Tejgl Edit
PB1MTJ
Programtervező informatikus
2020.12.16

1 oldal

①

E1:240 E2:36 E3:8

	Foglal			Maximális igény		
	E1	E2	E3	E1	E2	E3
P1	53	14	4	67	15	5
P2	0	5	1	13	5	3
P3	46	17	0	107	27	5
P4	127	0	1	132	25	4

a) Igény = Max. igény − foglal

	E1	E2	E3
P1	14	1	1
P2	13	0	2
P3	61	10	5
P4	5	25	3

b) szabad erőforrások: Σ foglalt

E1: 53+46+127 = 226 foglalt
14 szabad

E2: 14+5+17 = 36 foglalt
0 szabad

E3: 4+1+1 = 6 foglalt
2 szabad

Kezdet (14, 0, 2)

c) Végrehajtási sorrend:

P_2 lefut, majd felszabadul: kezdet (14, 5, 3)

P_1 lefut, majd felszabadul: kezdet (67, 19, 7)

P_3 lefut, majd felszabadul: kezdet: (113, 36, 7)

P_4 lefut, majd felszabadul: A rendszer biztonságos, nincs holtpont.

Feigl Edit
PB1HTJ

Feigl Edit
PB1HTJ
PT I
2020.12.16.

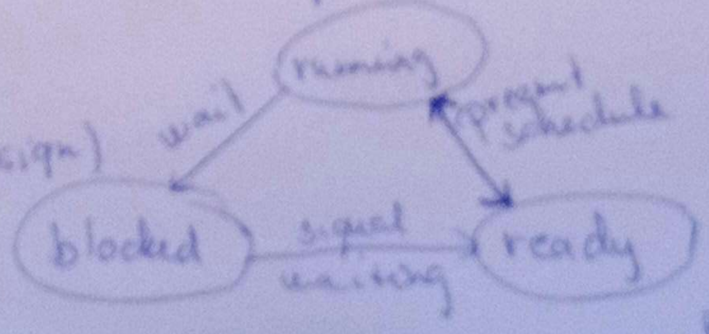
(2)

A processz – születik
– él: verseng erőforrásokért, osztozik erőforrásokon, kommunikál, szinkronizálódik
– exitál (megszűnik)

A processz kreációja: processzt csak processz kreálhat, így kialakul szülő-gyermek kapcsolat, létezik ősszülő is. (pl. init) Rendszerhívások:
↳ fork() – a szülő processz instrukciófolyamát két ágra választja, az egyik marad a szülő első ágra folyama a másik(?) ágra pedig gyermekét processzt kreál. Megnézi, hogy létrehozható-e gyerek, megadja a pid-jét, bejegyzi a PT-be. Lemásolja a szülő kontextusát. (logikai másolat, eltér pl. a pid.)
↳ exec() – az adott processzben futtatható fájl futtatása.

Exitálás:
↳ normális exitálás (terminálódás): a exit() vagy return rendszerhívásokkal vagy ha a main függvény véget ér.
↳ erőltetett exitálás: (más processzből) az abort() vagy kill() rendszerhívásokkal megvalósítható.

Állapot diag.



Állapotátmenetek
wait
signal waiting (assign)
preemt – elvesz }
schedule – kioszt }
ütemező végzi

Running – a processzé a CPU.
Blocked – valamilyen erőforrásra vár (CPU-től különbözőre)
Ready – minden erőforrás megvan, kivéve a CPU-t

3. oldal

Feigl Edit
PBMHT)
PT1
2020.12.16.

(3) Holtpont (deadlock)

Egy rendszer H (~~részhalmaz (holtpont)~~) részhalmaza holtpontban van, ha a H valamennyi eleme olyan erőforrásra vár, amit csak egy másik H részhalmaz beli elemtől kaphatna meg. *

Keletkezése:

A holtpont akkor alakul ki, ha nem megfelelően vagy egyáltalán nem használunk kölcsönös kizárást egy azt igénylő erőforrás esetében. (Minden esetben több kölcsönös kizárást igénylő erőforrásról van szó.)

Kialakulásának feltételei:

- kölcsönös kizárás
- foglalva várakozás
- körkörösség kialakulása
- nincs erőszakos erőforrás elvétel

Megoldási lehetőségei

1. Strucc módszer: figyelmen kívül hagyjuk a problémát (ugyanis elég kicsi az esélye a kialakulásnak. Előnye: 0 költségigény. Azonban, ha kialakul, akkor „lefagy" a gép. Kritikus rendszer esetén nem használható.
2. Detektálás és megszüntetés:
   - detektálás időközönként:
     - gyakran: gyors detektálás, nagy költségigény
     - ritkán: lassú detektálás, kisebb költségigény
   - detektálás (~~folyamatosan~~) események hatására:
     - erőforrás átadásakor (minden esetben): nagy a költségigénye (hibásan is)

4. oldal

③ folytatás:

<u>Megszüntetés</u>:

↳ erőforrás elvétellel: a processz amúgy is maradandó kárt szenved

↳ processz lelövéssel:

- minden p. lelövésével: drasztikus, de hatá
- egyenként lelövéssel és újra detektálással nagy a költségigény

↳ a kiváltó okok valamelyikének megszünte

3. <u>Megelőzéssel</u>: Minden erőforrás átadás előtt dönteni kell, hogy kialakulhat-e holtpont. Ezt valósítja meg a bankár algoritm csak akkor indulhat el egy processz nem veszélyezteti a holtpont kiala sával a rendszert. A foglalt erőforrások igények folyamatos vizsgálatát (minden forrás átadásnál) igényli.

* Holtpontba került processz nem futhat az erőforrásai nem tudnak felszabadu így az arra várakozó processzek is hol kerülnek.

Feigl Edit
PB(MT)
PTI
2020.12.16.

(4) IPC mechanizmusok

Az IPC - Inter Process Communication a processek kommunikációjára szolgáló mechanizmus

Az ~~kommunikáció~~ célja: - információcsere
- szinkronizáció

Maga a kommunikáció is igényelhet szinkronizációt

Kommunikáció fajtái:

- direkt: a kommunikáló processzeknek ismerniük kell egymást. (pl. signal) műveletei: send, receive
- indirekt: a kommunikáló processzeknek egy közvetítő entitást kell ismerniük (pl. üzenetsor, ~~pcb~~ postaláda)
  Az entitás lehet OS kötődésű, process kötődésű, attól függően ki hozza létre és szünteti meg.
  - egyirányú (asszimetrikus): az egyik csak ad, a másik fogad
  - kétirányú (szimmetrikus): mindkettő ad/fogad

- ~~szívó~~ pufferelt (nincs puffer, azonnali fogadást igényel)
  Korlátolt kapacitású és végtelen kapacitású pufferelt
  Ha van puffer az üzenet ide kerül (küldés után) innen olvassa ki az elsőt a fogadó.
  Ha korlátolt: - a küldő csak akkor vár, ha megtelt a puffer
  - a fogadó csak akkor vár, ha üres a puffer
  Ha végtelen: a fogadó csak akkor vár, ha üres a puffer.

Szinkron, aszinkron: ha mind a küldés, mind a fogadás szinkron → szinkron
ha valamelyik vagy mindkettő aszinkron → aszinkron

6. oldal

Feigl Edit
PBIM7J
2020.12.16

(4.)

IPC mechanizmusok:

↳ Fájlokon keresztüli kommunikáció:

- a processzek ugyanabba a fájlba írnak, ugyanabból olvasnak
- lassú, nagy az információátvitel
- indirekt, ~~(…)~~ pufferelt, változó üzenethosszos

↳ Clipboard (Vágólap)

- ctrl + c és ctrl + v
- ctrl + c az egyik processz ír a vágólapra
- ctrl + v a másik kiolvas a vágólapról
- indirekt, változó üzenethosszos, pufferelt
- nem az OS, hanem ált. a GUI biztosítja

↳ Signal:

- információ ~~továbbítására~~ jelzésre használják gyakran az OS rutinjai
- fix üzenethossz, kérő pufferelt, direkt
- ezzel lehet jelezni pl. kivételes eseményt
- kis információátvitel, gyors