11. tétel első része:

„A folyamatok közti kommunikáció eszközei (file, szignál, socket, csővezeték)”

A **folyamat** (vagy más néven processz) egy végrehajtható program elindításával előálló tevékenységsorozat a számítógépen, vagyis a programnak egy végrehajtási példánya. A végrehajtás több szálon is megvalósulhat. A folyamatoknak különböző állapotaik lehetnek (létrehozott, futásra kész, futó, várakozó, halott stb.). A mai multiprogramot operációs rendszerekben egyszerre száznál is több folyamat lehet jelen. Mindegyik egy egyedi azonosítóval rendelkezik, az a PID. Ezeknek a processzeknek a folyamatosnak látszó működését az ütemező (scheduler) segíti. A folyamatok kizárólagosan használnak erőforrásokat (memória, fájok, eszközök stb.). Ennek megvalósításához kölcsönös kizárás és a folyamatok szinkronizációja szükséges. A folyamatoknak gyakran szükségük van egymás közti információcserére is.

A fenti okok miatt a **folyamatok közötti kommunikáció** (inter-process communication, IPC) nélkülözhetetlen. A kommunikációnak különböző lehetőségei vannak. A legtöbb modern operációs rendszer támogatja az alábbiakat:

- fájl
- szignál
- socket
- csővezeték (pipe)
- osztott memória
- szemafor
- stb.

Fájl (állomány)

A folyamatok hozzáférhetnek a háttértáron tárolt fájlrendszer elemeihez. A kommunikáció egyik megközelítési módja szerint az egyik folyamat képes írni egy fájl tartalmát, a másik ezután képes olvasni a tartalmat, ezáltal megvalósítva az információcserét. Ez indirekt, kétirányú, aszinkron kommunikációt biztosít nagyméretű üzeneteket lehetővé téve. A fájlok kezelésének két eltérő megközelítése van: a **szöveges** (vagy formázott) és **bináris** (alacsony szintű) fájlkezelés. Előbbi esetben az állományban tárolt karaktersorozat beolvasás során különböző típusú értékekké alakulnak át. (Például a fájl részét képző `"-12345.67890"` Latin-2 (ISO 8859-2) kódolású 12 bájt hosszú sztring átalakítás után egy dupla pontosságú lebegőpontos ábrázolású (IEEE 754/1985) 8 bájt méretű változóba kerülhet eltárolásra.) Bináris fájlok esetén ugyanaz a bitsorozat reprezentálja az értéket a fájlban és a memóriában is. (Például a `"Imre"` sztring a `1701997897` négy bájton little-endian módon tárolt előjeles fix pontos ábrázolású egész számot is jelentheti.)

Függetlenül attól, hogy formázott- vagy alacsonyszintű fájlkezelésről van szó a következő alapvető műveletek végezhetőek el:

- megnyitás
- írás
- olvasás
- pozíció állítás
- bezárás

Használat előtt a fájlokat meg kell nyitni. A **megnyitás** során dől el, hogy melyik állománnyal fogunk dolgozni, azt milyen módon szeretnénk használni (csak írás, csak olvasás, írás és olvasás, hozzáfűzés). A megnyitás során egy fájlleíró jön létre, a későbbiekben ezzel tudunk hivatkozni a logikai fájlra. Szöveges fájlok esetén az `fopen`, bináris fájlok esetén az `open` rendszerhívás használható C nyelvű programokban.

Írás során értékeket tárolhatunk a fájlba, **olvasás** során pedig az állományban tárolt adat bekerül a memóriába. Szöveges fájlok írása/olvasása esetén például az `fprintf/fscanf` függvények, bináris fájlok esetén az `write/read` rendszerhívások használhatóak.

A fájlokhoz mindig tartozik egy pozíció (egyfajta képzeletbeli kurzor), ami azt a helyet jelenti, ahol a következő írási/olvasási művelet elkezdi majd kifejteni a hatását. A pozíció alapértelmezett értékét a megnyitás módja határozza meg, majd minden írási/olvasási művelet során automatikusan változik az írt/olvasott bájtok számának megfelelően. Emellett a pozíció manuális megváltoztatására is van lehetőség az `fseek/lseek` függvények segítségével.

A kívánt tevékenységek elvégzése után a megnyitott fájlokat be kell zárni. Szöveges fájlok esetén az `fclose` függvény segítségével, bináris állomány esetén a `close` rendszerhívással. Ekkor mentődnek el a módosítások, megváltoznak az állomány attribútumai, a fájltartalom hozzáférhetővé válnak más programok számára.

Szignál

Alapvetően a folyamatok szinkronizálására szolgál, amit egyfajta direkt, egyirányú kommunikációként is értelmezhetünk. A szignál egy (információtartalom nélküli) jelzés, amit egy processz küldhet egy másiknak. Többféle szignál létezik különböző események bekövetkeztének jelzésére. Például: `SIGKILL`, `SIGTERM`, `SIGINT`, `SIGALRM`, `SIGCHLD`, `SIGSTOP`, `SIGCONT`, `SIGUSR1`, `SIGPIPE`, stb. Ezek két kategóriába sorolhatóak, vannak **elkapható** és **nem elkapható** szignálok. Utóbbiaknak érkezése esetén egy jól definiált tevékenység fog végrehajtódni. Előbbiek esetén fel lehet készüli a szignál érkezésére, a programozó meghatározhatja, hogy mi legyen a végrehajtandó tevékenység abban az esetben, ha majd egyszer megérkezik az adott szignál (ha egyáltalán érkezik). A végrehajtandó tevékenység egy **szignálkezelő** eljárás formájában definiálandó. A szignál és a hozzá tartozó szignálkezelő összerendelésére a `signal` függvényt használjuk C nyelv esetén. Azután, hogy megtörtént az imént említett összerendelés, a program végrehajtása a szokásos módon folytatódik, viszont később, ha tetszőleges időpontban beérkezik az adott szignál, akkor a normál végrehajtás felfüggesztésre kerül, egy automatikus rejtett hívás révén lefut a szignálkezelő eljárás, majd folytatódik a program végrehajtása a következő utasítással. (A logika hasonló, mint egy hardveres megszakítás kezelése esetén.)

Egy folyamat küldeni is tud szignált egy másik processznek. Erre szolgál a `kill` függvény C-ben. Segítségével tetszőleges szignált tudunk küldeni egy adott PID értékkel rendelkező folyamat számára. Amennyiben a fogadó folyamat fel van készülve az adott szignál érkezésére, akkor ezzel megvalósul a szinkronizáció, azaz egy folyamat tudhatja a másikról, ahogy hol „tart" a végrehajtással. A `pause` függvény segítségével egy folyamat várakozó állapotba kerül mindaddig, amíg egy szignált nem kap. Ezután lefut az adott szignálkezelő majd folytatódik a program normál végrehajtása, így egy folyamat képes „bevárni" a másikat.

Socket

A socket (aljzat) két folyamat közötti **TCP/IP alapú kommunikáció** végpontja. Ezek a végpontok tartozhatnak azonos számítógépen lévő vagy különböző számítógépen lévő két folyamathoz is. Több típusa van a socket-eknek. Az egyik leggyakrabban alkalmazott „datagram" típusú, kapcsolat nélküli kommunikáció megvalósítására képes UDP szállítási réteg protokoll segítségével, a másik a „stream" típusú, amely kapcsolatorientált az alkalmazott TCP protokoll miatt. A kliens-szerver architektúrájú kommunikáció menete a két típus esetén eltérő, bár vannak mindenképpen szükséges lépések. A kommunikáció előtt az aljzatot létre kell hozni (`socket`) a végén pedig be kell zárni (`close`), közben tudunk üzenetet küldeni (`send, sendto, write`) és fogadni (`recv, recvfrom, read`). A szerver oldalon a socket-et mindig hozzá kell rendelni egy hálózati címhez (IP cím és port szám). Az adatfolyam (stream) jellegű kommunikáció esetén az aljzatok létrehozása után, de még a tényleges üzenetküldés előtt ki kell építeni egy kapcsolatot a szerver oldalon a `listen` és az `accept` függvények segítségével, kliens oldalon pedig a `connect` függvény megfelelő idejű meghívásával (C nyelv esetén). Az említett függvények számos speciális adatszerkezetet és típust használnak (például: `in_addr, sockaddr_in, sockaddr`, stb.).

Csővezeték (pipe)

FIFO jellegű, egyirányú, aszinkron kommunikációt tesz lehetővé azáltal, hogy a két folyamat között egy képzeletbeli csővezetéket valósít meg. Amit az egyik folyamat beír, az eljut a másikhoz, aki sorban kiolvashatja azt. A csővezeték lehet név nélküli vagy névvel rendelkező is.

Egy egyszerű speciális eset az, amikor egy folyamat alapértelmezett kimenetét parancssori környezetben átirányítjuk egy másik folyamat alapértelmezett bemenetére, egyirányú kommunikációt megvalósítva a `'|'` (függőleges vonal, pipe) karakterrel jelölt operátor segítségével. Például Linux rendszerben a „`less tmp.txt`" parancs az adott szöveges fájl tartalmát jeleníti meg a képernyőn (standard output), míg a „`wc -c`" parancs a begépelt szöveg (standard input) karaktereinek számát határozza meg. Ebben az esetben az összetett „`less tmp.txt | wc -c`" parancs a csővezeték révén a szöveges fájlban tárolt karakterek számát határozza meg, mivel a fájltartalom nem a kimeneten jelenik meg hanem bemenetként szolgál a másik folyamat számára.

Egy C nyelvű programban a `pipe` függvénnyel tudunk csővezetéket létrehozni. Ennek paramétere egy kételemű, egészeket tartalmazó tömb címe, ahol a hívás után két alacsony szintű fájlleírót találunk. A tömb második eleme egy beépített írható fájlt fog azonosítani, míg az első eleme egy olvasható fájlt, amelyek a `write` és `read` rendszerhívásokkal kezelhetőek.

Fork művelet után az egyik folyamat az első írhatja, a másik pedig olvashatja a cső két végét jelentő adatfolyamot, ezzel megvalósítva a folyamatok közötti kommunikációt.

Megjegyzés:

A tételhez kapcsolódó programozói eszközök használatának képessége szükséges lehet. A témakörrel kapcsolatos tágabb ismeretség nem hátrány.

Kapcsolódó tantárgyak:

- *Rendszerközeli programozás*
- *Számítógép architektúrák*
- *Operációs rendszerek*

Ajánlott irodalom:

- Niel Matthew, Richard Stones: *Beginning Linux Programming* (Wiley, 2004)
  3., 11., 13. és 15 fejezet.