A folyamatszálakról

A több szál fogalma először az időszeletes rendszereknél (*time sharing systems*) jelent meg, ahol egyszerre több személy is bejelentkezhetett egy központi számítógépre. Fontos volt a processzor idejének igazságos megosztása (kiosztása) a felhasználók közt. Így jött létre a *folyamat* és a *folyamatszál*.

Egy *folyamat* (*process*), *munka* (*job*) vagy *feladat* (*task*) olyan számítás (műveletek meghatározott sorrendben történő szekvenciális végrehajtása), amelyet konkurrensen, párhuzamosan hajthatunk végre más számításokkal. A folyamat a processzor aktivitásának absztrahálása, vagyis egy program futó példánya (egy végrehajtás alatt álló program – a végrehajtás megkezdődött, de még nem fejeződött be).

A folyamatokat és a párhuzamos végrehajtást leginkább úgy tudjuk szemléltetni, hogy minden egyes folyamathoz tartozik egy logikai processzor és egy logikai memória. A memória tárolja a programkódot, a konstansokat és a változókat, a programot a processzor hajtja végre. A programkódban szereplő utasítások és a végrehajtó processzor utasításkészlete megfelelnek egymásnak. Az operációs rendszer feladata, hogy a fizikai eszközökön (fizikai processzor, fizikai memória) egymástól elkülönítetten, védetten létrehozza és működtesse a folyamatoknak megfelelő logikai processzorokat és memóriákat – ezeket megfeleltesse a fizikai processzornak, fizikai memóriának, mintegy kiossza a fizikai processzort a logikai processzoroknak (a folyamatok versengenek a CPU-ért).

A folyamathoz kötődő további általános fogalom a *szál* (*thread*) fogalma. A szál egy folyamaton belüli végrehajtási egység, amely egy környezetből és saját utasítássorozatból áll. A folyamatszálakat a folyamaton belül lehet párhuzamosan végrehajtani. A szál tehát a kód-végrehajtás legkisebb, önállóan ütemezett egysége. Egy folyamatnak tetszőleges számú szálja lehet, de legalább egy mindig van.

Az operációs rendszerek és programozási nyelvek tervezői szükségét érezték annak, hogy egy folyamaton belül egymástól független számítási egységeket határozzanak meg. A szálakat akkor fejlesztették ki, amikor nyilvánvalóvá vált, hogy nem az időveszteségesen végrehajtható alkalmazásokra van szükség (pl. felhasználóra várni, hogy beavatkozzon), hanem olyan alkalmazásokra, amelyek lehetőleg egyszerre több művelethalmazt is végrehajtanak egyidejűleg, párhuzamosan (pl. háttérben futó helyesírás ellenőrzés, vagy érkező hálózati üzenetek feldolgozása), de ezek a műveletek ne különálló folyamatokba legyenek szervezve, hanem egy folyamaton belül jelenjenek meg. A különálló folyamatok létrehozása és az egymással való kommunikáció megvalósítása nagy többletmunkát jelentett ezen alkalmazások esetén.

A szálak tehát párhuzamos végrehajtású, közös memóriát használó programrészek a folyamatokon belül (egy program végrehajtása több szálon futhat). A szálaknak saját logikai processzoruk van (a CPU-ért ugyanúgy versenyeznek mint a folyamatok), azonban memóriáik nincsenek védetten elkülönítve, közös logikai memóriát használnak, csak a kódon és változókon osztoznak. A folyamat adatzónáját és környezetét minden szál közösen használja. Az egyetlen memóriaterület, amelyet egy szál elfoglal, az a hozzárendelt verem. Minden folyamatszál saját környezettel rendelkezik.

A fentiek miatt az operációs rendszer lényegesen gyorsabban tud végrehajtani egy átkapcsolást a szálak között, mint a folyamatok között. Ez a szálak alkalmazásának gyakorlati jelentősége.

A szál és a folyamat megkülönböztetésére használatos a *pehelysúlyú* (*lightweight*) és *nehézsúlyú* (*heavyweight*) folyamat elnevezés is.

Folyamatszálak általános jellemzői

A folyamatszálak jellemzőit az alábbiakban foglalhatjuk össze:

- A folyamatszálakat párhuzamosan futó végrehajtási egységekből álló programok írására használjuk.
- A folyamatszálak utasítás sorozatokat hajtanak végre, amelyek egybezártnak tekinthetők.
- A folyamatszál végrehajtását bármikor meg lehet szakítani, így az átadja a vezérlést egy másik szálnak.
- A folyamatszálakkal való műveleteket rendszerhívások (Windows NT, Sun Solaris operációs rendszerek esetében) vagy a programozási könyvtárak (C, C++) segítik.

Ha egy folyamatszál lehetséges állapotait vizsgáljuk, ez már a különböző implementációk szintjén másként valósul meg.

*Borland Delphi*ben egy szál lehet:

- *Aktív* vagy továbbindított (resumed)
- Felfüggesztett (suspended)

C/C++, *Java* programozási nyelvekben négy lehetséges állapota van egy szálnak:

- *Aktív* (fut a szál): a folyamat erőforrásait ellenőrzés alatt tarthatja. Más szál nem lehet aktív ugyanabban az időben, hacsak nincs több rendelkezésünkre álló processzor. Egy szál három okból állhat le:
  - A szál átadja a vezérlést egy másik szálnak.
  - A száltól az ütemező erőszakosan veszi el a vezérlést.
  - Az életciklusa végére ér (meghal).
- *Futtatható:* ha futtatásra készen áll. Az ilyen állapotban levő szál nem vár I/O-ra, csak arra, hogy az operációs rendszer futásra ütemezze.
- *Nem futtatható:* a szál nem kész a végrehajtásra. Ez akkor következhet be, ha:
  - *Blokkolva van:* valamilyen erőforrástól információt vár (pl. CPU, I/O, billentyűzet, szinkronizációs objektum). Ha megkapja a várt információt, akkor futtatható állapotba kerül.
  - *Alszik:* a szálunk leáll a végrehajtással, hogy a többi szál is esélyt kapjon a futásra. Ez nagyon fontos segítőkész szálak (*cooperative threads*) esetében, azért, hogy biztosítsák más szálaknak is a futásra való esélyt.
- *Halott:* Egy szálat akkor tekintünk halottnak, ha megszűnik létezni a fent említett állapotok egyikében. Egy szál meghalhat, ha eléri kódjának végét, vagy ha megöli egy eljáráshívás az őt tartalmazó folyamatból vagy egy másik szálból a folyamaton belül.

A szálak másik jellegzetes tulajdonsága a *prioritás*. A prioritás a szál fontosságára vonatkozik a többi futtatható állapotban levő szállal szemben. Amikor több szál is futásra

kész állapotban van, akkor a szálütemezőnek döntenie kell, hogy melyik szálat futtassa. Az ütemező a legnagyobb prioritású szálat választja. Ha éppen fut egy szál, akkor egy magasabb prioritású szál az aktív szál felfüggesztését vonja maga után, így megengedheti magának, hogy addig fusson amíg át nem akarja engedni a vezérlést egy másik szálnak, vagy egy magasabb prioritású szál meg nem szakítja futását.

A szálak nem módosítják a programunk szemantikáját, csak a műveletek időzítését változtatják meg. Így összefüggő problémák megoldására elegáns módszert nyújtanak.

Különösen előnyös folyamatszálakat használni az alábbi esetekben:

- *Lassú feldolgozásnál:* ha egy alkalmazás hosszasan számol, számít, nem tud üzeneteket fogadni, tehát nem tudja az eredmény kiírását sem frissíteni.
- *Háttérben folyó feldolgozás:* egyes feladatok nem időkritikusak, de folyamatosan kell futniuk (pl. a nyomtatás végrehajtása egy szövegszerkesztőben tipikusan másik szál feladata, hisz ez időt vesz igénybe, és a felhasználó szeretné munkáját folytatni a nyomtatás elindítása után, nem feltétlenül várja ki annak befejezését).
- *I/O műveletek:* az I/O műveleteknek előreláthatatlan késése lehet (pl. állományok másolása a háttérben).
- *Kiszolgálás:* Kliens-szerver architektúrák esetében a szerver minden egyes kliensre létrehoz egy folyamatszálat és párhuzamosan szolgálja ki őket.

A folyamatszálak használatának előnyei

A többprocesszoros rendszerek kihasználása: egy egyszerű egyszálas alkalmazás nem tud kihasználni két vagy több processzort. Több processzoros gépek esetén az operációs rendszer feladata az, hogy a processzorokat rejtetten ossza ki, anélkül, hogy a programozó, felhasználó tudná, hogy most pont melyik processzorban fut a kód. Így a szálak használatában (programozáskor) sem kell különbséget tenni az egy- és a többprocesszoros gépek között.

- Hatékony időmegosztás: szálakkal és folyamat-prioritásokkal biztosíthatjuk, hogy mindenki igazságos CPU időkiosztásban részesüljön.
- Hatékony erőforrás-kihasználás: a rendszerben a processzoron kívül lehetnek más erőforrások is, amelyeket hatékonyan meg lehet osztani a folyamatszálak között (pl. egy folyamat nem használja ki az adott erőforrást – lehetőleg minden erőforrás minden pillanatban maximálisan ki kell legyen használva).
- A feladat-végrehajtás gyorsítása: ha egy feladatot párhuzamosan végrehajtható részfeladatokra tudunk bontani, és ezeket párhuzamosan végre tudjuk hajtani (pl. valódi párhuzamossággal), akkor jelentős gyorsítást érhetünk el.
- Többféle feladat egyidejű végrehajtása: A számítógépet egyidejűleg többféle célra tudjuk felhasználni (pl. számítás közben levélírás, képnézés).

Az 1. ábrán a folyamatszálak lehetséges állapotai láthatók.

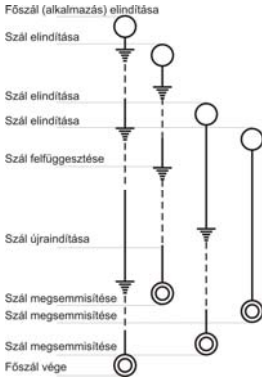
A 2. ábrán egy egyszerű alkalmazást mutatunk be, amely egy szálon fut (*főszál*, *elsődleges szál*, *main thread*). A 3. ábrán egy többszálú alkalmazás futását mutatjuk be. Az idő fentről lefelé telik.

1. ábra
Folyamatszálak lehetséges állapotai

2. ábra
Egy egyszerű egyszálas alkalmazás

Megfigyelhetjük, hogy:

- Az alkalmazásban a főszál nem fut állandóan. Lehetnek hosszú időintervallumok, amikor nem kap üzenetet, nem végez számítást. Az alkalmazás által lefoglalt memória és más erőforrások megvannak, az ablak is a képernyőn van, de a kódból nem hajt végre egyetlen részletet sem.
- Az alkalmazást elindítottuk, és a fő szál fut. Amikor az alkalmazás ablaka létrejött, nincs semmilyen más dolga, csak üzenetekre vár. Ha nincs új üzenet amit fel kell dolgozni, akkor az operációs rendszer felfüggeszti a szálat.
- Amikor a felhasználó a menüből kiválasztotta a parancsot, az operációs rendszer újra aktívvá teszi a szálat, megjelenik az adatbeolvasó dialógusdoboz. A főszál most újra aktív.



3. ábra
Egy egyszerű háromszálas alkalmazás

- Ez a felfüggesztés-továbbindítás többször is megismétlődik, amíg ki nem lépünk az alkalmazásból. A főszál nem használja ki hatékonyan az erőforrásokat, sok a szünet, az az időintervallum, amikor az alkalmazás nem csinál semmit.

Műveletek folyamatszálakkal

Folyamatszálakkal a következő műveleteket lehet elvégezni:

- Folyamatszál elindítása
- Folyamatszál felfüggesztése
- Folyamatszál újraindítása
- Folyamatszálak közötti kommunikáció, szinkronizálás
- Folyamatszál megsemmisítése

Folyamatszálak elindítása

Folyamatszálak elindítására a különböző programozási nyelvek függvényeket biztosítanak, az objektumorientált nyelvekben a folyamatszálakat osztályok valósítják meg, az elindítást pedig a konstruktor.

Gyakran bizonyos beállításokat szeretnénk eszközölni a szálon, mielőtt elindítanánk. Létrehozáskor megadhatjuk, hogy a szál létrehozása után aktív legyen vagy felfüggesztett. Ezeket a beállításokat a függvények vagy a konstruktor paramétereivel érhetjük el. Ha felfüggesztett állapotban hozzuk létre a szálat, akkor a főszál adatokat állíthat be, biztosítva, hogy amikor a szálat elindítjuk, fogja látni a módosításokat, az adatok frissítve lesznek.

Folyamatszál felfüggesztése

Folyamatszálak megszüntetésére a programozási környezet függvényeket biztosít. Felfüggesztés után a folyamatszál nem lesz aktív. Minden általa lefoglalt erőforrás megmarad, de a szál már nem dolgozik.

Folyamatszál újraindítása

Folyamatszálak újraindítására a programozási környezet függvényeket biztosít. Egy felfüggesztett szálat újra lehet indítani (aktívvá tenni). Ekkor újrakezdi működését, dolgozni kezd.

Folyamatszálak közötti kommunikáció, szinkronizálás

A folyamatszálak egymáshoz való viszonyukat, a köztük lévő kommunikációt tekintve lehetnek:

- független
- versengők
- együttműködők

A *független folyamatszálak* egymás működését nem befolyásolják. Végrehajtásuk teljes mértékben *aszinkron*, nem függnek egymástól, egymással teljesen párhuzamosan is futhatnak.

A *versengő folyamatszálak* nem ismerik egymást, de közös erőforrásokon kell osztozniuk. A korrekt és biztonságos erőforrás-kezelés *szinkronizálást* igényel (pl. ha egy szál nyomtatni akar, és a nyomtató foglalt, egy másik szál nyomtat, akkor a második meg kell hogy várja, amíg az első szál, amely lefoglalta a nyomtatót, befejezi a nyomtatást).

Az *együttműködő folyamatszálak* ismerik egymást, együtt dolgoznak egy feladat megoldásán, információt cserélnek – kommunikálnak egymással. A kooperatív viselkedést a programozó határozta meg, a feladatot tervszerűen bontottuk egymással kommunikáló folyamatszálakra. Az együttműködést, a kommunikációt adat- vagy információcsere útján tudják megvalósítani a szálak. A cserélt információ esetenként egyetlen bitnyi is lehet (*flag*), máskor akár több megabájt is lehet. A szálak közötti információcserének két alapvető módja alakult ki: *közös memórián* keresztül, illetve *üzenetküldéssel / fogadással.*

Közös memória

Közös memórián keresztül történő adatcsere esetén az együttműködő szálak mindegyike a saját címtartományában lát egy közös memóriát (globális rész az alkalmazás címtartományában). Ezt a közös memóriát egyidejűleg több szál is írhatja, illetve olvashatja, a *PRAM* (*Pipelined Random Access Memory*) modell szerint.

Az *olvas* és *ír* műveletek egyidejű végrehajtására a következő szabályok vonatkoznak:

- *olvasás-olvasás* ütközésekor mindkét olvasás ugyanazt az eredményt adja, és ez megegyezik a memória tartalmával;
- *olvasás-írás* ütközésekor a memória tartalmát felülírja a beírt adat, az olvasás eredménye a memória régi, vagy az új tartalma lesz, annak függvényeben, hogy melyik történt hamarabb, az írás vagy az olvasás;
- *írás-írás* ütközésekor valamelyik művelet hatása érvényesül, az utoljára beírt érték felülírja a memória tartalmát.

Az egyidejű műveletek nem interferálhatnak, nem lehet közöttük zavaró kölcsönhatás, harmadik érték sem olvasáskor, sem íráskor nem alakulhat ki. Az írás és olvasás műveletek a PRAM modell szintjén atomiak, tovább nem oszthatók. A *pipelined* elnevezés azt tükrözi, hogy a memóriához egy sorosítást végző csővezetéken jutnak el a parancsok.

Kommunikáció üzenetekkel

A szálak nem használnak közös memóriát. Rendelkezésünkre áll két művelet: a *Küld* (*Send*) és a *Fogad* (*Receive*).

A *Küld(<adat>, <szál>)* művelet végrehajtásakor a műveletet végrehajtó szál elküldi a megadott adatot a megadott folyamatszálnak, a *Fogad(<adat>, <szál>)* művelet pedig a megadott száltól érkező adatot tárolja.

Kovács Lehel

Szórakoztató kémia

A kémia az utóbbi időben a köztudatban nagyon leértékelődött, annak ellenére, hogy az emberi lét számára mondhatni a legáltalánosabban alkalmazott tudomány. A kémia állandó fejlődése biztosítja az élettudományok, mezőgazdaság, energetika, ipari technika, információrögzítés és közlés, a telekommunikáció, kulturális értékek mentése és megőrzése stb. feltételeit. E nagyon sokrétű nemes feladat mellett a mindennapi ember a kémia rovására tudja be az életét megkeserítő dolgokat, mint a mérgezések, harci anyagok pusztításai, ártalmas mesterséges táplálékok és élelmiszeradalékok, az anyagi károkat okozó korrózió, rothadási folyamatok és még annyi más nem kívánt jelenség, ami mind kémiai változások eredménye. Talán az a legvonzóbb, hogy ezek hatásának csökkentését, hasznos folyamatokkal való helyettesítését is a kémikusok tudják megoldani. Alkotó munkájukban a fizika, matematika vívmányait alkalmazva mind több, az emberiség javát szolgáló eredményt érnek el.

A vegyészek eredményes munkájához alapos felkészültség mellett sok kitartásra, türelemre, ötletességre, s sokszor humorérzékre is szükség van.

Vegyészek humora gyakran az általuk előállított, vagy tanulmányozott anyagok megnevezésében is megnyilvánul. Lássunk erre egy pár klasszikus példát!

Sexitiofen a $C_{24}H_{14}S_6$ összetételű vegyület nevét nem kecses alakjáról, hanem a vázát alkotó hat tiofen egységről kapta.