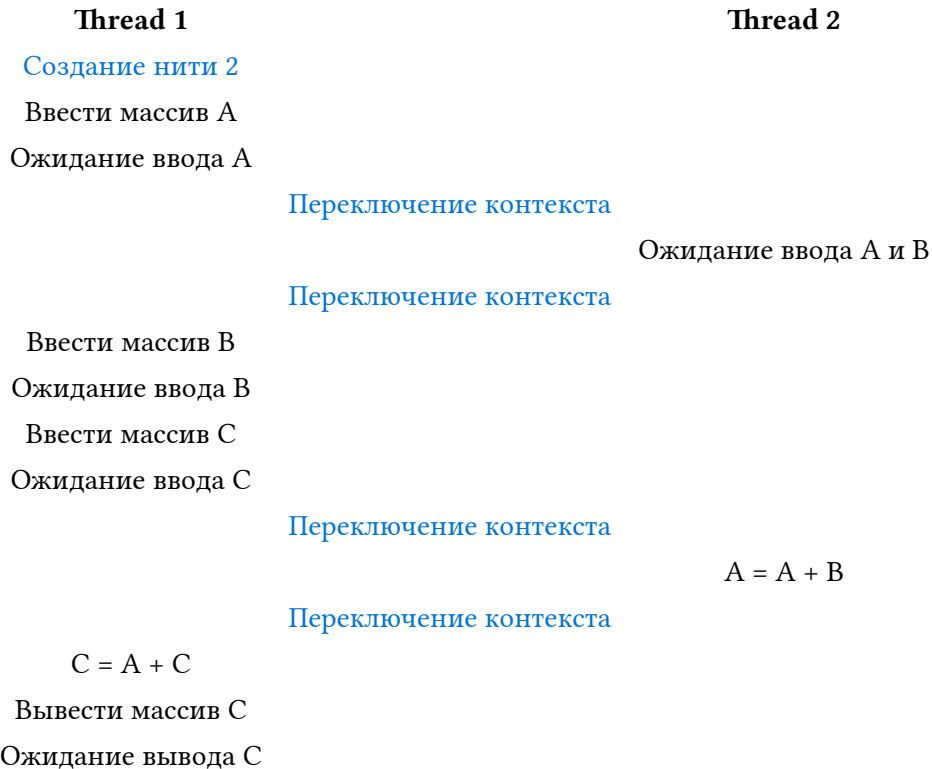
Между thread-ами есть отношение родитель-ребенок

У thread-ов тоже есть состояния, как и у процессов

Master thread (главная нить) — нить создаваемая при создании процесса

Thread-ы хороши, так как создавать их «дешевле», чем новые процессы

Thread-ы могут создавать либо на уровне библиотек, либо на уровне ядра



———— Алгоритмы синхронизации ————

Активность — последовательное выполнение ряда действий, направленных на достижение определенной цели

Операции внутри активности считаем атомарными (неделимыми): между операциями «отвлекаться» можно, во время операции — нельзя

Активность P из операций a b c

Активность Q из операций d e f

Последовательное выполнения PQ: a b c d e f

Псевдопараллельное выполнение (режим разделения времени): порядок операций от каждой активности фиксирован, но они могут чередоваться — интерливинг. Например, **a d e b c f**

Если активности зависимы, то могут быть проблемы

Недетерминированный набор — при одинаковых начальных результатах возможны разные результаты

Детерминированный набор — при одинаковых начальных данных результат всегда один

———— Условие Бернштейна ————

Достаточные условия детерминированности набор

Входные данные активности (W): объединение множеств входных данных всех активностей

Выходные данные активности (R): объединение множеств выходных данных всех активностей

Условия (для двух активностей P и Q):

- $W(P) \cap W(Q) = \emptyset$
- $W(P) \cap R(Q) = \emptyset$
- $R(P) \cap W(Q) = \emptyset$

—— Как чинить? ——

Нужно запретить «плохие» чередования

В не детерминированных наборах всегда встречается **race condition** (состояние гонки).

Mutual exclusion (взаимоисключение): если процесс захватил ресурс, то больше никто другой его не использует. Используется, когда не важно, кто первый захватил ресурс

Опр. Критические секции — участки, которые приводят к появлению race condition

Нужно сделать, чтобы критические секции выполнялись, как атомарные операции