25. Tétel

Ismertesse a QNX alapvető architektúráját és az üzenetek segítségével történő kommunikáció folyamatát!

QNX Neutrino

* jelenlegi generáció: QNX 6.x.x (6.4.0)
* OS neve: QNX Neutrino
* teljes rendszer neve: QNX Mommentics
* POSIX szabványt követ
  + egységes szemlélet
  + nem UNIX, de hasonló
* RT
* support:
  + beágyazott rendszerek
  + nagy teljesítményű elosztott rendszerek
* támogatott CPU-k:
  + x86
  + ARM
  + XScale
  + PowerPC
  + MIPS
* hordozható kód a verziók között
* többfelhasználós
  + virtuális konzol
  + futó folyamatok többfelhasználósak lehetnek

Mikrokernel

monolitikus ⬄ mikro

* monolitikus
  + egy mag ban az összes szükséges rendszerszolgáltatás (fájlrendszerek, kommunikáció, eszközkezelés)
  + közös memória terület
  + korai Linux kernelek (ma már moduláris, bár nem mikrokernel)
  + Windows NT kernel (performancia miatt. bár némi primitív mikrokernel-je van a drivereknek)
  + performancia
  + ha bukik, minden bukik 🡪 kékhalál 🡪 ☹
  + általában stabilabb, megbízhatóbb
* mikrokernel
  + a HW felett minimális réteg csak az OS magja
  + feladata:
    - alap rendszerhívások
    - memóriakezelés
    - processzekkel kapcsolatos műveletek
    - processzek közötti kommunikáció
  + a rendszerszolgáltatások ugyanolyan szoftverek, mint a felhasználói programok
  + előnye:
    - védeni lehet az OS-t és a különálló SW komponenseket
    - bármilyen szolgáltatás leáll, nem hal meg a kernel. Újraindítja, és működik minden tovább (bár így kisebb hibák simán elfedődhetnek a fejlesztő előtt, mert mindig megoldódnak)
  + hátránya: túlzott számú rendszerhívás + túl gyakori kontextusváltás 🡪 nagy overhead

QNX mikrokernel

* az OS magja egy busz
* mindenki erre kapcsolódik
* mindenki ide küldi az üzeneteit
* az SW komponensek egyenrangúak
* a kernel magától sosem kerül ütemezésre, csak …
  + közvetlen kernelhívás
  + exception
  + HW IRQ hatására
* alapvető működése:
  + üzenetkezelés
  + egyes üzenetekhez nem társít külön értelmezést, azokat csak a küldő-fogadó érti
  + feladata ezek szinkronizálása, ütemezése, erőforrások megfelelő kihasználása
  + párhuzamosan futó szálak
    - számítógépen belül és hálózaton keresztül is képesek kommunikálni egymással
    - bármilyen erőforrást használhatnak (helyi, hálózati)
* komplex rendszerek üzemeltetésére

Kernelszolgáltatások

Threads

* alapegység: process 🡪 szálak tárolója (egy processz – min. egy thread)
* egy processzhez tartozó összes thread egy memóriaterületen
* a threadeknek egyedi stackje van
* lehetséges állapotok
  + CONDVAR – feltételes változóra várakozik
  + DEAD – befejeződött, másik szál csatlakozására vár
  + INTERRUPT – megszakításra vár
  + JOIN – blokkolt és egy másik szálhoz való csatlakozásra vár
  + MUTEX – egy mutex miatt blokkolt
  + NANOSLEEP – alszik egy kicsit
  + NET\_REPLY – hálózati válaszra vár
  + NET\_SEND – hálózati jelre vár (pulse, signal)
  + READY – futásra kész, épp valaki magasabb prioritású dolgozik
  + RECEIVE – üzenet érkezését várja
  + REPLY – elküldött üzenetre vár választ
  + RUNNING – épp fut
  + SEM – egy szemafor küldésére várakozik
  + SEND – üzenetet küldött, a kézbesítésre vár
  + SIGSUSPEND – blokkolt állapotban van, egy jelre vár (sigwaitsuspend())
  + SIGWAITINFO – blokkolt állapotban van, egy jelre vár (sigwaitinfo())
  + STACK – a szál arra vár, hogy a hozzá tartozó stack le legyen foglalva (a szülő pl. ThreadCreate() fv-t hívott)
  + STOPPED – a szál blokkolt és SOGCONT jelre vár
  + WAITCTX – arra vár, hogy nem fixpontos tartalom használható legyen
  + WAITPAGE – fizikai memórialapra vár
  + WAITHREAD – a gyermek szál létrejöttére vár (ThreadCreate() )
* Ütemezés
  + 256 prioritási szint
    - user processes: 0-53
    - idle: 0
  + azonos prioritási szinten
    - FIFO: a szál addig fut, amíg nem blokkolódik, vagy egy magasabb prioritású futásra késszé nem válik
    - Round-Robin: A szál addig fut, amíg nem blokkolódik, egy magasabb szál meg nem szakítja, fel nem használja az időszeletét
    - Sporadikus: megadja, hogy egy adott időtartamon mennyi ideig futhat a szál 🡪 ha ezt elhasználta, alacsonyabb prioritásúvá vállik, és a feltöltési idő után újra visszakapja a prioritását

Signals

Message transmitting

Thread Synchronization

* Mutex – kölcsönös kizárás – csak egy szál lockolhatja az erőforrást, a többiek várakoznak. Ha egy magasabb prioritású kezd el várakozni, ideiglenesen az aktuálisan foglaló prioritása afölé emelkedik – prioritás inverzió
* Condvars – feltételes áltozó, egy esemény/feltétel beteljesülésére átbillenő változó (mutex-el együtt kell használni)
* Barriers – sorompó. A szálaknak be kell várniuk egymást egy adott ponton. Az inicializáló szál adja meg, hogy hány további szálnak kell megérkeznie. Ha a feltétel teljesül, akkor minden szál egyszerre folytathatja a tevékenységét. Mátrixműveleteknél hasznos
* Sleepon locks – condvarhoz hasonló. Nem csak egy feltétel teljesülését figyeli, több feltételes változó értékét figyeli (változók száma max a várakozó szálak száma lehet)
* Reader/Writer locks – írás olvasás külön lockolása
* Semaphores – általános lefoglalás/felszabadítás szinkronizáció
* FIFO scheduling – nem SMP rendszerekre szálak szinkronizációjára. Egy prioritási szál addig fut, amíg egy magasabb prioritású meg nem szakítja, vagy blokkolt állapotba nem kerül. Nem tartalmaz semmilyen explicit szinkronizációs mechanizmust.
* Send/Receive/Reply (Messages)
* Atomic operations: Műveletsorok megszakítás nélkülivé tétel

Scheduling

Timer Services

Process Management

* IPC

Üzenetek segítségével kommunikáció