Table of Contents

[Virtuális memória koncepció 1](#_Toc166943049)

[Igény szerinti lapozás 1](#_Toc166943050)

[Laptábla, laphiba kezelés 2](#_Toc166943051)

[Copy on Write 3](#_Toc166943052)

[Laphelyettesítő algoritmusok 4](#_Toc166943053)

[Thrashing 6](#_Toc166943054)

[Working set model 7](#_Toc166943055)

**9. Virtuális memória**

# Virtuális memória koncepció

A virtuális memória koncepciója lényegében a logikai memóriát elválasztja a fizikai memóriától, lehetővé téve ezzel a folyamatok futtatását, akkor is, ha azok nem férnek el teljes egészében a rendelkezésre álló fizikai memóriában. Ez a technika az operációs rendszereknek azt az előnyét biztosítja, hogy a programok nagyobbak lehetnek, mint a fizikai memória, és megkönnyíti a fájlok megosztását a folyamatok között, valamint az új folyamatok hatékony létrehozását is lehetővé teszi.

A virtuális memória a logikai memóriát egy nagyon nagy, egységes tárolási tömbként kezeli, ami a felhasználó számára megkülönböztethetetlen a fizikai memóriától. Egy folyamat virtuális címtartománya a folyamat memóriában való tárolásának logikai (vagy virtuális) nézetét jelenti, ami általában azt sugallja, hogy a folyamat egy adott logikai címen kezdődik – például a 0 címen – és egybefüggő memóriában található .

A virtuális memória a memóriakezelési egység (MMU) segítségével képezi le a logikai oldalakat a fizikai keretekre a memóriában. Ennek a technikának az előnyei között szerepel, hogy felszabadítja a programozókat a fizikai memória mennyiségével kapcsolatos aggodalmaktól, lehetővé téve, hogy inkább a programozandó problémára koncentráljanak. Emellett lehetővé teszi a folyamatok számára, hogy könnyedén megoszthassanak fájlokat és implementálhassanak megosztott memóriát .

# Igény szerinti lapozás

Az igény szerinti lapozás (demand paging) egy virtuális memória technika, amelynek lényege, hogy csak azokat a lapokat tölti be a rendszer, amelyekre éppen szükség van a program végrehajtása során. Ez a módszer segít megelőzni a felesleges memória használatot és csökkenti a betöltési időt, mivel nem szükséges az egész programot fizikai memóriába tölteni a futtatás megkezdésekor.

Működése

Amikor egy folyamatot futtatni szeretnénk, kezdetben egyáltalán nem töltünk be semmilyen oldalt a memóriába. Amikor a folyamat egy olyan oldalhoz hozzáférne, amely nincs a memóriában, akkor oldalhibát (page fault) generál, ami jelzi a rendszernek, hogy az adott oldalra szükség van. Ezt követően a lapozási alrendszer betölti az oldalt a háttértárról (általában lemezről) a fizikai memóriába, és frissíti az oldaltáblát az új információkkal.

Előnyei

Az igény szerinti lapozás segít optimális memóriahasználatot elérni, mivel csak a ténylegesen használt oldalak vannak a memóriában. Ez jelentősen csökkenti a memória szükségletet, lehetővé téve több folyamat egyidejű futtatását, javítva ezzel a CPU kihasználtságát. Továbbá, mivel az oldalak csak akkor kerülnek be a memóriába, amikor arra szükség van, csökken a rendszer indításának ideje és a folyamatok indításának késleltetése.

Kihívások

Az igény szerinti lapozás bevezetésekor számolni kell azzal, hogy a lapok betöltése oldalhibával és lemezműveletekkel jár, amelyek megnövelhetik az elérési időt, ha a lapok gyakran nem állnak rendelkezésre a fizikai memóriában. Ennek optimalizálására a lapcserélési algoritmusok (mint az LRU, FIFO stb.) és az oldalhibák gyakoriságának minimalizálása érdekében speciális technikák, mint a lokális referencia modellezése és oldalbetöltési stratégiák alkalmazására van szükség .

# Laptábla, laphiba kezelés

A laptábla használata és a laphiba kezelés kritikus aspektusai a virtuális memória működésének. A laptábla tartalmazza az egyes oldalak (lapok) keretszámát (frame number), amelyek a logikai címek fizikai memóriacímekké történő fordításához szükségesek. A laptábla minden egyes oldalához társul egy érvényességi bit, amely jelzi, hogy az adott oldal éppen a memóriában van-e (érvényes), vagy nem (érvénytelen).

Laphiba Kezelése

Amikor egy folyamat egy nem memóriában lévő oldalra próbál hozzáférni, az egy laphibát eredményez, amelyet a rendszernek kezelnie kell. A laphiba kezelésének lépései a következők:

Laphiba azonosítása: A laptábla segítségével megállapítják, hogy az oldal érvénytelen (nem a memóriában). Ez automatikusan csapdához vezet az operációs rendszer szintjén.

Oldal betöltése: Az operációs rendszer meghatározza, hogy az oldal hol található a háttértáron, és elindít egy lemezolvasási műveletet az oldal memóriába való betöltéséhez.

Szabad keret keresése: Amennyiben nincs szabad keret, a rendszernek oldalcsere-algoritmust kell alkalmaznia egy használt keret felszabadításához.

Folyamat folytatása: Miután az oldal bekerült a memóriába és a laptábla frissült, az utasítás, amely a laphibát okozta, újraindítható, mintha az oldal mindig is a memóriában lett volna.

Ezek a lépések biztosítják, hogy a virtuális memória hatékonyan kezelje a memória korlátozott fizikai kapacitását, lehetővé téve a folyamatok számára, hogy több memóriát használhassanak, mint amennyi fizikailag elérhető​​.

# Copy on Write

A Copy on Write (COW) egy optimalizációs technika, ahol a rendszer a folyamatok között megosztott lapokat csak akkor másolja le, ha valamelyik folyamat módosítani kíván egy ilyen lapot. Kezdetben a folyamatok ugyanazokat a fizikai lapokat használhatják, amíg valamelyikük írási műveletet nem hajt végre egy lapra. Írási kísérlet esetén a rendszer egy másolatot készít a lapról, és ezt rendeli hozzá a módosítást végző folyamat címtartományához. Ez a technika jelentős memória-megtakarítást eredményezhet, mivel csak a szükséges módosítások esetén szükséges új lapokat foglalni​​.

# Laphelyettesítő algoritmusok

A laphelyettesítő algoritmusok kulcsfontosságúak a virtuális memóriarendszerekben, ahol a memóriában lévő oldalakat lemezre cserélik ki a szükségesekkel. Itt vannak a legfontosabb laphelyettesítő algoritmusok és jellemzőik:

FIFO (First-In-First-Out): Ez az algoritmus a memóriában legkorábban betöltött oldalakat helyettesíti először. Egyszerűen kezelhető, de hajlamos a Belády-anomáliára, ahol a memória növelése több lapkiesést okozhat​​.

Optimális algoritmus (OPT): Az a lap cseréli ki, amely a leghosszabb ideig nem lesz szükséges. Ez garantálja a legkisebb lapkiesési arányt, de gyakorlati használata megvalósíthatatlan, mivel előre kellene ismerni az oldalhivatkozásokat​​.

LRU (Least Recently Used): Az LRU azokat az oldalakat helyettesíti, amelyek a legtávolabbi múltban voltak használatosak. Ez jobban megközelíti az optimális algoritmust, mivel figyelembe veszi az oldalak használati gyakoriságát. Az LRU algoritmus implementálása bonyolult lehet, mivel nyilvántartani kell az oldalhivatkozások idejét​​.

Second-Chance (második esély): Egy egyszerűsített LRU változat, amely a FIFO módszert használja, de egy úgynevezett referenciabittel kiegészítve, amely jelzi, ha egy oldalt használtak. Ha egy oldal kijelölésre kerül és a referenciabite 1-es, akkor a lap egy "második esélyt" kap, és az algoritmus a következő oldalt választja ki cserére​​.

Clock (óra): Gyakran használt LRU-alternatíva, amely az oldalakat egy kör alakú listában tartja nyilván, és egy mutatóval jelzi, melyik oldalt kell cserélni. Az oldalakat egy "óra" mutatójával pörgetik, és a használt oldalak maradnak, míg a nem használtak cserére kerülnek​​.

Ezek az algoritmusok különböző teljesítményt nyújtanak a különböző operációs rendszerekben és alkalmazásokban, attól függően, hogy milyen memóriahasználati mintákat mutatnak a futó folyamatok. Az ideális algoritmus kiválasztása az adott rendszer igényeihez és a lapkiesési arány minimalizálásához szükséges.

# Thrashing

A thrashing jelenség a virtuális memória túlhasználatakor jelenik meg, amikor a rendszer túl sok időt fordít az oldalcsere-folyamatok kezelésére, ami jelentősen lelassítja a rendszer működését. A thrashing akkor következik be, amikor a rendszerben futó folyamatok memóriaigénye meghaladja a rendelkezésre álló fizikai memória méretét. Ebben az esetben a rendszer gyakran cseréli az oldalakat a merevlemez (swap space) és a RAM között, ami drasztikusan lecsökkenti a CPU hatékonyságát, mivel kevesebb idő jut a folyamatok végrehajtására.

Amikor a rendszer thrashing állapotba kerül, a folyamatok gyakran várakoznak a memória hozzáférésre, mivel a rendszer a szükséges oldalakat próbálja betölteni a swap területről. Ez az állapot jelentősen ronthatja a rendszer teljesítményét, mivel az erőforrások egy jelentős részét a memória-kezelés emészti fel.

A thrashing problémájának kezelésére több megoldás is létezik:

A rendszer memóriakapacitásának növelése.

A futtatott folyamatok számának csökkentése vagy a memóriaigények optimalizálása.

Okosabb oldalcserélő algoritmusok használata, amelyek csökkentik a szükségtelen oldalcserék számát.

A virtuális memória kezelési technikáinak fejlesztése, például a demand paging és a copy-on-write technikák alkalmazása.

A thrashing kezelése kulcsfontosságú lehet a rendszer stabilitásának és hatékonyságának fenntartásában, különösen olyan környezetekben, ahol a memória korlátozott és a folyamatok nagy memóriaigényűek.

# Working set model

A working set model a virtuális memória kezelésének egyik módszere, amelyet Peter Denning vezetett be. Ez a modell azt próbálja meg határozni, hogy mely oldalak szükségesek egy adott időszakban egy folyamat futtatásához. A working set, vagy munkahalmaz, egy adott időablakon belül hivatkozott oldalak halmaza, amelyek szükségesek ahhoz, hogy a folyamat zökkenőmentesen futtatható legyen.

A working set modell célja kettős:

Teljesítmény növelése: A modell segítségével a rendszer megpróbálja a memóriában tartani azokat az oldalakat, amelyekre gyakran van szükség, ezzel csökkentve az oldalcserék számát és növelve a rendszer teljesítményét.

Thrashing megelőzése: Segít kiegyensúlyozni a memória kiosztását a futó folyamatok között úgy, hogy mindegyik megkapja a működéséhez szükséges minimális memóriamennyiséget, ezzel elkerülve a thrashing állapotot.

A working set modell alapelvei a következők:

Időablak: A working set méretét az adott időablak alatt hivatkozott oldalak határozzák meg. Ez az időablak lehet fix vagy változó hosszúságú, és a folyamat memóriaigényének változása szerint állítható.

Oldalbetöltési stratégia: Ha egy oldal nem tartozik a folyamat aktuális working set-jéhez, és a memória szűkös, akkor azt ki lehet tenni a memóriából.

Dinamikus kezelés: A working set dinamikusan változik a folyamat futása során, az oldalhivatkozások alapján frissül.

Gyakorlati alkalmazás:

A working set modelt több modern operációs rendszer is alkalmazza, mint például a Windows és néhány UNIX-szerű rendszer, ahol az oldalcsere döntéseknél figyelembe veszik a folyamatok working set-jét. Ez lehetővé teszi, hogy az operációs rendszer hatékonyabban gazdálkodjon a rendelkezésre álló memóriával, optimalizálja a CPU kihasználtságát, és javítsa az általános rendszerreakcióidőt.

Ez a modell különösen hasznos nagy, többfolyamatos környezetekben, ahol a memória erőforrások korlátozottak, és fontos a magas rendszerhatékonyság és stabilitás.