Системни средства за обработка в реално време

Васил Георгиев, ФМИ 110

v.georgiev@fmi.uni-sofia.bg
is.fmi.uni-sofia.bg

Съдържание

1. РВОС – характеристики
2. Планиране в РВОС
3. РВ-платформи в стандартни ОС
4. Бази данни в РВ/ВАС

Операционни системи за реално време (РВОС) - характеристики

- планиране на заданията (scheduling) – планира процесорното време между ядрото (което го изпълнява), резидентните драйвери на устройства (вкл. апаратните контроли – напр. MMU) и активните приложения
  - винаги прекъсващи приоритети
  - време за изпълнение на задание
  - период на задание (при цикличните задания като при индустриални роботи) – респ. такт на задание
  - закъснение – между регистрацията в сензора и изпълнението на конролното устройство
- изпълнение на прекъсване – времето за предаване на управлението на съответната процедура на прекъсването
- време за превключване на заданията (презереждане на контролния блок на заданията)

Статично планиране в РВОС

- прилага се при циклично повтарящи се задания
- алгоритъм с монотоннен темп (rate monotonic, RM): нарастващ приоритет по темпа на заданията – най-често повтарящото се задание прекъсва всяко друго
- при статични периоди и брой на заданията може планирането е напълно детерминистично
- може априори да се прецени дали съотв. задание ще се изпълни в критичния период
- ниско натоварване на процесора – напр. при 4 задания 69%
- неподходящ при променливи периоди на заданията

Динамично планиране в РВОС

- първо най-къс срок (Earliest-Deadline First, EDF)
  - най-добро натоварване на процесора
  - не може да се определи кое задание ще пропадне при моментно претоварване
- първо най-спешните (Minimum-Laxity First, MLF)
  - laxity = deadline time - current time - CPU time needed; L=0 → незабавно изпълнение за включване в срока
  - добро натоварване на процесора
  - не може да се определи кое задание ще пропадне при моментно претоварване (но по-добър от EDF)
- първо най-критичните (Maximum-Urgency-First, MUF ~Solaris)
  - хибриден статично-динамичен критерий: urgency = F(criticality,1/L, user priority)

Хибридно планиране – MUF

- първо най-критичните (Maximum-Urgency-First, MUF ~Solaris)
  - хибриден статично-динамичен критерий: urgency = F(criticality,1/L, user priority)
  - criticality: статична оценка при начално стартиране на заданието
    - подрежда заданията по период – като RM – от тези с най-висока честота (най-кратък период)
    - определя критичните N задания – тези които няма да пропаднат и при моментено претоварване
    - на тези N задания се присвоява висока критичност
    - допълнително (с цел уникална стойност на F) може да се отчете и приритет на потребителя, който инициира заданието (потребителите в такава РВОС са с уникални приоритетеи)
    - критичността е статична оценка (преди стартиране на заданието; L е динамична при всяко рестартиране на задание и L=0 означава моментално прекъсващо стартиране на съотв. задание
  - така планирането е:
    - първо тези с L=0 (ако има)
    - след това тези с максимална критичност
    - ако има задания с еднаква критичност – те се подреждат по L
    - ако има задания с еднаква критичност и L – те се подреждат по потребителски приоритет

Отстраняване на планиращи грешки в РВОС

- рестартиране на провалените задания
- съобщение за планираща грешка към системата с рестартиране на планиращия проце
- промяна приоритета на заданието без рестартиране
- рестартиране на системата със запазване на статуса на заданията
- подържане на одит-информация за анализ на грешката
- при някои итеративни системи в периоди на свръхтовар системата връща апроксимирана стойност на резултата (по предишните итерации) – с намалена точност без оценка на съответния алгоритъм

Защита на паметта за РВ процеси

- РТОС осигуряват контрол на достъпа до виртуалните страници на всеки процес (нишките от един процес имат достъп до всички негови старници) – процесен модел за ЗП
  - нишка (или процес) обръщение към външна страница се блокира и се генерира системно съобщение
  - процесния модел изисква поддържане на виртуална памет с транслираща таблица на виртуалните и физическите адреси, което обуславя следните типове свръхтовар:
    - поддържане на транслиращата таблица (ВА, ФА, разрешения за достъпа, процес)
    - превключване на процесите – смяна на ТТ в MMU
- при ВАС се прилагат и упростени модели на транслация ВА/ФА с резервиране на свързано адресно пространство за всяка нишка (което е възможно само при упростената йерархия на паметта – без дискове)
  - така вместо сложен списък от страници се поддържа списък с границите на всеки нишков контекст (фиг. 6.8)

Разширения на ядрото за РВ процеси – INtime/Windows

- изисквания за РВ-Windows
  - многонишково планиране с 64-128-256 нива на приритет и система за наследяване на приоритета между нишките на процесите
  - бързи системни таймери и тактове
- РВ недастатъци на Windows XP/Embedded
  - малък брой приоритети (32)
  - ниско ниво на темпорален детерминизъм на планирането
  - приоритетни инверсии – особено при прекъсване
- системната архитектра се разширява с РВ-виртуална машина (ВМ на Windows-ядрото + РВВМ конкурентно ядро) за високоприоритетни РВ процеси - фиг. 6.9.
- Windows-ядрото и изпълняваните от него процеси са само един ниско-приоритетен РВ процес за РВ-ядрото
- варианти за симетричен мултипроцесинг с резервация на процесори – време за реакция до 3 мкС
- достъпът до РВ-ядрото е чрез API, което поддържа обмена и планирането на процеси, нишки, семафори и други областти от паметта

Разширения на ядрото за РВ процеси – RTX/Windows

- РВ-модула се вгражда не като обвивка на ядрото (т.е. суперядро – като INtime), а като резидентен драйвер RTX, който поддържа интерфейс към РВ-планирането – RTSS (Real time subsystem) – фиг. 6.10.
  - с 128 нива на приритет, планиращия процес е с приоритет 127
  - свръхбързо планиране "без" закъснение
  - преодоляване на приоритетната инверсия чрез повишаващ приоритет за нископриоритетните нишки с достъп до високоприоритетни обекти
- производителност на RTX (Пентиум 3 – 800MHz)

Операция (мкС, мин/макс)	**Windows XP**	**Windows CE**	**RTX 5.5**
освобождаване на mutex	1.39 / 5000	4.9 / 13.3	0.70 / 3.26
освобождаване на semaphore	1.39 / 5000	3.9 / 8.4	0.61 / 3.43
прекъсване	4.3 / 5000	13 / 30.8	2.0 / 19

Бази данни в РВ и ВАС

- по принцип справките в корпоративни и лични *релационни* СУБД (Oracle/Access) се избягват в РВ и ВАС приложния поради недетерминираност и многослойна ресурсоемка архитектура
- прилагат се специализирани СУБД, които
  - имплементират хибридни модели с ускорени справки в сервизния слой и/или
  - тънък клиентски слой – напр. чрез включване на СУБД-функциите в основното приложение, вместо с обръщение към самостоятелна резидентен интерпретатор
- напр.:
  - SQLite (релационна) – http://www.sqlite.org/ и http://en.wikipedia.org/wiki/SQLite
  - Raima Database Manager++ (мрежова), http://www.linuxjournal.com/article/2325 и http://www.raima.com/products/)

Изисквания към РВ/ВАС СУБД

- памет – библиотечни функции, които се свързват статично с клиентския код и могат да се редуцират до 100-150 КБ обектен код (обикн. от C/C++)
- производителност – по-кратък цикъл на отговор в СУБД се постига чрез контрол върху операциите от ниско ниво и оптимизиран модел на данните
- надеждност – защита и свързаност на данните, самовъзстановяване и съхраняване на частично изпълнени транзакции
- детерминистичност – зависи от размера на данните, задава се със статистически резултати от еталонни тестове –
  - обичаен подход е да се прилагат записи с фиксирана настройвама дължина
- ниво на услугите – обикновено ниско – редуцира се свръхтовара, увеличава се детерминираността (при по сложно програмиране на клиента)

Релационен модел на данните в СУБД

- стандартен модел (http://en.wikipedia.org/wiki/Relational_model) с най-широко приложение в гобемите СУБД, но с ограничено приложение за РВ/ВАС СУБД
- индексиран достъп до записите по колони и редове
  - най-често двумерни таблици
  - при справка в повече от една таблица свързващата операция репликира съответните колони и създава нова таблица
  - търсенето **`select-from-where(-[and|or|join|orderby])`** протича на няколко фази
    - първо се проверяват по индекс и извличат записите от първата таблица, отговарящи на ключа от where-клаузата
    - след това с всеки от извлечените записи като ключ се проверява индексен файл и извлича указатели за съответсващите записи във втората таблица
    - тези операции се повтарят рекурсивно за всяка клауза
  - проблем в производителността се създава при проверката във индексния файл, която може да отнеме 2-3 итерации за файлов достъп до всеки запис от таблицата, който отговаря на клаузата
  - висока надеждност и достоверност на дааните
  - ниска производителност и висок разход на памет при създаването на свързващите таблици

Мрежов модел на данните в СУБД

- базира се на структуриране на данните като мрежа от множества (т.е. без индексиране) - http://en.wikipedia.org/wiki/Network_model, http://en.wikipedia.org/wiki/Navigational_database
- множеството е свързан списък, представящ отношение един-към-много
- главата на списъка съдържа указателите на всички елементи и всеки елемент съдържа указателя на главата – така достъпа до всеки елемент (и откриване на принадлежността на даден елемент) отнема една итерация или операция
- тъй като множествата се обхождат в определен ред, производителността зависи от структурирането на контекста
- индексnте релационни модели са ефективни при произволен достъп или при клауза за сортиране, а мрежовия модел – при един-към-много, много-към-един и рекурсивните връзки в контекста