Valósidejű rendszerek mintazárthelyi

Készítette: Koszó Norbert (k.norbi@gmail.com)

Jelen mintazárthelyi a 2009-ben feladott feladatokat tartalmazza és az általam helyesnek vélt megoldásukat. Semmilyen garanciát nem vállalok a helyességükre, a biztos információkat a tárgyhoz kiadott jegyzetekben találjátok, és a tárgy oktatói mondanak. A hibák javítását e-mailben várom. Jó tanulást, meg ilyenek!

1. Alapfogalmak, egyszerű operációs rendszerek

a) Ismertesse a hard- illetve soft-realtime rendszerek fogalmát!

Valósidejű rendszer definíciója: Azokat a rendszereket, amelyekkel szemben a környezeti, valós időskálához kötött követelményeket támasztjuk valósidejű rendszernek nevezzük, VAGY az olyan rendszereket, melyek determinisztikus válaszidővel rendelkeznek
Soft-realtime: a kritikus taszkok prioritással futnak,
Hard-realtime: a kritikus taszkok garantáltan befejeződnek a megadott idő alatt.
Lényegében az a különbség, hogy egy soft-realtime rendszernél a válaszidőt nem tudjuk garantálni, csak azt tudjuk, hogy nagyon gyorsak vagyunk, míg hard-realtime esetben a válaszidő garantált. Más oldalról megközelítve: soft-realtime rendszerekben ha egy taszk megkapta a processzort, akkor nem vesszük el tőle (neki kell lemondani, vagy elkezdeni valamire várakozni), míg hard-realtime rendszerekben „erőszakkal elvesszük” a processzort a folyamattól, ha egy real-time prioritású szálnak szüksége van rá.

b) Sorolja fel az elosztott rendszerekkel szemben támasztott követelményeket!

Erőforrás megosztás (memória, CPU, adatok, ilyenek), nyitottság (bővíthetőség), konkurencia (a megosztás miatt kezelni kell), skálázhatóság (ha nagyobb rendszert szeretnénk jó, ha az architektúrális módosítás nélkül megoldható), hibatűrés (bírja ki, ha valamelyik komponense tönkremegy – mondjuk a targoncás véletlenül nekitolat), átlátszóság (a felhasználónak ne kelljen azzal törődnie, hogy a rendszeren belül jelentős kommunikáció van), biztonság (mint standalone rendszereknél)

c) Ismertesse a Cristian-féle óraszinkronizációs algoritmust!

Egyszerű, de nagyszerű. Legyen P a szinkronizálandó processz, S pedig az idő szerver. P elküldi a szinkronizáció kérést S-nek. S feldolgozza a késést, a válaszában a lehető legkésőbb helyezi el az aktuális időt (T). P fogadja a választ, és a saját óráját beállítja T+RTT/2-re, ahol RTT a kérés elküldése és a válasz megérkezése között eltelt idő. Tehát P feltételezi, hogy a kommunikáció időigénye mindkét irányban ugyanakkora, ezért kell az RTT/2-t hozzáadni. A hiba csökkenthető, ha a kérést többször elküldve „azt választjuk” a válaszok közül, amelyiknél RTT minimális volt (hiszen, ha RTT kicsi, akkor várhatóan a kommunikáció asszimmetriája is kisebb, vagy legalábbis ebben bízunk).

d) Ismertesse a µCOS-II operációs rendszer felépítését és legfontosabb szolgáltatásait!

?

2. Linux felhasználói ismeretek

Milyen állományrendszer jogosultságokat ismer? Hogyan értelmezzük a jogokat? Hogyan állíthatjuk be a jogosultságokat?

Minden fájlhoz és mappához tartozik egy tulajdonos (egyetlen felhasználó), és egy csoport (felhasználók halmaza). Ezeket a `chown` és `chgrp` parancsokkal állíthatjuk be. Miután ezt megtettük, beállíthatjuk, hogy milyen jogai legyenek a tulajdonosnak, a csoportnak és a töb-

bi felhasználónak (tehát aki nem tulajdonos és nincs benne a csoportban). A jog lehet olvasás, írás, végrehajtás (mappa esetén belépés a mappába). A jogosultság beállítására a `chmod` parancs használható.

Ezen kívül beállítható még fájlokra `setuid` és `setgid` mód. A lényegük ugyanaz, csak az egyik felhasználóra, a másik csoportra vonatkozik. Ha beállítjuk a `setuid` bitet, akkor a fájlt bárki futtatja az olyan jogosultságokkal fog futni, mintha a tulajdonosa futtatta volna. A `setgid` hasonló csak csoporttal. (varázslások `chmod`dal: vir-ea3-12.pdf 26. oldal aljától)

3. Linux rendszer készítése

Milyen szoftver csomagokra van szükség egy Linux rendszer összeállításához, és melyiknek mi a feladata? Elméletben hogyan hozhatunk létre egy Linux rendszert x86-os gépen nem x86-os architektúrára? Milyen lépéseket kell végrehajtanunk?

Reméljük erre gondolt Gábor:

Kell egy Linux kernel, ami a kernel. Kell egy `libc` (`glibc` vagy `uClibc`), ami a C-ben megszokott library-ket tartalmazza (hogy legyen `printf()`, meg ilyenek), kell valamilyen `bootloader`, kell `busybox` (hogy legyenek alapvető programok, mint `cp`, `ls`, stb), opcionálisan `e2fsprogs` (`ext` fájlrendszerek piszkálásához, mint particionálás, hibakeresés, a fájlrendszer kezelését a kernel végzi), `Dropbear` (ha akarsz SSH szervert).

Ha a fejlesztői gép és a célgép architektúrája nem egyezik meg, akkor szükséges egy keresztfordító környezet elkészítése. Ehhez a következő lépéseken kell végigmennünk:

Szükségünk van egy, a célrendszerre konfigurált Linux forrásra, vagy legalábbis a header állományokra mindenképpen.

1. Le kell fordítanunk a `gcc` csomagot (lefordítjuk a fordítót). Ehhez szükségünk van egy lefordított `glibc` csomagra, amit viszont nem tudunk lefordítani, mert most fordítjuk a fordítót. Így `glibc` hiányában a fordítás nem lesz teljesen sikeres, de kapunk egy olyan fordítót, amivel `glibc` már lefordítható.
2. Lefordítjuk `glibc`-t.
3. Most, hogy van `glibc` tudunk fordítani teljes értékű `gcc`-t.
4. Így van keresztfordítónk, és bármit le tudunk fordítani a célrendszerre.

4. Linux alkalmazás fejlesztés

Egy TCP szerver létrehozásához milyen lépéseket kell végrehajtanunk? Mindegyik lépésről írjon egy pár mondatos ismertetést!

Partizánkodás a Berkley socket API-val.

1. Először létre kell hoznunk egy socketet. Ehhez meg kell adnunk a protokollcsaládot, a típust (sorrendtartó/nem sorrendtartó, kapcsolat alapú vagy sem).
2. Ez után történik a socket címhez kötése (IP, port). Erre van egy `sockaddr` struktúra, amivel jól meg lehet csinálni. Oda kell figyelni, hogy a hálózati bájtsorrend nem biztos, hogy olyan (inkább biztos, hogy nem olyan) mint amit a számítógépünk használ.
3. A `listen()` rendszerhívással állíthatjuk be, hogy a socket fogadni tudjon kapcsolódásokat, amelyeket az `accept()` hívással kell elfogadni. Ha az `accept()` (ami egyébként blokkolódó, de van non-blocking változata is) visszatért, akkor a kapcsolat létrejött, kezdhetünk beszélgetni.
4. A beszélgetés a `send()` és `recv()` függvényekkel történik.
5. Ha megbeszéltük akkor a végén `close()` hívás mindkét oldalon.

5. Linux kernel fejlesztés

Mi a különbség a kernel modulok és az alkalmazások fejlesztése között?

a) **Programozási nyelvek:** csak C-ben és assemblyben. Ha nagyon akarsz lehet C++-ban, de inkább ne erőltessük.

b) **Használható függvények:** nem használhatunk olyan megszokott függvényeket, mint pl. `printf()`. Azért, mert pl. a `printf() libc`-ben van implementálva, amivel a kernel modulok nem linkelődnek össze. Szerencsére a `legtöbb` függvénynek van „k”-s megfelelője, ami használható kernelben, pl: `printk()`, `kmalloc()`.

c) **Header állományok:** mivel nem használhatjuk a fejlesztői könyvtárakat, ezért a hagyományos fejállományokat sem (pl. stdio.h), csak a kernelhez kapcsolódóakat (ezek a linux/ és asm/ könyvtárban vannak általában)

d) **Memória használhat:** A kernelmodul adatait a fizikai memóriában tároljuk, azaz nem kerülnek ki lapcserével a háttértárra. Így lehetőségek szerint ne pazaroljuk a memóriát (bár ez tanács felhasználói programok esetén is megszívlelendő). Ha nagyon sok adatunk van, akkor lehetőség van virtuális memóriát allokálni.

e) **Konkurencia problémák:** Nem tudom mire gondolt Gábor, talán arra, hogy ha itt csinálsz egy deadlockot, akkor egy processzort elbuktál.

f) **Jogosultságok:** Monolitikus kernel → mindenkinek van joga mindenhez, nincsenek belső védelmi szintek.

g) **Kód struktúra:** Nem tudom, hogy itt mire gondolt Gábor

h) **Paraméterezés:** Lehetőségünk van paramétereket átadni a modul számára. A modulon belül a `module_param` makróval vehetjük át ezeket. Átadni az `insmod` híváskor tudjuk.