Table of Contents

[Többmagos rendszerek programozása 1](#_Toc166791747)

[Szinkronizációs Mechanizmusok 1](#_Toc166791748)

[Feladatok Felosztása 2](#_Toc166791749)

[Tesztelés és Hibakeresés 2](#_Toc166791750)

[Eszközök és Könyvtárak 2](#_Toc166791751)

[Konkurrencia vs párhuzamosság 3](#_Toc166791752)

[Amdahl törvénye 4](#_Toc166791753)

[Többszálú modellek 4](#_Toc166791754)

[Implicit thread modellek 5](#_Toc166791755)

[Fork() és Exec() rendszerhívások 5](#_Toc166791756)

[Thread törlése 6](#_Toc166791757)

**4. Szálak**

# Többmagos rendszerek programozása

A többmagos rendszerek programozása lehetőséget biztosít arra, hogy a szoftverek kihasználják a modern processzorok párhuzamos feldolgozási képességét. A szálak használata kritikus szerepet játszik ebben, mivel lehetővé teszik a feladatok egyszerre történő végrehajtását több CPU magon. Itt van néhány alapelv és gyakorlat, amelyeket figyelembe kell venni a többmagos rendszerek programozásakor:

1. Szálak és Párhuzamosság

A szálak olyan könnyűsúlyú folyamatok, amelyek megosztják a szülő folyamat erőforrásait, mint például a memória és a fájlleírókat. Egy többmagos rendszerben a szálakat párhuzamosan ütemezhetik a különböző magokra, lehetővé téve az alkalmazások teljesítményének jelentős növelését:

Szálkezelés: A modern operációs rendszerek, mint Windows, Linux, vagy macOS, magas szintű szálkezelési API-kat kínálnak, mint például a POSIX szálak (pthreads) vagy a Windows Threads.

Automatizált Párhuzamosság: Néhány magas szintű programozási nyelv, mint a Java vagy a .NET keretrendszer, automatikus szálkezelési lehetőségeket kínál, amelyek egyszerűsítik a többmagos programozást.

# Szinkronizációs Mechanizmusok

A szálak közötti megfelelő szinkronizáció biztosítása nélkülüli párhuzamos végrehajtás során versenyhelyzetek (race conditions), éhenhalás (starvation), vagy holtpontok (deadlocks) léphetnek fel:

Zárak: Mutexek, szemináriumok és monitorok segítségével biztosítható, hogy a kritikus szekciókhoz egyidőben csak egy szál férjen hozzá.

Atomikus Műveletek: Modern programozási nyelvek támogatnak atomikus műveleteket, amelyek biztosítják, hogy bizonyos műveletek megszakíthatatlanul fejeződjenek be.

Feltételes Változók: Ezek lehetővé teszik a szálaknak, hogy egy adott feltétel teljesüléséig várakozzanak, ezzel csökkentve a CPU felesleges használatát.

# Feladatok Felosztása

A hatékony többmagos programozás kulcsa a feladatok megfelelő felosztása és a munkaterhelés egyenletes elosztása a magok között:

Adatpárhuzamosság: A probléma úgy oszlik meg, hogy az adatok különböző részeit különböző szálak dolgozzák fel párhuzamosan.

Feladatpárhuzamosság: Különböző feladatok vagy műveletek hajthatók végre egyszerre, mindegyik külön szálon.

# Tesztelés és Hibakeresés

A több szálas programok tesztelése és hibakeresése jelentősen bonyolultabb, mint a szekvenciális programoké. A hibák, mint a versenyhelyzetek, gyakran nem determinisztikusak, így a kód alapos ellenőrzése és a szálak közötti interakciók szimulációja kulcsfontosságú.

# Eszközök és Könyvtárak

A fejlesztők számos könyvtárat és eszközt használhatnak a többmagos programozás megkönnyítésére:

Intéző Könyvtárak: OpenMP, Intel Threading Building Blocks (TBB), vagy a C++11 felett elérhető beépített szálkezelési könyvtárak.

Profilozók és Hibakereső Eszközök: Valgrind, Intel VTune, GDB és mások segítenek azonosítani a teljesítmény gócpontjait és a szinkronizációs problémákat.

A többmagos rendszerek programozása lehetőséget biztosít a modern alkalmazások teljesítményének jelentős növelésére, de megköveteli a párhuzamos programozási modellek mélyebb megértését és a megfelelő eszközök alkalmazását.

# Konkurrencia vs párhuzamosság

A konkurrencia és párhuzamosság két különböző fogalom, amelyek gyakran összefüggenek, de mégis különböző jelentéssel bírnak a számítógépes architektúrákban és programozásban.

Konkurrencia azt jelenti, hogy egy rendszer képes több feladatot támogatni úgy, hogy a feladatok előrehaladása lehetséges legyen. Konkurrens rendszerekben a feladatok időben átfedhetik egymást, de nem feltétlenül futnak valós időben egyszerre. Egy egymagú processzor esetében a konkurrencia azt jelenti, hogy a processzor ideje megosztásra kerül a különböző szálak vagy folyamatok között, amelyek váltakozva futnak, létrehozva az egyszerre futás illúzióját .

Párhuzamosság azt jelenti, hogy ténylegesen több művelet hajtódik végre egyszerre. Ez általában többmagos processzorokon vagy több processzoros rendszerekben valósul meg, ahol a különböző szálak vagy folyamatok ténylegesen egyidejűleg futnak különböző processzor magokon. Ez lehetővé teszi a számítási feladatok gyorsabb elvégzését, mivel a munka valódi időben oszlik meg .

Az operációs rendszerek tervezése során fontos megkülönböztetni a konkurrenciát és a párhuzamosságot, mivel ezek befolyásolják a rendszer teljesítményét és hatékonyságát. A konkurrencia kezelésére szolgáló eszközök, mint például a szálfüggetlen blokkok (threads) és folyamatok (processes) ütemezése, lehetővé teszik a feladatok hatékony kezelését még egyetlen processzormag használata esetén is, míg a párhuzamos számítási modellek, mint például a többmagos feldolgozás, kihasználják a hardveres erőforrásokat a teljesítmény növelése érdekében.

# Amdahl törvénye

Az Amdahl törvénye egy matematikai képlet, amely az alkalmazások teljesítményjavulásának lehetséges mértékét írja le, amikor több számítási magot adunk egy olyan alkalmazáshoz, amelynek vannak soros (nem párhuzamos) és párhuzamos összetevői is. Az Amdahl törvény szerint, ha a szerializáltan végrehajtandó műveletek aránya, és a processzormagok száma, akkor a gyorsulás képlete:

Például, ha egy alkalmazás 75%-ban párhuzamosítható és 25%-ban soros, akkor két maggal a gyorsulás 1,6-szoros lehet. Ha még két magot adunk (összesen négy), akkor a gyorsulás 2,28-szorosára nő. Fontos megjegyezni, hogy ahogy a végtelenhez közelít, a gyorsulás felé konvergál. Tehát ha egy alkalmazás 40%-ban végezhető sorosan, akkor a maximális gyorsulás 2,5-szeres, függetlenül attól, hogy hány processzormagot adunk hozzá.

# Többszálú modellek

A többszálú modellek három fő típusát különböztethetjük meg: sok-egyhez, egy-egyhez és sok-sokhoz modell. Mindegyik modell más-más kapcsolatot alakít ki a felhasználói szálak és a kernel szálak között:

Sok-egyhez modell: Ebben a modellben sok felhasználói szintű szál egyetlen kernel szálhoz van hozzárendelve. Ez az elrendezés hatékony lehet a szálkezelésben, mivel a szálkezelést a felhasználói területen lévő szálkönyvtár végzi, ám hátránya, hogy ha egy szál blokkolódik, az az egész folyamatot blokkolhatja, és többmagos rendszereken nem képes kihasználni a párhuzamos futtatás előnyeit​​.

Egy-egyhez modell: Minden felhasználói szálhoz pontosan egy kernel szál tartozik. Ez a modell lehetővé teszi a szálak nagyobb fokú konkurrenciáját és párhuzamosságát, mivel több szál is futtatható párhuzamosan több processzoron. Hátránya, hogy a kernel szálak kezelése magasabb rendszererőforrás-igényű, ami korlátozhatja a létrehozható szálak számát​​.

Sok-sokhoz modell: Több felhasználói szál több kernel szálhoz van hozzárendelve, ami rugalmasságot biztosít, és lehetővé teszi, hogy a rendszer dinamikusan igazodjon a szálak és erőforrások optimális kihasználásához. Ez a modell a legalkalmasabb nagy teljesítményű és skálázható alkalmazások számára, mivel nem korlátozza feleslegesen a felhasználói szálak számát, és kihasználhatja a többmagos processzorok előnyeit​​.

Mindegyik modellnek megvannak a maga előnyei és hátrányai, és a választás gyakran az alkalmazás specifikus igényeitől és a rendelkezésre álló hardveres erőforrásoktól függ.

# Implicit thread modellek

Az implicit szálas modell olyan programtervezési paradigma, amelyben a szálak kezelését és létrehozását elrejtik az alkalmazásfejlesztők elől, és ezt a feladatot fordítóprogramokra és futásidejű könyvtárakra bízzák. Ennek a megközelítésnek három jellemző módszere van:

Szálmedence (Thread Pools): A szálmedence előre létrehozott szálak gyűjteményét jelenti, amelyek egy feladat elvégzésére várnak. A szálakat egyszer létrehozzák, majd újra és újra használják, ahelyett, hogy minden egyes feladat elvégzéséhez új szálat hoznának létre és semmisítenének meg. Ez jelentősen csökkentheti a szál létrehozásának költségeit és kezelheti az egyidejűleg futó szálak számát, amely fontos a rendszererőforrások hatékony kihasználása szempontjából .

OpenMP: Egy olyan készlet fordítóprogram-direktívákból és API-ból áll, amely támogatja a párhuzamos programozást megosztott memóriás környezetben. Az OpenMP lehetővé teszi, hogy az alkalmazásfejlesztők fordítási utasításokat helyezzenek el kódjukban, amelyek meghatározzák, hogy mely kódrészletek hajthatók végre párhuzamosan .

Grand Central Dispatch (GCD): Ez egy olyan technológia, amely kiterjesztéseket biztosít a C nyelvhez, és egy futásidejű könyvtárat, amely lehetővé teszi az alkalmazásfejlesztők számára, hogy meghatározzák a kódrészletek párhuzamos végrehajtását. A GCD kezeli a szálkezelés részleteit, és ütemezi a feladatblokkok végrehajtását, amelyeket futásidejű várólistákra helyeznek .

Ezek az implicit szálas modellek különösen hasznosak többmagos processzorokkal rendelkező rendszerekben, ahol a szálak hatékony kezelése és a processzorok teljesítményének maximalizálása kulcsfontosságú.

# Fork() és Exec() rendszerhívások

A UNIX operációs rendszerben a fork() és az exec() két alapvető rendszerhívás, amelyek a folyamatkezelés két lépésre bontásával működnek: új folyamat létrehozása és új program futtatása.

fork() rendszerhívás: Ez a hívás létrehoz egy új folyamatot, amely a szülő folyamat pontos másolata, beleértve az címteret (memória), futási kontextust, nyitott fájlok listáját és egyéb rendszererőforrásokat. A fork() után mind a szülő, mind az új gyermek folyamat ugyanott folytatja a végrehajtást, ahol a fork() meghívása történt. A visszatérési érték a szülőben a gyermek folyamat azonosítója (PID), míg a gyermekben nulla .

exec() rendszerhívás: Az exec() család hívások, mint például execl(), execp(), stb., egy meglévő folyamat memóriatartalmát egy új programmal helyettesítik. Ezt úgy teszik, hogy az új programot betöltik a folyamat címterébe, ezzel teljesen felülírva az előző futtatási kontextust. Az exec() sikeres hívása után a folyamat az új program kódját kezdi el futtatni, és az eredeti program állapota teljesen elveszik. Ha az exec() hívás sikertelen, akkor hibaüzenetet ad vissza, és a kontroll a hívó folyamathoz tér vissza .

Ezek a rendszerhívások lehetővé teszik a UNIX rendszerekben, hogy a folyamatok rugalmasan kezeljék az erőforrásokat és a programokat, megkönnyítve az operációs rendszer feladatainak és a felhasználói alkalmazásoknak az integrációját.

# Thread törlése

A szál törlése a szál életciklusának végét jelenti, amely akkor fordul elő, amikor egy szál befejezi a feladatát, vagy explicit módon megszakítják. A Pthreads (POSIX threads) kezelésére számos módszer létezik a szálak törlésére:

Aszinkron törlés: Egyik szál azonnal megállít egy másik szálat.

Halasztott törlés: A célszál időnként ellenőrzi, hogy törölni kell-e. Ebben az esetben a szál elvégzi az ellenőrzést, és csak bizonyos, előre meghatározott pontokon (cancellation points) hajtható végre a törlés.

A pthread\_cancel() funkcióval indítható a szál törlése, amely csak egy kérés a célszál törlésére, az aktuális törlés azonban attól függ, hogy a célszál hogyan állította be a törlési kérések kezelését. A POSIX szálak lehetővé teszik a szálak számára, hogy letiltsák vagy engedélyezzék a törlést. A törlés alapértelmezett típusa a halasztott törlés, ami azt jelenti, hogy a törlés csak akkor történik meg, ha a szál eléri a törlési pontot. Egy törlési pont elérésénél, ha függőben lévő törlési kérés található, egy úgynevezett takarító funkció (cleanup handler) hívódik meg, amely lehetővé teszi a szál számára, hogy felszabadítsa a megszerzett erőforrásokat, mielőtt a szál befejeződne.