Table of Contents

[Kritikus szakasz probléma – megoldás 1](#_Toc166848878)

[Szinkronizáló hardver 1](#_Toc166848879)

[Mutex zárak 2](#_Toc166848880)

[Szemaforok 3](#_Toc166848881)

[Klasszikus problémák: korlátos méretű puffer, olvasók-írók, étkező filozófusok 5](#_Toc166848882)

[Monitorok 5](#_Toc166848883)

[Feltételes változókkal rendelkező monitorok 7](#_Toc166848884)

**5. Folyamat szinkronizálás**

# Kritikus szakasz probléma – megoldás

A kritikus szakasz probléma megoldásának kulcsa a kölcsönös kizárás elvének biztosítása, amely garantálja, hogy a folyamatok egyidejűleg nem férhetnek hozzá ugyanahoz az erőforráshoz. A kölcsönös kizárás elérése érdekében több megközelítés létezik, mind hardveres, mind szoftveres szinten. Az egyik leggyakoribb megoldás a mutex zárak használata.

Mutex zárak: Ezek a zárak egy logikai változót használnak az elérhetőség jelzésére. Ha a zár szabad, a folyamat zárolhatja azt az acquire() hívással, ami belépés előtt megtörténik, és felszabadítja az release() hívással, amikor kilép a kritikus szakaszból. Ez a megoldás garantálja a kölcsönös kizárást, mivel egyidejűleg csak egy folyamat birtokolhatja a zárat .

Szemaforok: Ezek a szinkronizációs eszközök szintén használhatók a kritikus szakaszok kezelésére. A szemaforok jellegzetessége, hogy egy számláló alapján működnek, amely a rendelkezésre álló erőforrások számát jelzi. Egy folyamat, mielőtt belépne a kritikus szakaszba, csökkenti a számlálót, és növeli, amikor kilép, ezzel szabályozva az erőforrásokhoz való hozzáférést .

Hardveres megoldások: Néhány modern számítógépes rendszer speciális hardver utasításokat biztosít, mint például a "test and set" vagy a "compare and swap", amelyek atomi módon végrehajtva garantálják a kölcsönös kizárást. Ezek az utasítások lehetővé teszik, hogy egy memória terület tartalmát biztonságosan módosítsuk anélkül, hogy közben más folyamatok zavarnák a műveletet .

Ezek a technikák különböző előnyökkel és hátrányokkal rendelkeznek, és a választás gyakran függ az alkalmazás specifikus igényeitől és a rendelkezésre álló hardvertől. Az operációs rendszer tervezői általában egy kombinációt használnak ezen eszközök közül, hogy biztosítsák a hatékony és biztonságos folyamatok közötti szinkronizációt.

# Szinkronizáló hardver

A szinkronizáló hardver eszközök jelentősen megkönnyítik a kritikus szakasz probléma kezelését azáltal, hogy biztosítják az atomi műveletek elvégzését, amelyek nélkülözhetetlenek a kölcsönös kizárás és a versenyhelyzetek elkerülése érdekében. A kritikus szakasz probléma megoldására a modern számítógépes rendszerek speciális hardver utasításokat biztosítanak, amelyek lehetővé teszik egy szó tartalmának tesztelését és módosítását, vagy két szó tartalmának atomi cseréjét. Ezek az utasítások tartalmazzák a test and set és a compare and swap utasításokat, amelyek kulcsfontosságúak a kölcsönös kizárás biztosításában, különösen többprocesszoros környezetekben.

Test and Set: Ez az utasítás atomi módon végrehajt egy tesztelést és állít be egy értéket. Ha két ilyen utasítást egyszerre hajtanak végre különböző processzorokon, akkor ezek az utasítások egymás után, tetszőleges sorrendben hajtódnak végre. Ez az utasítás lehetővé teszi a lock nevű logikai változó használatát a kölcsönös kizárás megvalósítására.

Compare and Swap: Ez az utasítás három operandussal működik, és csak akkor módosít egy értéket, ha az megfelel a várt értéknek. Ez az utasítás szintén atomi módon hajtódik végre, és használható a kritikus szakaszok védelmére, ahol egy globális változó (zár) kezeli a belépést és kilépést .

Ezek az utasítások különösen hatékonyak kritikus szakaszok kezelésére, mivel atomi műveleteket biztosítanak, ami elengedhetetlen a versenyhelyzetek elkerüléséhez. A kölcsönös kizárás biztosítása mellett ezek a hardveres megoldások lehetővé teszik, hogy a szoftverfejlesztők és rendszermérnökök hatékonyabban kezeljék a szinkronizációs problémákat.

# Mutex zárak

A mutex zár (vagy kölcsönös kizárás zár) olyan szinkronizációs eszköz, amelyet arra használnak, hogy megakadályozzák több folyamat vagy szál egyidejű hozzáférését egy megosztott erőforráshoz. A mutex zárak biztosítják a kölcsönös kizárás elvét, ami alapvető a versenyhelyzetek elkerüléséhez és az adatok konzisztenciájának fenntartásához kritikus szakaszokban.

Mutex Zárak Működése

Zárszerzés (acquire()): Egy folyamat vagy szál megkísérli megszerezni a zárat a kritikus szakaszba való belépés előtt. Ha a zár szabad, a folyamat megszerzi azt és belép a kritikus szakaszba. Ha a zár foglalt, a folyamat blokkolódik, vagyis várakozni kezd, amíg a zár el nem érhetővé válik.

Zárfelszabadítás (release()): Miután a folyamat befejezte a kritikus szakaszban végzett műveleteket, felszabadítja a zárat, lehetővé téve más folyamatok számára, hogy megszerezzék azt.

Spinlocks és Mutex Zárak

A mutex zárak egy speciális formája a spinlock, amely egy aktív várakozást alkalmazó zár, ahol a folyamat egy ciklusban "forog", amíg a zár el nem érhető. Ez előnyös lehet olyan esetekben, ahol a zárak várhatóan csak rövid ideig tartanak fenn. A spinlockok használata különösen többprocesszoros rendszerekben hasznos, ahol egy szál egyik processzoron "foroghat", miközben egy másik szál egy másik processzoron végzi a kritikus szakaszt .

Mutex Zárak és Halottzárlat

A mutex zárak nem csak a versenyhelyzetek elkerülésére szolgálnak, hanem olyan problémák kezelésére is, mint a halottzárlat (deadlock). Egy jól ismert probléma, ahol a mutex zárak alkalmazása szükséges, az az úgynevezett "ebédző filozófusok" problémája, ahol minden filozófus két villához próbál hozzáférni az evéshez. A helyes zárkezelés biztosítja, hogy a filozófusok ne ragadjanak halottzárlatba azáltal, hogy egyszerre mindkét villára várakoznak .

Ezek az információk az operációs rendszerek tervezésekor és a többszálas alkalmazások fejlesztésekor kulcsfontosságúak a hatékony és biztonságos számítógépes rendszerek biztosításához.

# Szemaforok

A szemaforok olyan szinkronizációs eszközök, amelyek a folyamatok közötti hozzáférés szabályozására szolgálnak egy vagy több megosztott erőforráshoz. A szemaforokat két fő típusra lehet osztani: számláló és bináris szemaforokra.

Számláló szemaforok: Ezek a szemaforok egy pozitív egész számot tartalmaznak, amely azt jelzi, hogy hány erőforrás áll rendelkezésre. Ha egy folyamat erőforrást kíván használni, akkor a wait() művelettel csökkenti a szemafor értékét. Ha az erőforrás szabad, a folyamat folytathatja; ha nem, a folyamat blokkolódik. Amikor egy folyamat befejezi az erőforrás használatát, a signal() művelettel növeli a szemafor értékét, ezzel jelzi, hogy az erőforrás ismét szabad .

Bináris szemaforok: Ezek gyakorlatilag mutex zárakhoz hasonlóan működnek, azaz a szemafor értéke csak 0 vagy 1 lehet. A bináris szemafor használatakor a wait() művelet akkor állítja blokkoló állapotba a folyamatot, ha a szemafor értéke 0, ami azt jelenti, hogy az erőforrás foglalt. Amint a szemafor 1-re vált (amikor a signal() műveletet egy másik folyamat végrehajtja), a blokkolt folyamat folytathatja a végrehajtást .

A szemaforok tehát alapvető szerepet játszanak a kritikus szakaszok, erőforrások hozzáférésének szabályozásában, és különösen hasznosak lehetnek olyan összetettebb szinkronizációs problémák megoldására, mint a producer-consumer probléma, ahol több folyamat is hozzáférhet azonos erőforrásokhoz anélkül, hogy versenyhelyzet vagy halottzár állapot alakulna ki.

# Klasszikus problémák: korlátos méretű puffer, olvasók-írók, étkező filozófusok

A következő három klasszikus szinkronizációs probléma bemutatja a különböző típusú szinkronizációs kihívásokat és megoldásokat operációs rendszerekben:

Korlátos méretű puffer (Bounded Buffer) probléma: Ez a probléma a termelő-fogyasztó helyzetet modellezi, ahol a termelők és a fogyasztók között egy fix méretű puffer van. A termelők nem tudnak elemet hozzáadni a pufferhez, ha az tele van, és a fogyasztók nem vehetnek ki belőle, ha üres. A szinkronizációt szemaforok segítségével oldják meg, amelyek nyomon követik a szabad és a foglalt helyek számát, valamint egy mutex zárat használnak az adatok közös hozzáférésének védelmére .

Olvasók-írók probléma: Itt a kihívás abban áll, hogy több olvasó folyamat egyidejűleg férhet hozzá az adatokhoz olvasás céljából, de az íróknak kizárólagos hozzáférésre van szükségük az írás idején. A szinkronizációs megoldások között szerepel a különböző szemaforok használata az olvasók és írók számának kezelésére, biztosítva, hogy az írók ne éhezzenek, miközben az olvasók aktívak.

Étkező filozófusok probléma: Ez a probléma öt filozófust modellez, akik felváltva gondolkodnak és esznek. Minden filozófusnak két villa kell az evéshez, amelyek közül mindegyik egy-egy szomszéd között van. A megoldás szemaforok használatát foglalja magában, ahol minden villához egy szemafor tartozik, és a filozófusok csak akkor kezdhetnek enni, ha mindkét szomszédos villa elérhető. Ez a megközelítés segít elkerülni a halottzárat és az éhezést, de bizonyos feltételek mellett mégis előfordulhat éhezés.

Ezek a problémák és megoldások rávilágítanak a szinkronizációs mechanizmusok fontosságára a versenyhelyzetek kezelésében és a rendszer erőforrásainak hatékony kihasználásában operációs rendszerekben.

# Monitorok

A monitorok olyan magas szintű szinkronizációs absztrakciókat jelentenek, amelyeket a kölcsönös kizárás és az állapotváltozók kezelésére használnak programokban, különösen többszálú környezetekben. A monitorok egy adott szoftvermodulon belül garantálják a kölcsönös kizárást, és biztosítják, hogy csak egy szál futtatható aktívan a monitoron belüli kódrészen egy adott időben. Ezzel csökkentik a versenyhelyzetek kialakulásának lehetőségét és megkönnyítik a szinkronizációt.

Monitorok Alapelvei és Működése

Kölcsönös kizárás: A monitorok biztosítják, hogy a kritikus szekciókban egyszerre csak egy szál futtatható. Ezáltal megakadályozzák az adatinconsistentia és versenyhelyzetek kialakulását.

Állapotváltozók: A monitorok állapotváltozókat (condition variables) használnak, amelyek segítségével a szálak várakoztathatók vagy ébreszthetők bizonyos feltételek teljesülése esetén. Ezeket az állapotváltozókat a monitoron belül deklarálják, és két alapműveletet támogatnak: wait() és signal(). A wait() művelet egy szálat várakoztató állapotba helyez, míg a signal() művelet ébreszti a várakozó szálakat .

Szemaforok használata: A monitorok implementálásához gyakran használnak szemaforokat. Például egy mutex szemafor garantálja a kölcsönös kizárást a monitor belépési pontján, míg további szemaforok kezelik az állapotváltozókhoz kapcsolódó várakozásokat és ébresztéseket .

A monitorok egyik klasszikus alkalmazása az étkező filozófusok problémájának megoldása, ahol a filozófusok közötti versenyhelyzeteket és az esetleges halottzárakat (deadlocks) a monitoron keresztül kezelik, biztosítva, hogy a filozófusok egymást nem zavarják evés közben .

A monitorok használata segít csökkenteni a kölcsönös kizárás explicit programozásának szükségességét, és strukturáltabbá teszi a szinkronizációs logikát a többszálú rendszerekben.

# Feltételes változókkal rendelkező monitorok

A monitorokban a feltételes változók kulcsfontosságú szerepet játszanak a szinkronizációs mechanizmusok között, mivel lehetővé teszik a folyamatok számára, hogy várakozási állapotba kerüljenek egy bizonyos feltétel teljesüléséig. A feltételes változók két alapvető műveletet támogatnak: wait() és signal().

wait() művelet: Ezt a műveletet a folyamat hívja meg, amely várakozni kíván egy bizonyos feltétel teljesüléséig. A wait() művelet végrehajtásakor a folyamat várakozó állapotba kerül, és a monitorban végzett további végrehajtásra csak akkor kerülhet sor, ha a feltétel teljesül, amelyet egy másik folyamat signal() műveletének végrehajtása eredményezhet.

signal() művelet: Ezt a műveletet egy folyamat hajtja végre, amely értesíti a várakozó folyamatot, hogy a megadott feltétel mostmár teljesült. Ha a signal() művelet végrehajtásakor van várakozó folyamat, akkor a rendszer egyet kiválaszt a folytatásra. Ha nincs várakozó folyamat, a signal() műveletnek nincs hatása .

A monitorokban való folyamatok felfüggesztése és folytatása kritikus, mivel ezek a műveletek biztosítják, hogy a folyamatok nem fognak egymással versenyezni vagy egymást zavarni, miközben a monitoron belül végrehajtott kódrészleteken dolgoznak. A wait() és signal() műveletek alkalmazása biztosítja, hogy a folyamatok csak akkor lépjenek be a monitorba vagy folytassák a végrehajtást, ha a megfelelő szinkronizációs feltételek teljesülnek.