Table of Contents

[Ütemezési kritériumok 1](#_Toc166849719)

[Ütemezési algoritmusok: FCFS, SJF, SRTF, RR, prioritásos ütemezés 2](#_Toc166849720)

[Többszintű sor ütemezés 2](#_Toc166849721)

[Szál ütemezése 3](#_Toc166849722)

[Többprocesszoros ütemezés 4](#_Toc166849723)

[Valós idejű ütemezés 5](#_Toc166849724)

[Rate monoton és a legkorábbi határidő elsőként ütemezés 5](#_Toc166849725)

**6. CPU ütemezés**

# Ütemezési kritériumok

A CPU ütemezési kritériumok a következők:

CPU kihasználtság: Az ütemezőnek maximalizálnia kell a CPU kihasználtságát, ami azt jelenti, hogy a CPU-t a lehető legtöbb időben aktívan használják. Ideális esetben a CPU kihasználtsága 0 és 100% között változik, de a valós rendszerekben 40% (könnyű terhelésű rendszerekben) és 90% (nehéz terhelésű rendszerekben) között mozog .

Áteresztő képesség (Throughput): Ez arra utal, hogy mennyi munka végezhető el adott idő alatt. Az áteresztő képesség a befejezett folyamatok számával mérhető időegységenként. A hosszabb folyamatok esetében ez lehet óránként egy folyamat, míg rövidebb tranzakcióknál akár másodpercenként tíz folyamat is lehet .

Fordulóidő (Turnaround time): Egy adott folyamat szemszögéből ez az időtartam, ami alatt a folyamat befejeződik. A fordulóidő magában foglalja az időtartamokat, amelyek alatt a folyamat várakozik a memóriába kerülésre, a készenléti sorban várakozik, a CPU-n fut, és I/O műveleteket végez.

Várakozási idő (Waiting time): Az ütemezési algoritmus a folyamatok várakozási idejét befolyásolja a készenléti sorban. Ez az időtartam a készenléti sorban töltött idő összegét jelenti.

Válaszidő (Response time): Interaktív rendszerekben a fordulóidő nem feltétlenül a legjobb mérőszám. Gyakran előfordul, hogy egy folyamat képes előállítani némi kimenetet viszonylag korán, és új eredményeket számíthat ki, miközben az előző eredményeket a felhasználónak továbbítják. Ebben az esetben a válaszidő a kérelem benyújtásától az első válasz előállításáig eltelt időtartamot jelenti.

Ezek a kritériumok segítenek a CPU ütemezési algoritmusok összehasonlításában és kiválasztásában, hogy megfeleljenek a rendszer és az alkalmazások különböző igényeinek.

# Ütemezési algoritmusok: FCFS, SJF, SRTF, RR, prioritásos ütemezés

Elsőként érkezők elsőként szolgáltatva (FCFS): Az egyik legegyszerűbb CPU ütemezési algoritmus. Ez az algoritmus a folyamatokat abban a sorrendben ütemezi, ahogyan azok megérkeznek a készenléti sorba, tehát FIFO (elsőként be, elsőként ki) módon kezeli őket. Hátránya, hogy hosszú várakozási idők alakulhatnak ki, ha egy hosszú CPU kitörésű folyamat érkezik először​​.

A legrövidebb munka először (SJF): Ez az algoritmus mindegyik folyamathoz hozzárendeli a következő CPU kitörés hosszát, és a legkisebb következő CPU kitöréssel rendelkező folyamatot választja ki futtatásra. Ezt a módszert optimálisnak tekintik, mivel minimalizálja az átlagos várakozási időt a folyamatok számára. Azonban a következő CPU kitörés hosszának előrejelzése nehézségeket okozhat​​.

Legkisebb fennmaradó idő először (SRTF): Az SJF preemptív változata, amely azonnal megváltoztatja a futó folyamatot, ha egy újonnan érkező folyamatnak rövidebb a hátralévő futási ideje, mint a jelenleg futónak​​.

Körkörös (RR): Ez az algoritmus időmegosztó rendszerekre van tervezve, ahol minden folyamat egyenlő, előre meghatározott időtartamot kap a CPU-n (időkvantum), majd ha ez lejár, a folyamatot a készenléti sor végére helyezi. Az időkvantum hosszának megválasztása kritikus, mert ha túl rövid, túl nagy lehet az ütemezési overhead, ha túl hosszú, akkor az algoritmus FCFS-sé degradálódik​​.

Prioritásos ütemezés: Ebben a módszerben minden folyamathoz prioritást rendelnek, és a CPU-t a legmagasabb prioritású folyamat kapja meg először. Ez lehet preemptív vagy nem preemptív. A prioritásokat különböző tényezők, mint például a válaszidő, az erőforrásigények, vagy akár külpolitikai tényezők alapján határozhatják meg​​.

Ezek az algoritmusok különböző környezetekben és követelmények mellett különböző hatékonysággal működnek, és a megfelelő algoritmus kiválasztása az operációs rendszer tervezésének fontos része.

# Többszintű sor ütemezés

A többszintű sor ütemezési algoritmus a készenléti sort több különálló sorba osztja fel, amelyeket különböző tulajdonságok alapján rendelnek a folyamatokhoz, például memória mérete, folyamat prioritása vagy típusa. Minden sor saját ütemezési algoritmussal rendelkezik, és a folyamatok véglegesen hozzá vannak rendelve egy adott sorhoz, amikor belépnek a rendszerbe.

A különböző sorok között is szükség van ütemezésre, amelyet általában fix prioritású preemptív ütemezéssel valósítanak meg. Például az interaktív (előtér) folyamatok sorának abszolút prioritása lehet a háttérfolyamatok sorával szemben. Ez azt jelenti, hogy egy interaktív szerkesztési folyamat érkezése a készenléti sorba megszakíthat egy éppen futó háttér folyamatot, ha szükséges.

Továbbá lehetséges az időszeletelés alkalmazása a különböző sorok között is, ahol minden sor egy meghatározott részesedést kap a CPU időből, amit aztán saját belső folyamatai között oszthat meg. Például egy előtér-háttér sor esetében az előtér sor 80%-os, míg a háttér sor 20%-os CPU időt kaphat, ahol az előtér sorban az RR (Round Robin), míg a háttér sorban az FCFS (First-Come, First-Served) algoritmus szerint történik az ütemezés .

# Szál ütemezése

A szál ütemezése az operációs rendszerek egyik kulcsfontosságú funkciója, amely lehetővé teszi a folyamatokon és szálakon belüli erőforrás-megosztást és -kezelést. Itt a hangsúly kifejezetten a szálak ütemezésére helyeződik, ami különbözik a folyamatok ütemezésétől. A szálak ütemezésével kapcsolatos megközelítések többféle modellel valósíthatók meg, amelyeket két fő csoportba sorolhatunk: felhasználói szintű és kernel szintű szálak kezelése.

Felhasználói szintű szálak: Ezeket a szálakat egy felhasználói szintű könyvtár kezeli, és az operációs rendszer kernelje nem tud róluk. A felhasználói szintű szálakat a könyvtár ütemezi és osztja be egy vagy több kernel szálra. Ez a modell lehetővé teszi, hogy a felhasználói alkalmazások finomabban szabályozzák a szálkezelés működését, de korlátozza azokat az eseteket, amikor a kernelnek közvetlenül kellene beavatkoznia.

Kernel szintű szálak: Ebben a modellben minden szál közvetlenül a kernel által van kezelve, ami lehetővé teszi a szálak jobb integrációját az operációs rendszer többi részével, beleértve az ütemezési és erőforrás-kezelési funkciókat is. Ez a megközelítés jobban kihasználja a többmagos processzorok adta lehetőségeket, mivel a kernel képes minden egyes szálat külön ütemezni és kezelni.

A szálak ütemezésében két versenyterületet különböztetünk meg:

Process-Contention Scope (PCS): Itt a verseny a folyamathoz tartozó szálak között zajlik, ahol a szálakat a folyamathoz tartozó LWP (Lightweight Process) kezeli.

System-Contention Scope (SCS): Ebben az esetben a verseny az összes rendszerben lévő szál között zajlik, és a kernel dönti el, hogy melyik szál futtatása legyen prioritás​​.

Ezek a modellek különböző előnyöket és hátrányokat kínálnak attól függően, hogy a rendszer milyen típusú feladatokat lát el, és milyen hardveres konfiguráció áll rendelkezésre. Az optimális szálkezelési stratégia kiválasztása kulcsfontosságú a rendszer teljesítményének és válaszkészségének maximalizálása érdekében.

# Többprocesszoros ütemezés

A többszintű sor ütemezés több különböző készenléti sort használ, amelyek mindegyikének saját ütemezési algoritmusa van. Ezek a sorok általában a folyamatok tulajdonságai alapján vannak kategorizálva, mint például a memóriaigény, a folyamat prioritása vagy típusa. Például elkülöníthetőek előtérbeli (interaktív) és háttérbeli (batch) folyamatok sorai. Az előtérbeli sorokat gyakran körkörös (RR) algoritmussal, míg a háttérbeli sorokat elsőként érkezett, elsőként szolgált (FCFS) algoritmussal ütemezik. A sorok közötti ütemezést általában fix prioritású preemptív ütemezéssel valósítják meg, ahol az előtérbeli sorok abszolút prioritást élveznek a háttérbeli sorokkal szemben .

A többszintű visszajelzős sor ütemezés egy rugalmasabb változata a többszintű sor ütemezésnek, amely lehetővé teszi a folyamatok mozgatását a különböző sorok között. Ennek alapötlete, hogy a folyamatokat a CPU-használatuk jellemzői alapján kategorizálja. Ha egy folyamat túl sok CPU-időt használ, alacsonyabb prioritású sorba kerülhet. Ez a rendszer az I/O-intenzív és interaktív folyamatokat magasabb prioritású sorokban tartja, míg egy folyamat, amely túl hosszú ideig marad egy alacsonyabb prioritású sorban, magasabb prioritású sorba kerülhet, ezzel elkerülve az éhezés jelenségét.

# Valós idejű ütemezés

A valós idejű operációs rendszerek CPU-ütemezési stratégiái különösen fontosak a kritikus időzítésű feladatok kezelésében. Ezek az ütemezési stratégiák két fő kategóriába sorolhatók: puha (soft) és kemény (hard) valós idejű rendszerek.

Puha valós idejű rendszerek: Ezekben a rendszerekben a valós idejű folyamatok magas prioritást élveznek a nem kritikus folyamatokkal szemben, de a rendszer nem garantálja, hogy a kritikus folyamatok pontosan mikor lesznek ütemezve. A puha valós idejű rendszerek jellemzően relatív prioritási garanciákat kínálnak, azaz biztosítják, hogy a valós idejű folyamatokat előnyben részesítik, de nem szolgáltatnak garanciát arra, hogy egy folyamat mikor lesz pontosan ütemezve, ha futtatható állapotba kerül .

Kemény valós idejű rendszerek: Ezek a rendszerek szigorúbb követelményeket támasztanak. Ezekben a rendszerekben a feladatokat a határidejük előtt kell kiszolgálni; a határidő lejárta utáni kiszolgálás egyenértékű a kiszolgálás elmaradásával. A kemény valós idejű rendszerek garantálják, hogy a kritikus folyamatokat az előírt határidőn belül kiszolgálják .

# Rate monoton és a legkorábbi határidő elsőként ütemezés

Ezen kívül, a valós idejű CPU-ütemezési algoritmusok két gyakori típusa a következő:

Rate-Monotonic Scheduling (RMS): Ez az ütemezési algoritmus statikus prioritást rendel az egyes folyamatokhoz a periódusuk hossza alapján. Rövidebb periódusú folyamatok magasabb prioritást kapnak. Ez az algoritmus feltételezi, hogy a folyamatok periódikusak és minden CPU-robbanásuk megegyezik az időtartamukban .

Earliest-Deadline-First (EDF) Scheduling: Ez az ütemezési algoritmus dinamikusan állítja be a prioritásokat a határidők alapján. A korábbi határidővel rendelkező folyamatok magasabb prioritást kapnak. Az EDF politika lehetővé teszi, hogy a prioritásokat igazítsák a futtatható állapotba kerülő új folyamat határidőihez .

Ezek az ütemezési stratégiák különösen fontosak olyan rendszerekben, ahol a folyamatoknak szigorú időbeli korlátoknak kell megfelelniük, például beágyazott rendszerekben vagy missziókritikus alkalmazásokban.