# 25 – 26. Tétel

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

26. Ismertesse a QNX Inter Process Communication lehetőségeit!

## 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 – Inter Process Communication

* **Message parsing – szinkron üzenettovábbítás (kernel szolgáltatás)**
  + üzenetküldő (kliens) – fogadó (szerver)
  + üzenetküldési lehetőségek
    - egyszerű üzenet küldése: memóriacímmel + hosszúsággal
      * a rsz. nem foglalkozik az adatokkal, csak átmásolja a kliens memóriatartományáról a szerverére az adatot – a művelet szinte csak memóriakezelés 🡪 gyors
    - több részből álló üzenet küldése
      * I/O vektor az egyes üzenetrészek címével + hosszával
  + üzenetküldés folyamata: kliens küld egy üzenetet + blokkolt állapotba kerül, amíg nem kap receive-t 🡪 szerver fogadja, és válaszol rá (ha épp nem egy másik üzenetre vár, ami blokkolja)
  + hálózaton keresztül
    - QNX natív hálózata: Qnet
    - lassabb, de kényelmes
  + csatornák
    - szerver szálnak kell létrehoznia 🡪 kliens csatakozik hozzá
    - Pulse:
      * rögzített méretű adatok továbbítására (1 byte kód + 4 byte adat)
      * nem blokkolja a résztvevőket
      * interrupt handlerekben szeretik használni
      * prioritásöröklődés – prioritás inverzió elkerülése miatt
  + alapszabályok
    - két szál közvetlenül ne küldjön üzenetet egymásnak
    - a szálakat hierarchiába rendezzük, és mindig az alacsonyabb küldjön üzenetet a magasabbnak 🡪 deadlock elkerülése
* **Signals – jelek (kernel szolgáltatás)**
  + olyan, mintha SW megszakítás lenne
  + a jeleknek is handlerei vannak, amit a kernel hív
  + a végrehajtható feladaot, hívások korlátozottak (mert kernelben hajtódik végre)
  + jelenként különböző az alapértelmezett tevékenység
  + a kernel 64 jelet támogat (32 alapvető POSIC jelet)
  + 1 bájt kód + 4 bájt adat, ami várakozási sorba kerülhet (mint a Pulse)
  + működési szabályok
    - a jelek kezeléséhez kapcsolt tevékenység az egész folyamatra érvényes
      * lehet figyelmen kívül hagyás
      * lehet feldolgozás
    - maszkolásuk szál szintjén érvényes
    - közvetlenül szálnak küldött jel csak adott szálnak kézbesíthető
    - folyamatnak küldött jel az első olyan szálhoz kerül, amelyik nem blokkolja a jelet
* **POSIX message queues – (külső folyamat szolgáltatása)**
  + külső szerver folyamat végzi a kezelést
  + két szerver implementáció
    - mqueue – hagyományos erőforrás kezelő
    - mq – alternatíva
  + az üzenet sorok névvel rendelkeznek + elérhetőek az állományrendszereken keresztül is
    - /dev/mqueue/
    - /dev/mq/
* **Shared Memory – (külső folyamat szolgáltatása)**
  + legnagyobb sávszélességet biztosítja
  + hozzáférés asszinkron – szinkronizálni kell (szemaforral, vagy mutexel)
  + egy processzen belül az összes szál azonos memóriaterületen osztozik – elérik egymás területét
  + Process Manager intézi
  + használata
    - létrehozás 🡪 fájlleírót ad: shm\_open()
    - méret kezdetileg nulla 🡪 be kell állítani: shm\_ctl()
    - elkészült a memóriaszelet 🡪 be kell állítani, hogy elérhető legyen: mmap()
    - védelmi paraméterek
      * végrehajtható program?
      * írható? olvasható?
      * gyorsítótárba kerülhet-e a tartalma?
* **Pipes – (külső)**
  + IO csatorna folyamatok között
  + külső szoftverkomponens menedzseli
  + shell haszálja általában: egy parancs kimenetét egy másik bemenetére tegye
  + egyoldalú kommunikációk
* **FIFOs – (külső)**
  + hasonló a csövekhez
  + ugyanaz a szoftver kezeli
  + különbség: névvel rendelkeznek, elérhetőek a fájlrendszeren keresztül

### Interrupt handling

* feladat: a külső eseményekre reagálás 🡪 késleltetés a rendszerben
  + minél gyorsabb
  + minél hatékonyabb
  + minél rövidebb ideig tartson
* Megszakítás késleltetés – Interrupt Latency
  + a megszakítás után nem azonnal hajtódik végre a lekezelés
  + függ
    - le vannak-e tiltva a megszakítások
    - éppen egy másik megszakítás fut
* Ütemezési késleltetés – Scheduling Latency
  + megszakítás kezelés után egy magasabb szálat kell végrehajtani (valamilyen esemény segíségével)
  + késleltetés a megszakítás kezelő utolsó utasítása és az eredeti szál első utasítása között eltelt idő

## Rendszerszolgáltatások

### Fejlesztői kernel

* kernel események megfigyelése valós időben
* minimális méretnövekedés
* 97-98%-a valós kernel sebességének
* naplózás

### Többprocesszoros rendszerek

* akár diszkrét akár többmagos
* működési módok
  + aszimmetrikus – különböző OS-ek vagy azonos rendszer különböző példányai futnak egyes processzorokon
  + szimmetrikus – egy OS több CPU-n, processzek, threadek tudnak váltani az egyikről a másikra
  + csatolt többprocesszoros – az OS egy példánya kezeli a CPU-kat, de az alkalmazások nem tudnak futás közben váltani

### Process Manager

* procnto SW modul
* feladatai
  + folyamatok kezelése (létrehozás, felszámolás …)
  + memória kezelése
  + névtér kezelése (azonos névtérben vannak a háttértárolók, perfiériák, rendszerkomponensek, rendszerszolgáltatások)

### Dynamic linking

* hagyományos C tulajdonság – libary-k közös használata, nem statikusan mindenhova bemásolva

### Resource Manager

* HW perifériák, virtuális eszközök kezelése
* nem igényel speciális kapcsolatot a maggal – leálítható
* támogatott erőforrások (ahogy ő bontja fel)
  + device
  + filesystem

### Filesystems

* állományrendszerek kezelése a kernelen kívüli SW modulban
* ennek előnye:
  + egyes állományrendszerek igény szerinti elindíthatóság, leállíthatósága
  + általános kommunikációs megoldásokkal elérhetőek
  + általános névtéren keresztül elérhetőek az állományrendszerek kacsolódási pontokon keresztül
  + egy állománykezelőből több tucat futhat egyszerre
  + hálózton keresztüli elérés könnyen biztosított
* támogatott állományrendszerek
  + image – OS modulok (pl. boot image)
  + block – blokkorientált állományrendszerekhez (merevlemezek, optikai meghajtók)
  + flash – FFS2 for NOR devices, ETFS for NAND devices
  + network: távoli hozzáféréshez: NFS, CIFS
  + virtuális:
    - csomag
    - tömörített

### Karakteres IO

* elérési útvonal állományként: /dev/
* eszközök csoportosítása/elérhető eszközök
  + soros vonali terminál
  + párhuzamos port (nyomtató pl.)
  + szöveges üzemmódú konzol
  + pszeudo terminál

### Hálózatkezelés

* kernelen kívüli app
* egységes interfész az összes áhlózati kapcsolathoz
* Qnet
  + natív
  + átlátszó hálózat – az egyes csomópontok szorosan kapcsolódnak egymáshoz
  + legfontosabb feladatai
    - állományrendszerek távoli elérése
    - alkalmazások rugalmas skálázhatósága
    - elosztott alkalmazások készítése
    - processzor erőforrás megsokszorozása
  + saját névfeloldás, de támogatja a TCP/IP DNS-t is
  + nem routeolható
  + bővítés: Internet Protocol-ba ágyazás
* TCP/IP hálózat