- Программное обеспечение – совокупность программ и описание их u/ (Прикладное & Системное)
- Мат. Обеспечение – совокупность методов и их описаний u/ 4/ решения задачи.
- СПО – ОС - u/4/ обеспечения функционирования вычислительной системы. Ф-ции: Обеспечить эффективное прохождение заданий usera через систему (с точки зрения usera – среднее время ожидания и производительность).Повышения эффективности u/ dev.
- Мультипроцессорные системы – совокупность процессоров в одном конструктиве, имеющих сильную связь по передаче данных (Общая память, единая ОС)
- Многомашинные комплексы – sys из нескольких машин объединенные в единое целое w/u/ каналов связи
- Распределенная sys – совокупность выч. узлов, связанных между собой каналами связи, с точки зрения usera представляют собой единое целое.
- Непосредственный доступ – работа usera в системе с полным контролем работы машины – ресурсы в полном распоряжении usera.
- Косвенный доступ – подразумевает спец. Программу через которую ведется работа с машиной (Прообраз ОС, пакетный режим)
- Коллективный доступ – одновременная работа нескольких userov на машине (Логика прерываний)
- Режимы реализации многопрограммной работы –1.Классическое мультипрограммирование – обработка задач w/u/ различного оборудования 2. Параллельная обработка – квантование 3. Разделение времени – объединение 1&2 4.Режим реального времени – время реакции системы соответствует ранее заданному 5. Мультипрограммирование со свопингом.
- Истинно || выполнение – 1.Чистое совмещение (различные задачи занимают различные ус-ва) или (Одна задача на нескольких dev – д/б/ построена на || алгоритме) 2. u/ конвеер – задача раэбивается на шаги.
- Ускорение при || обработке – Тпос/Тпар – Только в случае однородных процессоров
- Гетерогенная система (неоднородная) – каждый узел имеет свою производительность – (Hard+SoftOS)
- Закон Амдала(?) – Получить ускорение в n раз на n процессорах практически невозможно т.к. любая программа содержит по крайней мере одну последовательную операцию.
- Классификация Фишла(?):SISD,SIMD,MISD,MIMD (Single(одиночная),Multi(множественная),Instruction(Команда),Data(Данные)) (Нарисовать картинку с ОЗУ из которой поступают данные и команды и один или кучу процессорных эллементов) КОМПУТЕР=ОПЕРАЦИИ+ОБМЕН+(ДАННЫЕ) (…) – т.к. || вычисление переходит в || обработку.
- || проганье – Мелко, Средне, Крупно – зернистое. Зерно – определяется Твычисления/Тобмена.
- Организация памяти SRSW,SRMW,MRSW,MRMW (Read,Write,Single,Multi)
- PARA-PC Velsh – Рисунок – много ОЗУ много ПЭ – все подключены к облаку – Коммуникационная среда – (Статическая и динамическая коммутация(ПЭ-ПЭ или ПЭ ОЗУ)) => Можно моделировать (Общую Память) (Общая Шина) (КЭШ-шинно ориентированные – Общая шина + Каждый ПЭ общается с шиной via КЭШ (содержит программный код и промежуточные данные) (Система с виртуальной памятью – предидущее без ОП))
- Модернизация ОС – 1.Своя ОС 2. Дополнить исходники своими ф-циями 3. Добавление нового слоя в ОС – обеспечение виртуального выполнения наших ф-ций в рамках ф-ций ОС
- Классификация ОС – одно-много-программные, одно-много-процессорные, распределенные, виртуальные.
- Виртуальные ОС – разрешает нескольким useram работать на одном и томже dev на разных ОС.
- Заявка на входе ОС – задание – проходят через три уровня сверху (верхний,средний,нижний). Прошедшие верхний уровень претендуют на захват ресурсов в системе. Средний уровень обрабатывает приоритеты переводит задания в ранг задач. Планировщик среднего уровня активизирует ОС при наличии свободного ресурса.
- Задача – внутренняя единица работы системы, которой система выделяет ресурс. (д/ иметь Task Control Block) В каждой задаче имеется программа подчиненная задаче (обработка данных задачи).
- Планировщик нижнего уровня – определение очередности задач по выделению времени процессора.
- Семиуровневая модель – схема системы планирования с учетом (Маштабируемость, Разделяемость, Параллельность) 1. Предварительное (входное) планирование исходного потока заявок (задача фильтрации) 2. Структурный анализ взаимосвязи входного потока заявок по ресурсам с определением общих ресурсов (Анализ) 3. Структурный анализ заявок и определение возможности распараллеливания (Задача распараллеливания) 4. Адаптация распределения работ соответственно особенностям ВС (Задача адаптирования) 5. Составление плана – расписания выполнения взаимосвязанных процедур. Оптимизация плана по времени решения и кол-ву ресурсов (Задача Оптимизации) 6. Планирование потока задач претендующих на захват времени процессора на каждый прцессор – задача распределения. 7. Выделение процессорного времени, активизация задач. Перераспределение работ в ВС, при отказе оборудования (задача распределения – перераспределения).
- Динамическое планирование – задача планирования решается на том же оборудовании, что и выполняется план, который она составляет. Быстрое (чтобы не грузить оборудование) и неоптимальное решение
- Статическое планирование – план составляется на другой машине &&/|| в другое время. Оптимальный(оптимизированный) план за обозримое (физически нормальное время) Критерий оптимальности – время решения задачи и минимизация u/ресурсов. (NP-полная – времення сложность - експонента)

1. Найти план решения рас||ной задачи представленной в виде ориентированного ациклического графа, время решения которого было бы равно Ткритическому с минимальным числом процов. 2. Найти мин. Время решения задачи при заданном кол-ве процов.

- NP-полная – времення сложность - експонента
- Временная сложность – время решения задачи в зависимости от замерности.
- Предел Брементана(?) – if вычислительная сложность 2^63 => нужна машина с земной шар.

- Планирование в RealTime – в sys планировщика добавлены временные ограничения заданные зарание, необходимо составить план распределения задач по ресурсам, чтобы все задачи выполнялись в системе в заданное время или ранее того. (На Худой Конец – мин. суммарное отклонение)
- Строго неоднородная система – заявка не может бвть выполнена на данном ресурсе.
- Полунеоднородная sys – заявки одинаковы а ресурсы нет visa verse.
- Балансовое планирование – в распределенных системах и сетях. Задачи: 1. Равномерная загрузка узлов системы с попыткой улучшения кол-ва задач || загрузки оборудования. 2. Определение адреса решаемой задачи. 3.Перераспределение заявок. 4. Повышение надежности системы – где находится контрольная точка и куда сбросить ин-фу.
- Задача управления ресурсами – Возможность планирования выполнением работ, идентификация, организация обеспечения ресурсов с уточнением местонахождения, определение доступности ресурсов и их стоимости, согласование распределения ресурсов и оптимизация их u/.
- Процесс планирования распред. ресурсов и задач ::= отображение Макрофакторов – 1.Системы (архитектура, структура, хак-ка) 2. Поток заданий (параметры заданий, требований, структуры) 3. Времени. – Выполнение заданий в пространственно временных координатах.
- Вычислительная система – граф системы от 4-х параметров = {мн-ва узлов системы, описание(вес) узла, связи между узлами, вес связи (время передачи ин-фы)}
- Поток заданий – граф данных = {мн-во работ системы, наличие связей, вес работы, вес связи}
- Требование к задачам планирования работ по ресурсам – Нахождение отображения мн-ва ресурсов в мн-во работ исходя из условий оптимизации. (scheduling – без указания места расположения, mapping – с указанием места расположения)
- Общие требования к системе планирования 1. Принцип равного доступа – нет задач в режиме бесконечного откладывания. 2. Max. производительность системы – заявок в ед. времени. 3. Max. кол-во коллективных userov с приемлемым временем ответа. 4. Предсказуемость – каждое задание д/б/ выполнено за примерно одинаковое время с теми-же аппаратными затратами несмотря на загрузку системы. 5. Мин. затрат ресурсов на решение задачи планирования. 6. U/ балансовое планирование – загрузка ресурсов системы. 7. Баланс между временем ответа и применением. 8. U/ динамических приоритетов – во избежании бесконечного откладывания. 9. U/ Вынужденных приоритетов. 10. Снижение эффективности под большими загрузками (нет отказам!!!) Многие требования конфликтны => задача планирования – сложная.
- Классификация || систем по обмену данными 1. SMP (одинаковые процы) – Связь = ОП || PointToPoint. (UniformMemAccess) (NotUMA – Память разделена в пространстве и времени) По доступу к памяти – сосредоточенная && рассредоточенная
- Мультипрограммироваие – в машине несколько задач каждая из которых занимает некоторое ус-во. – Время решения задач уменьшается, но решение отдельной задачи может увеличится. (Сравнивая с пакетным режимом – загрузка процов увеличена, если задача захватила проц. то она его не отдает)
- Квантование - || проганье – выход из ситуации монопольного владения ресурсами. Уменьшение времени решения задачи и реакции системы. (-)=> при переключении задач по кванту надо сохранять ее состояние => появляется задача выбора кванта времени.

ДИСЦИПЛИНЫ ОБСЛУЖИВАНИЯ ЗАЯВОК

- Дисциплина обслуживания заявок – правило по которому решается задача стоящая перед системой – обработка очереди заявок к каждому ресурсу. ЕСЛИ в системе нет одинаковых устройств то очереди обрабатываются по линейной схеме => планирование только во времени ИНАЧЕ планирование во времени и в пространстве.
- Транзакция – короткое непрерываемое задание u/ в многопотоковой системе.
- Планировщик работает от ресурса – освободился ресурс => начинаем опрос очереди. ЕСЛИ очередь одна ТО два планировщика не могут ее обработать.
- Бесприоритетное обслуживание – заявка поступает в систему без установленного приоритете – RANDOM,FIFO,LIFO,RR,FBN, Корбато.
- Алгоритм Корбато – отбор по длине программного кода, длинные заявки сразу попадают вверх очереди. Время обработки расчитывается от длинны заявки.
- Приоритетное обслуживание – заявка на вход системы с заданным приопитетом.
- Относительный приоритет – не может прервать задачу на ресурсе, даже если она имеет низкий приоритет.
- Абсолютный приоритет – прерывает задачу на ресурсе если та имеет более низкий приоритет.
- Динамический приоритет – ЕСЛИ возникает опасность бесконечного откладывания. ЕСЛИ по мнению sys, заявка слишком долго занимает ресурс, ТО ее приоритет понижается. ЕСЛИ заявка очень долго ожидает ресурс ТО ее приоритет повышается.
- Приоритеты можно смешивать получая различные х-ни (QoS)
- В зависимости от вида неоднородности системы отношение заявка-ресурс m/b/ взвешенным (степень претендования заявки на ресурс) и невзвешенным (только возможность размещения). Для строго неоднородной системы – надо найти такое распределения заявок по ресурсам, чтобы кол-во распределенных заявок было max. Для взвешенного отношения – кол-во распределенных заявок было max. при max сумме весов этих распределений. Строго неоднородная система: Kv = for(j=0;j<N;j++) *=Xj; где Xj={0;1} – выполнение или невыполнение обязательного требования. Неоднородная система: for(j=0;j<N;j++) Kv*=Xj*(for (I=1;I<M;I++) +=Yi); Yi – степень выполнения оптимизирующих требований. Ki – вес требования

МНОГОУРОВНЕВЫЙ МЕТОД РЕШЕНИЯ СЛОЖНЫХ ЗАДАЧ

- Декомпозиция – разбиение сложной задачи на более мелкие. (Каждый класс задач на своем уровне связь вниз – выполнение ф-ции, вверх возврат результата) 1.Визуальное программирование 2. ЯВУ 3. Asm 4. Уровень ОС 5. Машинный язык 6. Машинные команды. 7. Dev.

- 6 уровней языков Запрос к файлу (операция, имя, логическая запись) – 1. Символьный (опр. уникального имени файла) 2. Базовый (определение хар-к доступа к файлу) 3.Права доступа 4. Логический (Координата записи) 5. Физический (Номер физического блока)
- Сеть: Физический (соед с передачей) Канальный (Доступность среды, Соединение машин) Сетевой (Единая транспортная система) Транспортный (Надежность передачи данных) Сеансовый (диалог в сети) Представления (форма передачи данных) Прикладной (Набор средств для доступа к сетевым ресурсам)

СОСТОЯНИЯ ЗАДАЧ (процессов)

- Основная ф-ция ОС – слежение за прохождением задачи от входа до выхода
- Задача после второго уровня планирования обзаводится PCB. Для системных задач эти блоки создаются во время загрузки. (init kernel)
- Разновидности программ: 1. Простейшая структура (имеют в своем теле все что нужно для выполнения) 2. Оверлей (заданные перекрытия два модуля на одном оверлейном уровне не могут быть выполнены Инфа передается via корневой модуль) 3. Динамически посл.(Память выделяется по мере надобности – по цепочке. Уровни не выделяются. Корневой модуль всегда в памяти) 4. Динамически ||.
- Цепочки связи между задачами – имя модулей кто/кого вызывает
- Поля ожидания – попытка вызова занятого или незагруженного модуля.
- Системе – пох. Системная задача или нет – она смотрит только на проиритет.
- 4 – состояния процесса – P1 – обработка системных и проблемных задач P2 – обработка прерываний P3 – дешифрация прерываний P4 – Exeption Переход на P3 при появлении сигнала прерывания, и выполнения дешифрации (определяется источник прерывания) – Реализовано на аппаратном уровне.
- 5 – состояний задачи – А – активность (выделенно время процессора) Ож – (ожидание завершения операции) Гот- (активизировани стоит в ичереди) Подг – (не задана-зафиксирована работа в системе) З (отдала все ресурсы но блок управления еще сохранился) (Из А – в любое, Из Ож -> Гот, Из Подг -> Гот, Из Гот –> А)

ДИНАМИЧЕСКОЕ ПЛАНИРОВАНИЕ

- Активизируется при наличии ресурса -> Планировщик загружается в ту область, куда будет грузится задача. По структуре -> Невосстанавливаемые(для последующего вызова необходимо перезаписывать снова в память), самовосстанавливаемые , реинтерабельные(чистые – процедуру можно загружать одновременно несколько раз).
- Можно реализовать планировщик, активизирующийся на группу заявок => управление от ОС.
- Пространственный планировщик – решает задачу максимального паросочетания.(Алгоритм Карзанова и Карпа-Хопкрафта – Поиск максимального потока или увеличивающегося чередующегося пути)
- Max. парасочетание – max кол-во ребер, не имеющих инцендентных вершин || в матрице инцендентности найти max кол-во единичек в которых не совпадают координаты.
- Теорема Герта: max парасочетания получаются, если от свободной вершины нельзя найти увеличивающийся путь.Идея: для двудольного графа, случайно, выделяются ребра, которые не имеют инцендентных вершин. Получаем базовое решение…..
- Временная сложность – зависимость времени решения задачи от размерности задачи. (Берем наихудший случай)
- Теорема2: Если в двудольном графе имеется несколько вершин со степенью, и ребра инцендентные этим вершинам образуют веер, то любая из этих вершин со своим напарником может быть, безусловно, взята в решение.
- Следствие1: Все остальные вершины входящие в веер должны быть удалены а граф редуцирован.
- Следствие2: Мощность, для поиска паросочетания д/б уменьшена на кол-во вершин входящих в веер.
- ЕСЛИ процент заполнения матрицы > 50 => алгоритм ОБЯЗАТЕЛЬНЫХ НАЗНАЧЕНИЙ неэффективен (даже 30%) => u/ обычный алгоритм или u/ алгоритм Симоненка
- Algo.Simonenko – При анализе матрицы связности m/ выделить не только обязательные но и конфликтные назначения, которые точно не нужно u/ теорию перманентов – Т. Фробинце-Кинга.
- Теорема. ЕСЛИ в лвумерной матрице перманент матрицы проще эл-тов, стоящих по диагонали, то совершенное паросочетание m/b получено перестановкой строк и столбцов (изоморфное преобразование графа – влияет на его сущность)
- ЕСЛИ перманент = 0 ТОГДА мощность парасочетания = (размерность матрицы) – кол-во нулей на главной диагонали.
- ЕСЛИ в гафе m/ выделить мост (ребро соединяющее две части графа) ТО мост – конфликтен.
- Один Мост – признак явной цикличности Много Мостов – неявная цикличность
- Алгоритм определения конфликтных назначений – ЕСЛИ в матрице связности можно выделить основную подматрицу (из нулей) x*y, x+y=N и расположить ее в верхнем правом углу, тогда все единицы входящие в матрицу симметричную ей являются конфликтными и должны быть удалены из рассмотрения -> граф редуцируется => граф распадается на две части и поиск парасочетаний может вестись отдельно. (ЕСЛИ несколько основных подматриц ТО граф распадается на несколько частей)
- ТЕОРЕМА: ЕСЛИ в матрице связности m/ выделить несколько основных подматриц x+y>N ТО мощность парасочетания = 2N-(x+y)=N-(x+y-N)
- ЕСЛИ в матрице есть нижняя треугольная матрица нулей ТО система имеет решение соответствующее единицам стоящим на главной диагонали и оно единственное.
- Выделение подматриц - Полная – все эл. = 1 Основная – такая Полная, которая не содержится ни в какой другой. Покрытием Булевой Матрицы называется мн-во полных подматриц, покрывающих все единичные ее эллементы.
- Алгоритм Мальгранжа – для поиска всех основных подматриц – 1. Инвертировать исходную матрицу. Для определения полных подматриц необходимо выполнить операции AND&OR. Завершаем тогда, когда склеив дважды получаем тоже.

- Алгоритм сортировки пузырьком – 1. Найти строку с min суммой единиц. Ставим ее на место верхней. Для этой строки определяем сумму единиц по столбцам где стоят единицы и столбец с max суммой единиц ставим влево.2. Усекаем матрицу на соответствующие строку и столбец и продолжаем …
- 3D - При выделении основной подматрицы мы сосредотачиваем нули в верхней правой части, если выделять подматрицу x+y=N то кол-во единиц в ней будет минимальным. ЕСЛИ есть кластерная станция, каждый кластер объеденяет между собой процессоры, а кластеры между собой соединяются по общей шине => m/u эту методику для разбивки мн-ва задач на части и каждую часть отдать кластеру => минимизировать кол-во передач.

ОПТИМИЗАЦИЯ

- u/ Генетических алгоритмов, Метод оценочных ф-ций, Метод направленного поиска
- Генетические алг. – все решения располагаются в виде поиска. Есть базовое решение выбранное случайным образом, потом по специальной формуле берем ближайшее решение и потом исходим уже от него.
- Методъ Отжига (Anealing) – основан на фиксации базовой точки, расчитывается направлене, где находится ближайшая точка, лучше базовой и повторяем до тех пор пока не найдем решение удовлетворяющее нашим запросам. (-) – за известное кол-во шагов можно не получить нужное решение.
- Метод оценочных ф-ций – все решения пытаемся разбить по зонам и оценить вес зоны, и искать дальше в зоне решения. (-) – Не свегнда можно разбить и найти оценочную ф-цию с заданной скоростью счета.
- Метод направленного поиска – Есть поле решений. Можно идти от оптимального или от общего значения. Берем оптимальное значение и если оно нам не подходит то увеличиваем зону поиска: зона оптимизированных значений, если и здесь нет, тогда уходим в область приемлемых значений. СУТЬ: Сузить область поиска, отсекая неприемлимые значения.
- Метод поиска max парасочетания во взвешенном графе. Когда одна и таже задача на разных узлах может решатся за разное время, когда отличается стоимость решения задачи и т.д. => распределенная система. Вес – степень претендования заявки на ресурс должна быть вычеслена по формуле:??????
- Модифицированный Венгерский метод (n^1.5) Ищем max парасочетание с min суммой весов. 1. Получение нулей а) из каждого элемента столбца вычитаем минимальный элемент этого столбца.б)из каждого эл. строки вычитаем мин. эл-т этой строки (попытка получения оптимального варианта решения) 2. Исщем max парасочетание (делаем матрицу только из нулей) 3. Находим min опоры а) помечаем кружочком строку не содержащую отмеченного нуля. б) помечаем кружочком каждый столбец, содержащий зачеркнутые нули, какой-либо из помеченных кружочком строк. в)Помечаем кружочком строку, содержащую отмеченный ноль в помеченном кружочком столбце г) б и в 4. Выделение min опоры. Проведем линии через невыделенные кружочками строки и столбцы помеченные кружочками. 5. Расширение зоны поиска (добавление 0) Берем подматрицу, через которую не проходят эти линии, берем min эл-тов этой подматрицы, вычитаем этот эл-т из этих всех столбцов, черезкоторые линии не проходят, и прибавляем его к эл-там строк через которые они проходят ~8-|
- Модифицированный метод – для || RealTimeOS.(четко заданно время выхода заявки из системы – время ответа) u/ Венгерский метод. Исходные данные – на вход || системы поступаем мн-во работ. Каждая из которых хар. Временем прихода работы в систему, критическим временем выхода, и временем работы. Время выполнения i-той работы по отношению к узлу имеющему max производительность. Назначается штраф за единицу времени запаздывания. Матрица отношения работа-ресурс. И далее мутим-мутим, пока не придум к матрице решаемой по Венгерскому методу. Отсекаем значения заведомо х-вые.
- Метод Ветвей и границ.(NP) Есть базовое решение (А) из него формируем решение (Б) ти решение (!Б) и определяем перспективность данного направления в числах. ЕСЛИ на какомто шаге мы получили перспективность больше вышестоящего ТО возвращаемся назад и идем по другой ветке. Конец – когда доходим до max значения парасочетания с min весом.
- Эвристические (основанные на опыте людей), Пошаговое конструирование – разделение всего процесса решения на определенное кол-во шагов и нахождение оптимального или квазиоптимального решения. ЭТО – соответствует принципу оптимальности Белмана.
- Для задач назначения можно добавить стратегию мини-максного выбора добавив понятие надежного состояния системы.
- Составление расписания занятий (преподаватели, аудитории, лаборатории, время) => точка в пятимерном пространстве. => свести к двумерному пространству (булева матрица – может ли пройти занятие) и произвести обязательные назначения – выбираем строку находящуюся в наихудшем положении. Расчитаем степень претендования.
- Метод исключающего планирования – если планирование работает по шагам и шаги не возвратны то после первого шага можно задаче сразу кинуть на ресурс, не дожидаясь следующих шагов. (+) – Значительное уменьшение времени ожидания.

СТАТИЧЕСКОЕ ПЛАНИРОВАНИЕ

- Def. Джонсона – найти план распределения по ресурсам, при котором время решения не превышало бы критического с min кол-вом процов.
- Найти min кол-во ресурсов, чтобы задача решалась за min время.
- Для 2-х процов задача решается точно. Для одного в условиях RealTime.
- Исходная ин-фа -> Двудольный ациклический направленный граф ={описание вершин, описание дуг, описание (вес-время исполнения) каждой вершины, описание (вес-время пересылки) каждой дуги} ::= Многопроцессорное описание. Сложность решения – отсутствие ф-ции цели.
- Метод определения зоны нахождения оптимального решения – Декомпозиция – Распланировать задачу распределения, а потом задачу погружения. ОП – пересылки ==0. 1. Зануляем пересылки. Определяем Ткритическое. Определяем минимальное число процессоров, которое нельзя уменьшить.

Nlow=]Sum(Ti)/Tkp[2. Можно определить max число процессоров, выше которого практически не выгодно иметь больше процессоров – max ширина яруса. 3. Определим время решения, если число процессоров меньше Nlow. Треш=Sum(Ti)/Nkp. Nlow и Nhi надо скорректировать w/u тщательного анализа графа u/ понятие транзитности вершин графа (явная – можно переместить вершину на другой ярус без изменения критического пути, неявная, мультипликативная) и выполняем квазикластеризацию(соединение двух узлов в один) для этой дуги => пересылка зануляется. Выбираем кластеризацию дающую min время планирования.

- Алгоритм Янсека: Противоположен предидущему. Загружаем все на один процессор, а потом распределяем по процессорам.
- Алгоритм планирования по спискам (оценочных ф-ций) для взвешенного графа. U/ стратегию раннего планирования в сочетании с анализом связности графа. Проблема – число про-цов m/b < ширины яруса. Берем вес узла и определяем сколько процессоров от него зависит, на какие он влияет. Расчитываем вес и прибавляем те веса, на которые влияет узел и определяем оценочную ф-цию и ставим на процессор. Задача имеет точное решение для древовидного графи и для двух процессоров (algo Джонсона)
- Метод критического пути 1 – Статика –нашли путь и и-ец 2 – динамический – поиск пути изменяющегося динамически. Берем исходный граф, находим критический путь и грузим на один процессор и пересылки удаляем. Вычисляем критические пути из оставшихся вершин без изменения предидущего и т.д. Нужно учитывать условия предшествования – статика. Динамика – альтернативные перестановки вершин на процессорах для улучшения времени планирования.
- Алгоритм пошагового конструирования с локализацией зоны поиска решения: u/ базовое решение – всем задачам по процессору, уменьшить число переборов вариантов, для нахождения оптимального решения, число шагов приблизить к числу узлов графа и сделать его невозвратным. Перебор со всеми вершинами и нахождеие пути при котором будет минимальное время планирования, просматриваем граф снизу, определяем вершину которая определяет время, делаем псеевдосвязь определяющую предшествование
- Другой метод: Анализ длинны графа, веса вершин в графе не одинаковые, не превышают максимального веса вершин в ярусе, т.е Sum весов этих объединенных вершин не должна превышать max числа в ярусе.
- Третий метод: ЕСЛИ имеются || ветви ТОГДА в графе выделить || ветвь, состоящую из вершин и сумма времен решения этих вершин не превышает max числа вершин рядом стоящего яруса ТО можно объединить в один кластер. Обнуляем вершины, ставим там точку, оставляем пересылки. Выделяем каждому узлу процессор и берем две топологии с общей шиной и полносвязную систему. Для полносвязной системы худшее время решения задачи = Ткритическое. (max путь по графу). Для шинной топологии Ткр=Sum(Вершин)=Thi. Получение третей точки – берем граф в ярусно-последовательной форме. Берем самый широкий ярус, закрепляем каждую вершину этого графа за процессором. От каждой вершины строим критический путь вверх-вниз, и на процессор от соотв. вершины ставим вершины этого пути, исключая решенные вершины, и обнуляем вершины, которые вошли в критический путь. Проделываем эти действия для каждой вершины самого широкого яруса.

РЕШЕНИЕ ЗАДАЧ СТАТИЧЕСКОГО ПЛАНИРОВАНИЯ В СИСТЕМАХ С ОП

- Пересылки зануляем. Определяем критический путь. (Дэйкстра – только один путь)(Лойд-Уоршил – все пути n^3). Ранее планирование – задача запускается на процессор как только она может запустится. Позднее планирование – задача задерживается для запуска до тех пор пока ее задержка не будет влиять на критическое время. Признак того, что вершина готова к обработке – у нее в столбце все нули. После первого такта зануляем первую строку. Сравниваем два графика ЕСЛИ вершины имеют одинаковое время старта то они входят в критический путь) Все вершины входящие в критический путь помещаем на первый процессор. Для каждой вершины посчитать временную степень свободы. Пытаемся справа-налево убрать задачу с процессора. – Наполнение карманов – Рюкзаков.
- Полная задача статического планирования. Граф имеет веса вершин и пересылок . Надо решить все задачи о которых он говорил. W/u условие предшествования без прерывания задач, определить минимальное время и минимальное число процессоров.
- Идея алгоритма Шаркара-Янга: 1. Погружаем исходный граф с учетом условий предшествования в ВС имеющую кол-во процов. Равное кол-ву узлов в графе. 2. Рекурсивно рассматриваем все дуги начиная с большей определяем связи и делаем альтернативную кластеризацию этой вершины с вершинами, с которыми у нее есть связь и определяем каждый раз время планирования и там, где оно меньше кластеризуем нижнюю вершину. 2. Рассматриваем кластеризованные вершины как кластер и назначаем ему номер из вершин входивших в него и продолжаем итерацию, пока не дойдем до верхней вершины. Возможны варианты: ЕСЛИ при очередном шаге кластеризации, кластер становится разрывным ТО выполнять ликвидацию разрыва ПУТЕМ подъема вершины, которая держит нижнюю, и поднять нижнюю. Как только дошли до конца мы можем получить неразрывный или разрывный кластер. Т.е. мы получили || не получили оптимальное решение
- Метод одиночных ф-ций – для каждой вершины надо вычислить приоритет погружения. Берем вес вершины и добавляем веса веса вершин по пути, и их кол-во. Чем больше вес тем меньше приоритет.

МОДУЛЬНЫЙ ПРИНЦИП ПРОГРАММИРОВАНИЯ

- Первое применение – разработка трансляторов. Развивался – при проектировании ОС.
- Идея: Есть прога. в виде кусков. Каждый кусок m/ выполнятся с определенными соглашениями. Если этот кусок имеет стандартную структуру, параметрическую универсальность, ф-циональную завершенность => u/ SUBJ.
- Модуль д/ иметь связи по данным и по управлению(w/u common reg's) Надо запомнить адрес возврата, Организация действий по осуществлению возврата из вызываемого осуществляется вызываемым модулем. Выполняется w/u области сохранения. ЕСЛИ модуль А вызывает В ТО В должен хранить в себе ин-фу о возврате в А. В начале модуля есть структура - настройка параметров, сохранение, организация связи, передача параметров.

- Связь по управлению – связь для организации возврата. Возврат модуль В должен иметь команду, делающую так, что в области сохранения модуля А сохраняется состояние регистров. В модуле В есть адрес области сохранения в модуле А. Сохраняется адрес возврата, для возврата адреса следующей команды.
- Связь по данным – данные через общий регистр || через системный регистр передается адрес списка параметров.
- Виды модулей в системе (различные библиотеки модулей) ->
- (Исходный)->ТРАНСЛЯТОР->(Объектный)->РЕДАКТОР_СВЯЗЕЙ->(Загрузочный)->ЗАГРУЗЩИК->(Абсолютный)
- Изменилась система управления памятью => загрузочный модуль требует настройки адресного пространства (EXE)
- Функции загрузчика : распределение памяти, настройка адресных const's,связывание, загрузка. Абсолютные, Настраивающие, Непосредственно связывающие
- Настраивающий – работает в загр. Модуле которому больше ничего не надо. Постепенно его ф-ции взял на себя редактор связей.
- Непосредственно связывающие – инфу – для них готовит компилер. (в спец. виде.)
- Непосредственно в настраивающем загрузчике каждый модуль может транслироваться отдельно. Чтобы передать сообщение редактору связей надо ему непосредственно указать, что надо транслировать.
- В каждом модуле в начале трансляции выделяются вектора перехода, внешние и внутренние.(Экспорт и Импорт процедур и ф-ций). Кроме того для выполнения настройки каждая команда отмечается битом переместимости. ОС выделяет и пользуетс глобально выделенной памятью, а загрузчик с локальной.
- При динамической загрузке возможны стратегии глобальной и локальной настройки адресных констант (актуально для сегментной организации памяти) – Глобальная – настройка всех адресных констант при загрузке; Локальная – настройка только при исполнении. Проблема настройки связана с перемещением программ в ОП, ЕСЛИ ОС работает со свопингом – нужна перенастройка адресных констант. (глобальная – перенастройка всех адресных констант, локальная – вычисление адреса той переменной, которая находится реально в ОП)

ГЕНЕРАЦИЯ-ИНСТАЛЯЦИЯ-ИНИЦИАЛИЗАЦИЯ

- ОС проектируется 4/u ее во всем наборе режимов, на которые она проектировалась, и на всем оборудовании, которое выпускается. Она д/ работать на любой установке. ::= ДИСТРИБУТИВ (исходный вариант) ОС
- ОС работает на конкретном оборудовании, с конкретной архитектурой и использованием конкретных режимов. Из ДИСТИБУТИВА выбирают только те модули, которые поддерживают конкретную конфигурацию – ГЕНЕРАЦИЯ ОС.
- РЕЗИДЕНЦИЯ – ОС полученная в результате генерации. Место нахождения резиденции – РЕЗИДЕНТНЫЙ ТОМ.
- При работе с ОС всю резиденцию нет смысла хранить в ОП – необходимость хранения ОС в ОП – требование которое пользователь предъявляет к ОС. Из резиденции выделяется совокупность программ – СУПЕРВИЗОР (управляющие проги), а в ОП грузится ядро супервизора, включающие только те программы, которые отвечают за реакцию системы.
- ГЕНЕРАЦИЯ системы выполняется w/u языка генерации.
- ИНСТАЛЯЦИЯ – локальная перенастройка резиденции системы.
- При загрузке ядра ОС, необходимо, чтобы система знала, где находится активный раздел. (Для его определения u/ любая модификация бутового загрузчика)
- Потом выполняется загрузка ОС – основная цель – выполнить инициализацию ядра и ОС.
- Инициализация ядра – для формирования системных таблиц, в которых сосредоточена вся информация о программах супервизора и всех динамически меняющихся параметров. – u/ табличный метод управления
- Табличный метод управления – принятие управляющих воздействий на основании ин-фы, сосредоточенной в спец. таблицах (sys). Кол-во и структура таблиц определенна разработчиком системы. (IDT,Процессов, Дескрипторная файлов, Памяти, ВУ, Таблица таблиц.)
- Резидентные проги. – находятся в ОП, поддерживают быстрый отклик системы, Проги обработки прерываний, Процессы 4/ работы с ВУ, управление памятью, управление процессами, начальная прога. Управления заданиями.
- Транзитные – проги., которые могут понадобится 4/ выполнения ф-ций ОС (но не резидентные) – могут вызываться по оверлейной || динамически-последователььной схеме в транзитную зону. (раньше выделялось после ядра жесткая область 4/транзитов – теперь – транзиты грузятся в любую область ОП) => возможно загружать в свободные блоки (области) BIOS – например в видяху.

ИЗМЕНЕНИЕ ТЕХНОЛОГИИ I/O

- Раньше в ОС u/ прямой i/o (Проц. сам выполнял i/o) Изменения начались после появления логики прерываний => стало возможно u/ мультипрограммирование и появились спец процессоры i/o – КАНАЛЫ =>!!! Распараллеливание i/o с работой основного проца.
- DMA – при блочной передаче приостанавливает проц. Символьные ус-ва прерывают проц.
- Для уравнивания скоростей между ВУ и процом. u/ буферизация (двупортовая память || RoundRobin- двойная буферизация (две микрухи – в одну пишем из другой читаем) – меньша подверженость тупику, чем одиночная)

БИБЛИОТЕКИ

- Впервые появились 4/ компиляторов и были встроены в них. Состав: Каталог, Сама Библиотека, Управляющая программа – Библиотекарь. Основные ф-ции – включение, исключение …
- Постепенно стали превращаться в FS – кроме основных ф-ций (поиск,u/) решают задачу оптимизации.

ОБЩАЯ СХЕМА ЗАГРУЗКИ M$-DOS(сокращенно)

- Проц – Грузит BIOS (FFFF:0000) -> POST -> Int 19h (Boot from HDD||FDD) -> MBR (Boot) -> io.sys (логический и-фейс между sys и dev w/u BIOS резидентные драйвера основных ПУ) ->msdos.sys (FS,MEM,Dev,Process Control) -> command .com – shell; связный список драйверов из config.sys -> EXEC command.com -> autoexec.bat
- Схема памяти (Вектора, Область БИОС, Область ДОС, io.sys msdos.sys, Дрова, Коммандир – COM, транзиты(под 640 кб),Граф-буфер, UHB,ПЗУ,UMB)

- После загрузки command.com и autoexec.bat система готова к работе. Срабатывает Командир при <CR> - интерпретирует и передает команду ОС. В следствии дырок в памяти sys позволяет запихнуть туда дрова.

ОБЩАЯ СХЕМА Ф-ЦОНИРОВАНИЯ ОС

- Во время загрузки – грузятся только программы необходимые 4/ быстрой реакции системы.
- Загрузка sys (активизированы системные задачи и ядро в памяти) Супервизор – совокупность управляющих программ, поддерживающих ядро системы
- Табличный метод управления – Состояние ВС -> Таблицы -> Решения.
- Ввод система понимает на уровне языка системного ввода. Определение правомерности задачи. U/ библиотеку языка управления заданием. ЕСЛИ задание неверное ТО оно via главный планировщик направляется на(в)зад usaru.
- Задание во второй очереди начинают обрабатываться при наличии в системе ресурсов. Система загружает планировщик по транзитной схеме, который обрабатывает входную очередь, для выборки наиболее приоритетного задания для загрузки.
- При работе программ системного ввода u/ режим спуллинга – согласование скоростей на входе и выходе.
- Инициатор – фиксирует блок управления задачей. Как только есть блок – Процесс готов – ему надо только время проца.
- 3 уровень – выделяет задачам время прца. Прога системного вывода работает в режиме спуллинга, ЕСЛИ задания выдают ин-фу на ус-ва системного вывода то создается выходная очередь заданий.

СТРУКТУРА ОС (СПО) – Обработка и Управление

- Обработка – изменение ин-фы с которой работает прога в составе ОС (дрова, загрузчик)
- Управление : Заданиями – слежение за прохождением заданий от входа до выхода на всех этапах его выполнения. Задачами – (Прцессами) – слежение за всеми задачами активизированными в системе и процессами их выполнения на ресурсах. Памятью – решение задач эффективного u/ mem (internal) в соответствии с ее организацией.(Защита – согласование ин-фы в кешах) Данными – эффективное размещение и u/ данных на внешних носителях (проблема эффективности u/ процессора) Внешними ус-вами …