Operációs rendszerek

Ütemezés

1.1

Pere László (pipas@linux.pte.hu)

PÉCSI TUDOMÁNYEGYETEM TERMÉSZETTUDOMÁNYI KAR

INFORMATIKA ÉS ÁLTALÁNOS TECHNIKA TANSZÉK

A scheduler

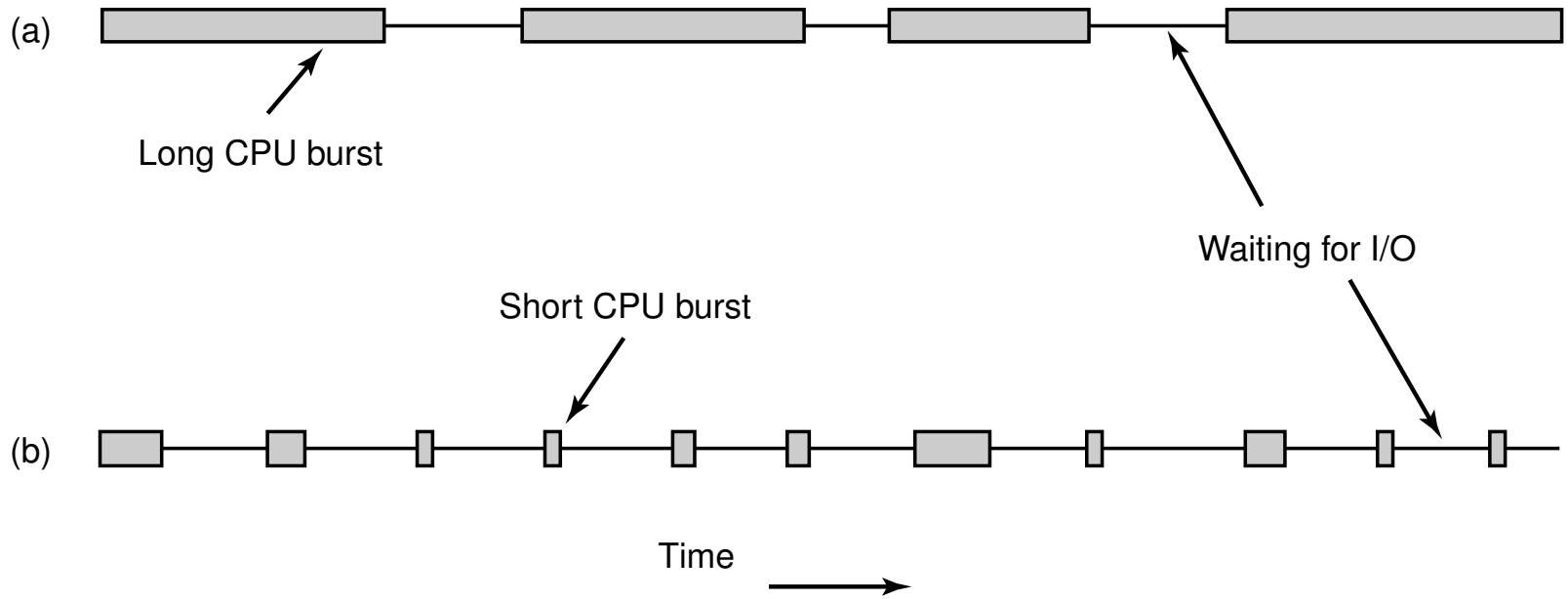
Ha több folyamat van ready állapotban, mint ahány processzor van, dönteni kell hogy melyiket futtassuk. A döntést az ütemező (*sceduler*) végzi az ütemező algoritmus (*scheduler algorithm*) alapján.

A ütemezés nagymértékben meghatározza a számítógép viselkedését, a felhasználók elégedettségét (ha vannak felhasználók!).

Mikor kell dönteni?

- Új folyamat létrehozásakor dönteni kell, hogy a szülő vagy a gyermek fusson.
- Amikor egy processz befejeződik, döntenünk kell, hogy a ready processzek közül melyiket futtassuk.
- Ha a futó processz blokkolódik (I/O, szemafor), másik processzt kell választani.
- Ha I/O IRQ fut be, döntenünk kell, hogy a blokkolás alól felszabadult processz fusson-e?
- Preemtive scheduler esetén időszakonként megszakítás érkezik, amelyik kiváltja a schedulingot.

A folyamatok viselkedése



1. ábra. CPU- és I/O igényes folyamatok

A folyamatok viselkedése

Alapjában véve kétféle viselkedést figyelhetünk meg:

1. CPU igényes viselkedés: hosszú CPU burst-ök, számításigényes feladatok.
2. I/O igényes viselkedés: rövid CPU burst-ök, I/O igényes viselkedés.

A CPU sebessége gyorsabban növekszik a technológiai fejlődés során, mint a háttértárak sebessége: egyre inkább I/O igényes viselkedés figyelhető meg.

Ha az I/O igényes folyamat futni akar, hagynunk kell, hogy jobban kihasználja a perifériákat.

Preemptive

A preemptive scheduler képes elvenni a futás jogát a folyamattól akkor is, ha az nem mond le a futásról és nem küld rednszerhívást.

A preemptive OS olyan operációs rendszer, amelynél kernel módban is preemptive schedulert használ.

Nyilvánvaló, hogy preemptive scheduler $\neq$ preemptive OS.

A környezet

Más igények más scheduling algoritmust igényelnek. A következő környezeteket különböztetjük meg:

- batch (hosszú időszeletek, preemtive vagy nem-preemtive schedulerrel)
- interactive (a preemtive scheduler általános követelmény)
- real time (dedikált programok, preemtive scheduling nem igazán elég, sokszor nem kell, mert a processzek tudják mi a dolguk)

Célok I.

Minden rendszeren felmerülnek a schedulerrel szemben a következő követelmények:

- Igazságosság (minden folyamat kapjon korrekt esélyt a futásra)
- Házirend betartása (ha vannak szabályok, azoknak érvényesülniük kell)
- Egyensúly (minden erőforrás ki legyen használva)

Célok II.

Batch rendszerek esetében felmerülő igények:

- Teljesítmény (lehető legtöbb job adott időegység alatt)
- Válaszidő minimalizálás (job feladás és vége közt minél rövidebb várakozás)
- CPU terhelés (lehetőleg folyamatosan magas CPU terhelés)

Célok III.

Interaktív rendszereken felmerülő igények:

- Válaszidő kordában tartása, a kérések időben történő teljesítése
- Igények kielégítése, a felhasználók szubjektív megelégedettségének követelménye

Célok III.

Real-time rendszereken felmerülő igények:

- Válaszidők betartása
- Bejósolhatóság, multimédia rendszereken minőségromlás elkerülése

Batch rendszerek

A következő néhány scheduler algoritmust batch rendszereken használhatjuk.

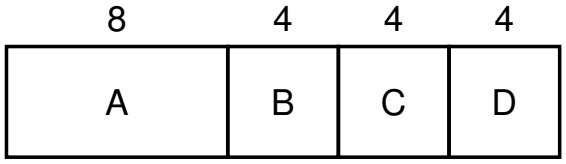
First-Come First-Served

Ez a nonpreemtive scheduling algoritmus a következőképp működik: mindig az kapja meg a processzort, aki először kéri. Ha egy folyamat blokkolódik, a sor következő folyamata fut. Ha egy folyamat futásra késszé válik, a sor végére kerül.

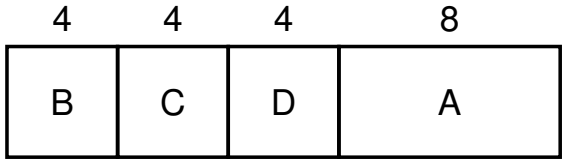
Ez a folyamat az I/O igényes folyamatok számára igen kedvezőtlen.

Shortest job first

A feladatok végrehajtására szükséges idő sokszor megbecsülhetõ, hiszen ugyanazokat a programokat futtatják újra és újra. Ha a feladatok egy időben rendelkezésre állnak, a hosszabb futási idejű feladatokat hátra sorolva rövidebb átlagos várakozási idő érhetõ el.



(a)



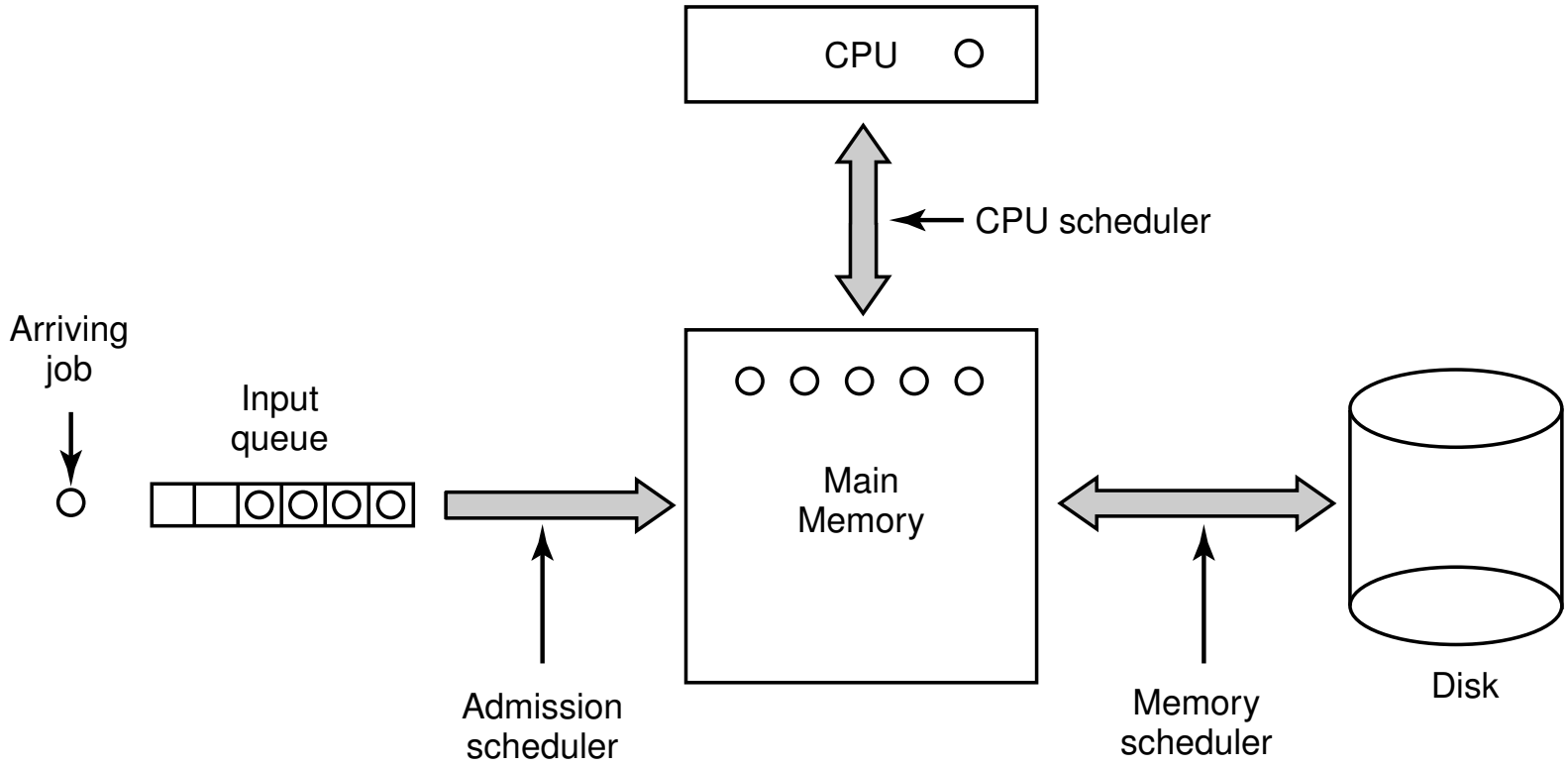
(b)

Shortest remaining time next

A shortest job first algoritmus preemtive változata a shortest remaining time next.
Ez az algoritmus azokat a joboknak ad elsőbbséget, amelyeknek futási idejéből kevesebb van hátra.

Háromszintû scheduling

A batch rendszerekben három helyre építhető scheduler. Háromszintű schedulingról beszélhetünk, ha mindhárom helyen ütemezés történik.



Admission scheduler

A bemeneten található, eldönti, hogy melyik beérkezõ jobot engedi a rendszerbe és melyiket nem.

Az admission scheduler megteheti, hogy a beérkezett jobot várakoztatja és a késõbb jövõt részesíti elõnyben.

Memory scheduler

A memory scheduler eldönti, hogy melyik job legyen a memóriába és melyiket helyezze el ideiglenesen a háttértáron.

A memory scheduler törekedhet pl. arra, hogy a memóiában folyamatosan legyenek CPU igényes és I/O igényes jobok is. Megteheti, mert statisztikát készítve a jobokról meg tudja állapítani melyik csoportba sorolhatóak.

A memory scheduler dönti el a multiprogramozás fokát (*degree of multiprogramming*).

CPU scheduler

A CPU scheduler dönti el, hogy az aktuális pillanatban melyik job fusson.

Amikor egyszerűen ütemezőről beszélünk, általában erről a komponensről van szó.

Interaktív rendszerek

A következő néhány scheduler algoritmust interaktív rendszerekben használhatjuk.

Ezek az algoritmusok batch rendszerekben is használhatóak CPU schedulerként.

(Interaktív rendszerekben háromszintû schedulert nem használhatunk, de kétszintűt igen, CPU és memory scheduler lehet interaktív rendszerekben is.

Round-Robin

A folyamatok körben vannak elhelyezve, ha a processz blokkolódik vagy lejár az ideje, a sor végére kerül és következik az utána található processz.

Az egyetlen kérdés ennél az algoritmusnál, hogy mennyi időnként hívjuk megszakításból a schedulert.

2. ábra. A Round-robin scheduler

Preemptive időzítés

Túl rövid időközöket beállítva rossz lesz a hatékonyság – túl sok idő fordítódik az átkapcsolásokra –, túl hosszú időközöket beállítva viszont rossz lesz a válaszidő.

A $20 - 50ms$ időzítés megfelelőnek tűnik.

Priority scheduling

Minden processzhez rendelünk egy prioritást és mindig a legmagasabb prioritású futásképes processznek adjuk a futás jogát.

Annak érdekében, hogy elkerüljük a kiéheztetést megtehetjük, hogy:

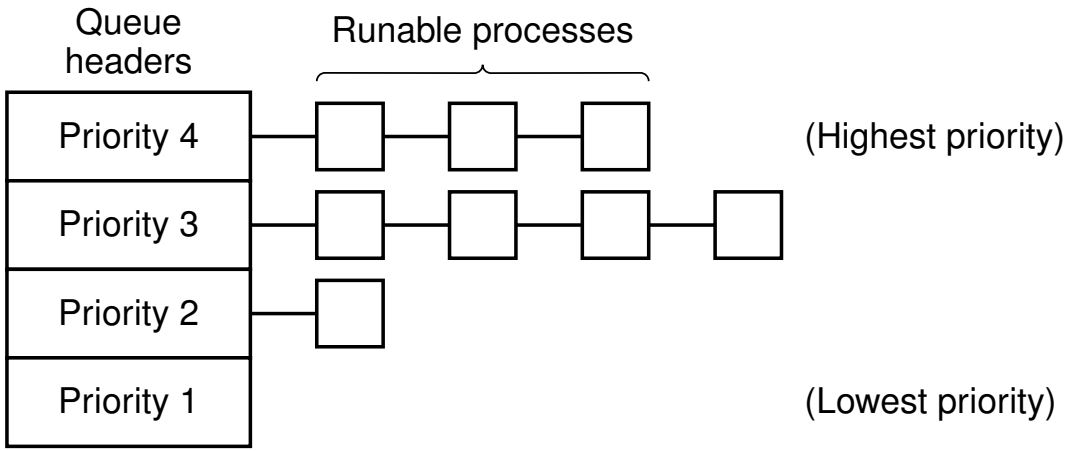
- Minden körben csökkentjük a futó processz prioritását.
- Maximáljuk az időablakot, amelyben a processz futhat.

A prioritásokat kezelhetjük statikusan (pl. professzori szint felett 100), vagy dinamikusan (pl. I/O bound processzek magasabb prioritással futnak).

Priority scheduling

A prioritással szervezett scheduling-ot összekapcsolhatjuk a round robin módszerrel.

Minden prioritási szinthez tartozik egy round robin kör. Ha a magasabb szinten nincsen futásképes processz, az alacsonyabb szinten lévők kapják meg a jogot.



3. ábra. Ütemezés prioritási osztályokkal

Shortest Process Next

A shortest job first jó átlagos válaszidőt adott, ezért szerencsés volna interaktív rendszerekbe is bevezetni.

Ha a terminálon kiadott minden egyes parancsot önálló jobnak tekintünk, akkor például bevezethetünk ilyen scheduling algoritmust.

Garantált scheduling

A garantált scheduling lényege, hogy igéretet teszünk a felhasználónak, hogy adott CPU mennyiséget kap.

A legegyszerűbb ígéret: ha n felhasználó (processz) van, akkor $\frac{1}{n}$ részét kapja meg a felhasználó (processz).

Számítsuk ki, hogy az adott pillanatban a processz által fogyasztott CPU idő és a neki megígért CPU idő hányadosa mennyi és a legkisebb hányadossal rendelkező processzt futtassuk.

Lottery scheduling

Minden processz ticketeket kap, amelynek több a ticketje, nagyobb eséllyel fog futási jogot kapni.

A kooperáló processzek ticketet adhatnak át egymásnak.

Fair-share scheduling

Ha egy felhasználó több processzt futtat mint a többi, akkor több processzoridőt kap, hiszen nagyobb az esélye annak, hogy az ő processzei kapnak futási jogot.

Néhány rendszer megvizsgálja, hogy az adott processzt ki futtatja és megpróbálja igazságosan elosztani a processzoridőt. Ez az igazságos scheduling.

Real-time rendszerek

Real-time rendszerekben általában egy vagy több eszköz ingereket ad a rendszernek és annak bizonyos időn belül válaszolnia kell ezekre az ingerekre.

A következőkben áttekintjük a real-time rendszerek ütemezését.

Real-time rendszerek típusai

Felosztás igény szerint:

- Hard real-time systems: az időzítéseket mindenképpen be kell tartani.
- Soft real-time systems: az időhatárok túllépése nem szerencsés, de bizonyos esetekben elfogadható.

Felosztás környezet szerint:

- Peridikus, azaz az „ingerek” periodikus jelleggel érkeznek. (akár több eseménysor)
- Aperiodikus, azaz az ingerek bejósolhatatlan jelleggel ismétlődnek.

Schedulable

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1 \tag{1}$$

m darab eseményt kell lekezelni.

P_i az i-edik esemény periódusideje.

C_i az i-edik esemény kezeléséhez szükséges idõ.

Feltételezzük, hogy a context switching elhanyagolható időt vesz igénybe.

Statikus és dinamikus s.

Real time rendszereknél használhatunk:

1. Statikus schedulingot, amely során a döntéseket a rendszer indulása előtt meghozzuk. Ez csak előre ismert ingereket adó rendszereknél használható.
2. Dinamikus scheduling, a döntéseket futási időben hozzuk meg.