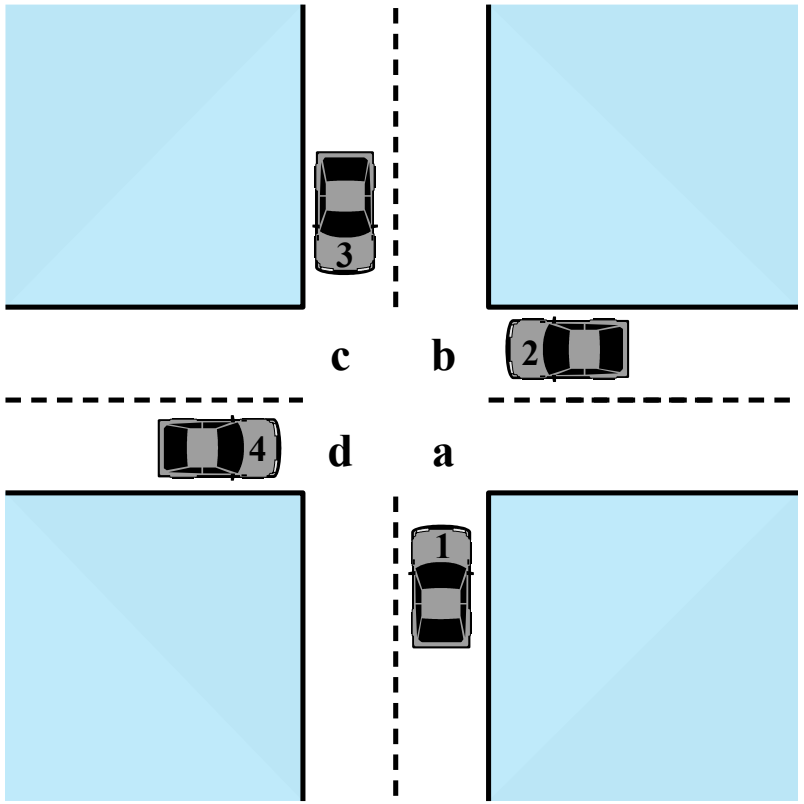
Zastoji

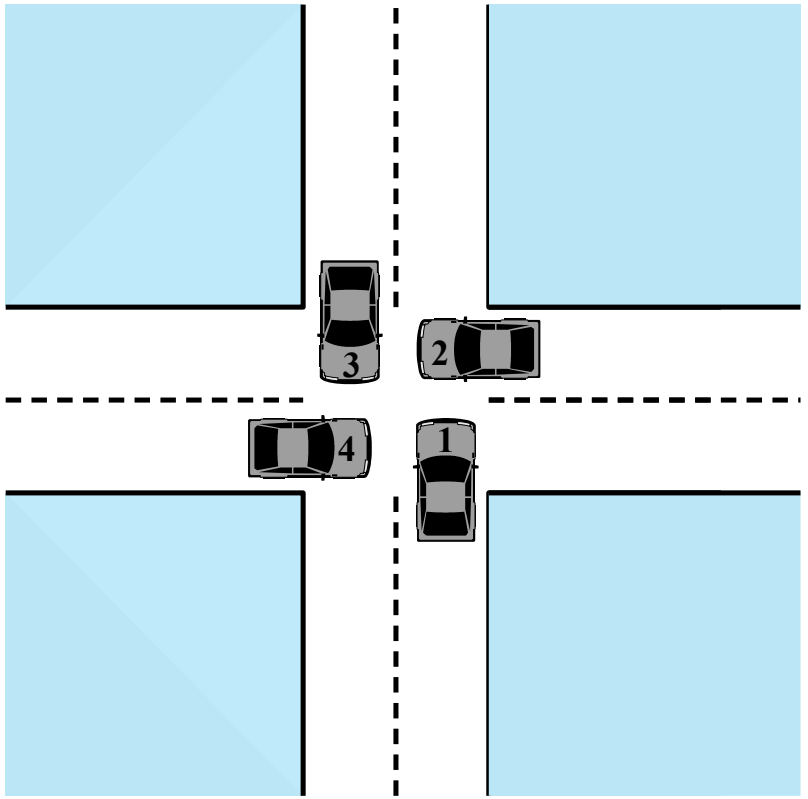
Zastoj

- Trajno blokiranje skupa procesa koji se ili natječu za resurse sustava ili međusobno komuniciraju
- Skup procesa je u zastoju kada je svaki proces u skupu blokiran čekajući događaj koji može pokrenuti samo drugi blokirani proces u skupu
- Trajno jer se nijedan događaj nikada ne pokreće
- Nema učinkovitog rješenja u općem slučaju

Prikaz zastoja

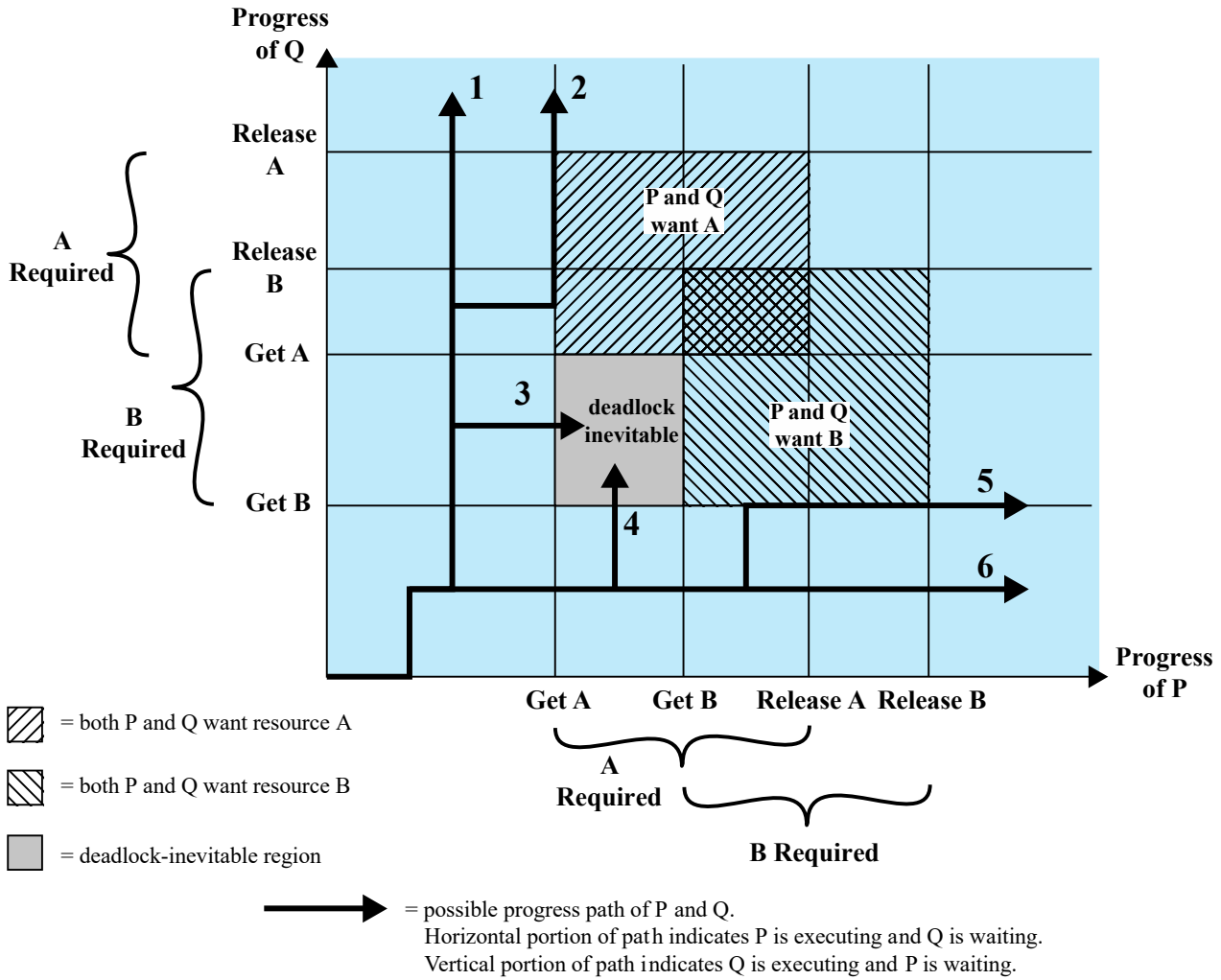


(a) Deadlock possible

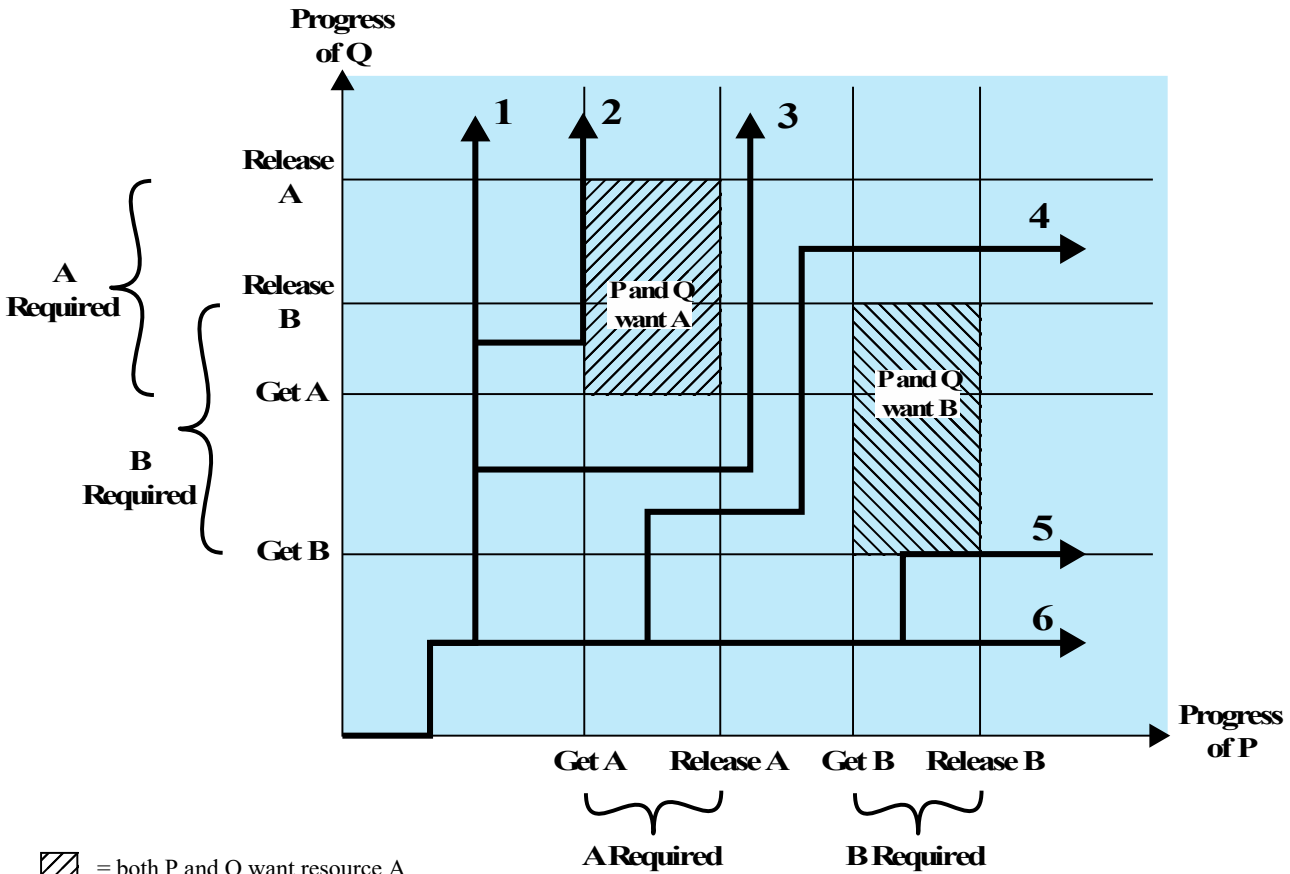


(b) Deadlock

Primjer zastoja



Primjer bez zastoja



Kategorije resursa

Za višekratnu upotrebu

- Može se sigurno koristiti samo jednim procesom u isto vrijeme i ne iscrpljuje se tom upotrebom
  - Procesori, I/O kanali, glavna i sekundarna memorija, uređaji i strukture podataka kao što su datoteke, baze podataka i semafori

Potrošni

- Onaj koji se može stvoriti (proizvesti) i uništiti (konzumirati)
  - Prekidi, signali, poruke i informacije
  - U I/O međuspremnicima

Dva procesa koji se natječu za višekratnu upotrebu resursa

Step	Process P Action
p_0	Request (D)
p_1	Lock (D)
p_2	Request (T)
p_3	Lock (T)
p_4	Perform function
p_5	Unlock (D)
p_6	Unlock (T)

Step	Process Q Action
q_0	Request (T)
q_1	Lock (T)
q_2	Request (D)
q_3	Lock (D)
q_4	Perform function
q_5	Unlock (T)
q_6	Unlock (D)

Primjer 2: Zahtjev za memorijom

- Dostupan je prostor za dodjelu 200Kbajta, a događa se sljedeći slijed događaja:

P1	P2
. . .	. . .
Request 80 Kbytes;	Request 70 Kbytes;
. . .	. . .
Request 60 Kbytes;	Request 80 Kbytes;

- Zastoj se događa ako oba procesa napreduju do svog drugog zahtjeva

Potrošni resursi zastoj

- Razmotrimo par procesa u kojima svaki proces pokušava primiti poruku od drugog procesa, a zatim poslati poruku drugom procesu:

P1	P2
. . .	. . .
Receive (P2);	Receive (P1);
. . .	. . .
Send (P2, M1);	Send (P1, M2);

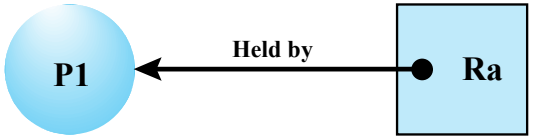
- Zastoj se događa ako je prijem blokiran

Pristupi za zastoje

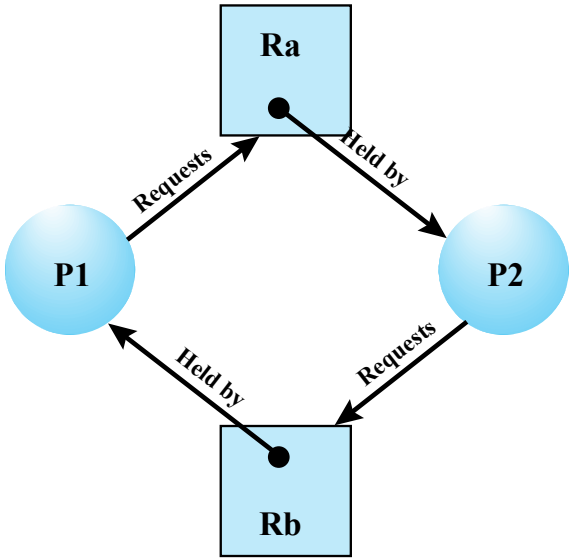
- Ne postoji jedinstvena učinkovita strategija koja se može nositi sa svim vrstama zastoja
- Tri su pristupa uobičajena:
- Prevencija zastoja
  - Onemogućite jedan od tri neophodna uvjeta za pojavu zastoja ili spriječite da se dogodi uvjet kružnog čekanja
- Izbjegavanje zastoja
  - Nemojte odobriti zahtjev za resursima ako ova alokacija može dovesti do zastoja
- Detekcija zastoja
  - Odobravajte zahtjeve za resurse kada je to moguće, ali povremeno provjeravajte prisutnost zastoja i poduzmite mjere za oporavak

Grafikoni raspodjele resursa

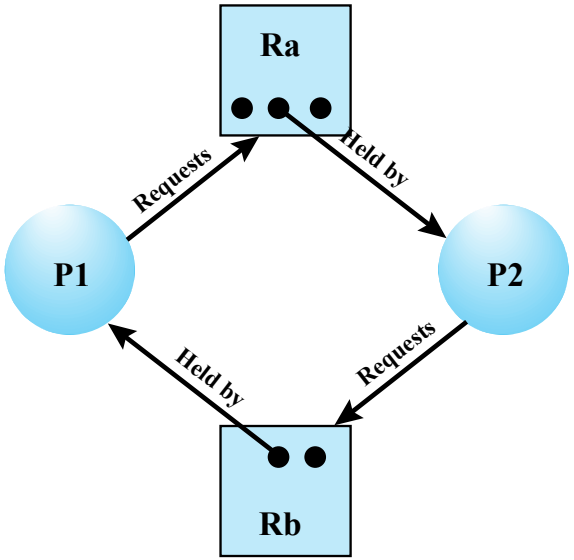
(a) Resouce is requested



(b) Resource is held



(c) Circular wait



(d) No deadlock

Uvjeti za zastoj

Međusobno isključivanje	Zadrži i čekaj	Bez prevencije	Kružno čekanje
• Samo jedan proces može istovremeno koristiti resurs • Nijedan proces ne može pristupiti resursu dok se ne dodijeli procesu	• Proces može zadržati dodijeljene resurse dok čeka na dodjelu drugih resursa	• Nijedan resurs ne može se nasilno ukloniti iz procesa koji ga drži	• Zatvoreni lanac procesa, takav da svaki proces sadrži barem jedan resurs potreban sljedećem procesu u lancu

Strategije prevencije zastoja

- Projektirajte sustav na takav način da je isključena mogućnost zastoja
- Dvije glavne metode:
  - Neizravno
    - Spriječite pojavu jednog od tri neophodna uvjeta
  - Direktno
    - Spriječite pojavu kružnog čekanja

Prevencija stanja zastoja

- Međusobno isključivanje
  - Ako pristup resursu zahtijeva međusobno isključivanje, tada OS mora podržavati uzajamno isključivanje
  - Neki resursi, kao što su datoteke, mogu dopustiti višestruke pristupe za čitanje, ali samo isključivi pristup za pisanje
  - Čak i u ovom slučaju može doći do zastoja ako više od jednog procesa zahtijeva dopuštenje za pisanje
- Zadrži i čekaj
  - Može se spriječiti zahtjevom da proces zahtijeva sve svoje potrebne resurse odjednom i blokiranjem procesa dok se svi zahtjevi ne mogu odobriti istovremeno

Prevencija stanja zastoja

- Bez prevencije
  - Ako je procesu koji ima određene resurse odbijen daljnji zahtjev, taj proces mora osloboditi svoje izvorne resurse i ponovno ih zatražiti
  - OS može spriječiti drugi proces i zahtijevati da oslobodi svoje resurse
- Kružno čekanje
  - Uvjet kružnog čekanja može se spriječiti definiranjem linearnog redoslijeda tipova resursa

Izbjegavanje zastoja

- Omogućuje tri neophodna uvjeta, ali donosi razumne izbore kako bi osigurao da se točka zastoja nikada ne postigne
- Dinamički se donosi odluka hoće li trenutni zahtjev za dodjelu resursa, ako se odobri, potencijalno dovesti do zastoja
- Zahtijeva poznavanje budućih zahtjeva procesa

Dva pristupa izbjegavanju zastoja

Uskraćivanje dodjele resursa

- Naziva se bankarov algoritam
- Stanje sustava odražava trenutnu dodjelu resursa procesima
- Sigurno stanje je ono u kojem postoji barem jedan slijed dodjeljivanja resursa procesima koji ne dovodi do zastoja
- Nesigurno stanje je stanje koje nije sigurno

Određivanje sigurnog stanja

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Claim matrix **C**

	R1	R2	R3
P1	1	0	0
P2	6	1	2
P3	2	1	1
P4	0	0	2

Allocation matrix **A**

	R1	R2	R3
P1	2	2	2
P2	0	0	1
P3	1	0	3
P4	4	2	0

C – A

R1	R2	R3
9	3	6

Resource vector **R**

R1	R2	R3
0	1	1

Available vector **V**

(a) Initial state

Određivanje sigurnog stanja

	R1	R2	R3
P1	3	2	2
P2	0	0	0
P3	3	1	4
P4	4	2	2

Claim matrix **C**

	R1	R2	R3
P1	1	0	0
P2	0	0	0
P3	2	1	1
P4	0	0	2

Allocation matrix **A**

	R1	R2	R3
P1	2	2	2
P2	0	0	0
P3	1	0	3
P4	4	2	0

C – A

R1	R2	R3
9	3	6

Resource vector **R**

R1	R2	R3
6	2	3

Available vector **V**

(b) P2 runs to completion

Određivanje sigurnog stanja

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	3	1	4
P4	4	2	2

Claim matrix **C**

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	2	1	1
P4	0	0	2

Allocation matrix **A**

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	1	0	3
P4	4	2	0

C – A

R1	R2	R3
9	3	6

Resource vector **R**

R1	R2	R3
7	2	3

Available vector **V**

(c) P1 runs to completion

Određivanje sigurnog stanja

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	4	2	2

Claim matrix **C**

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	0	0	2

Allocation matrix **A**

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	4	2	0

C – A

R1	R2	R3
9	3	6

Resource vector **R**

R1	R2	R3
9	3	4

Available vector **V**

Određivanje nesigurnog stanja

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Claim matrix **C**

	R1	R2	R3
P1	1	0	0
P2	5	1	1
P3	2	1	1
P4	0	0	2

Allocation matrix **A**

	R1	R2	R3
P1	2	2	2
P2	1	0	2
P3	1	0	3
P4	4	2	0

C – A

R1	R2	R3
9	3	6

Resource vector **R**

R1	R2	R3
1	1	2

Available vector **V**

(a) Initial state

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Claim matrix **C**

	R1	R2	R3
P1	2	0	1
P2	5	1	1
P3	2	1	1
P4	0	0	2

Allocation matrix **A**

	R1	R2	R3
P1	1	2	1
P2	1	0	2
P3	1	0	3
P4	4	2	0

C – A

R1	R2	R3
9	3	6

Resource vector **R**

R1	R2	R3
0	1	1

Available vector **V**

(b) P1 requests one unit each of R1 and R3

Prednosti izbjegavanja zastoja

- Nije potrebno preduhitriti i vraćati procese unatrag, kao u otkrivanju zastoja
- Manje je restriktivna od prevencije zastoja

Ograničenja izbjegavanja zastoja

- Maksimalni zahtjevi za resursima za svaki proces moraju biti unaprijed navedeni
- Procesi koji se razmatraju moraju biti neovisni i bez zahtjeva za sinkronizaciju
- Mora postojati fiksni broj resursa za dodjelu
- Nijedan proces ne može završiti dok drži resurse

Strategije prevencije zastoja

Strategije prevencije zastoja vrlo su konzervativne

- Ograničite pristup resursima nametanjem ograničenja procesima

Strategije otkrivanja zastoja rade suprotno

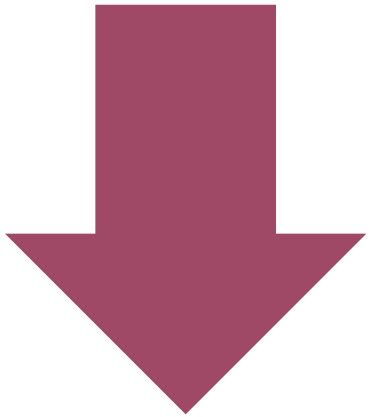
- Zahtjevi za resurse odobravaju se kad god je to moguće

Algoritam za otkrivanje zastoja

- Provjera zastoja može se vršiti jednako često kao i svaki zahtjev za resursom ili, rjeđe, ovisno o tome kolika je vjerojatnost da će doći do zastoja

Prednosti:

- Dovodi do ranog otkrivanja
- Algoritam je relativno jednostavan



Nedostaci

- Česte provjere troše znatno procesorsko vrijeme

Strategije oporavka

- Prekinite sve procese u zastoju
- Sigurnosno kopirajte svaki proces u zastoju na nekoj prethodno definiranoj kontrolnoj točki i ponovno pokrenite sve procese
- Sukcesivno prekidajte procese u zastoju dok zastoj više ne postoji
- Sukcesivno preuzmite resurse dok zastoj više ne postoji

Integrirana strategija zastoja

- Umjesto da pokušavate dizajnirati OS koji koristi samo jednu od ovih strategija, moglo bi biti učinkovitije koristiti različite strategije u različitim situacijama
  - Grupirajte resurse u više različitih klasa resursa
  - Koristite strategiju linearne dodjele prethodno definiranu za sprječavanje kružnog čekanja kako biste spriječili zastoje između klasa resursa
  - Unutar klase resursa koristite algoritam koji je najprikladniji za tu klasu

Klasne strategije

- Unutar svakog razreda mogu se koristiti sljedeće strategije:
  - Izmjenjiv prostor
    - Sprečavanje zastoja zahtjevom da se svi potrebni resursi koji se mogu koristiti budu dodijeljeni odjednom, kao u strategiji prevencije čekaj i čekaj
    - Ova strategija je razumna ako su poznati maksimalni zahtjevi za pohranu
  - Resursi procesa
    - Izbjegavanje će često biti učinkovito u ovoj kategoriji, jer je razumno očekivati da procesi unaprijed deklariraju resurse koji će im biti potrebni u ovoj klasi
    - Moguća je i prevencija putem naručivanja resursa unutar ove klase
  - Glavna memorija
    - Čini se da je prevencija prevencijom najprikladnija strategija za glavnu memoriju
    - Kada je proces preuzet, jednostavno se zamjenjuje u sekundarnu memoriju, oslobađajući prostor za rješavanje zastoja
  - Interni resursi
    - Može se koristiti prevencija putem naručivanja resursa

Problem filozofa

- Niti jedan od filozofa ne mogu koristiti istu vilicu u isto vrijeme (međusobno isključenje)
- Nijedan filozof ne smije umrijeti od gladi (izbjegavajte zastoj i gladovanje)

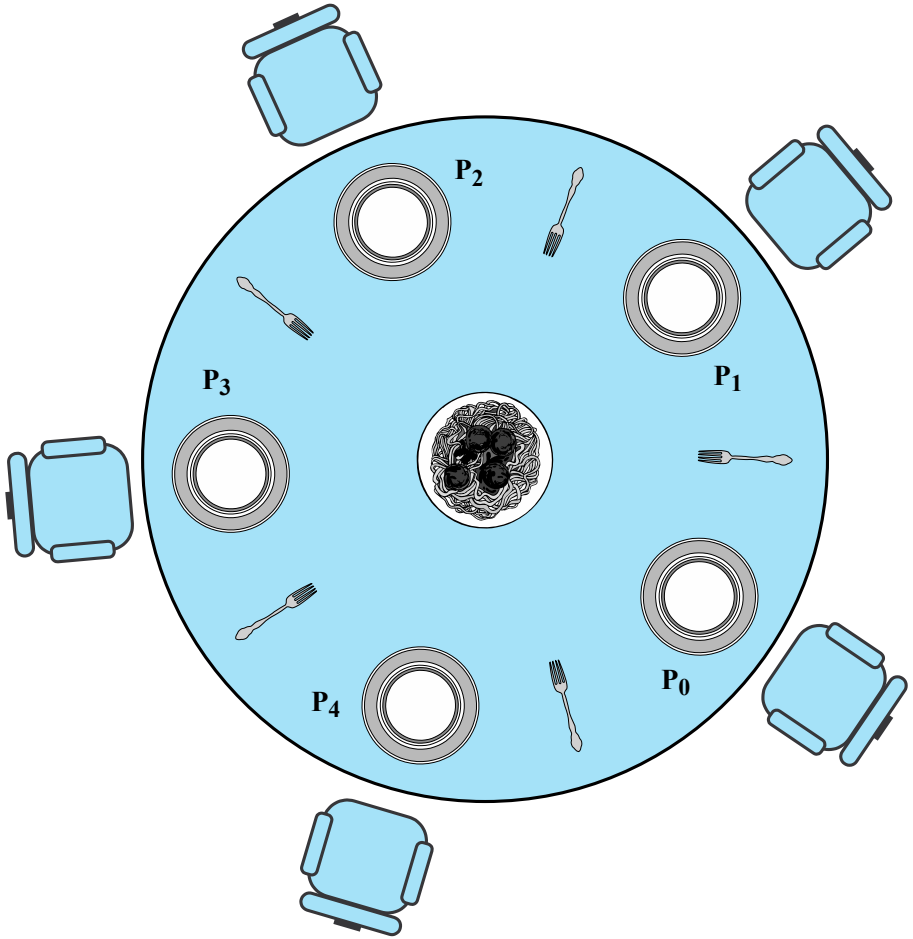


Figure 6.11 Dining Arrangement for Philosophers

Prvo rješenje problema filozofa

```
/* program diningphilosophers */
semaphore fork [5] = {1}; int i;
void philosopher (int i) {
    while (true) {
        think();
        wait (fork[i]);
        wait (fork [(i+1) mod 5]); eat();
        signal(fork [(i+1) mod 5]); signal(fork[i]);
    }
}
void main() {
parbegin (philosopher (0), philosopher (1), philosopher (2),
          philosopher (3), philosopher (4));
}
```

A Second Solution to the Dining Philosophers Problem

```
semaphore fork[5] = {1}; semaphore room = {4}; int i;
void philosopher (int i) {
      while (true) {
      think();
      wait (room);
      wait (fork[i]);
      wait (fork [(i+1) mod 5]); eat();
      signal (fork [(i+1) mod 5]); signal (fork[i]);
      signal (room);
      }
}
void main() {
      parbegin (philosopher (0), philosopher (1), philosopher (2), philosopher (3),
      philosopher (4));
}
```

Rješenje za problem filozofa korištenjem monitora

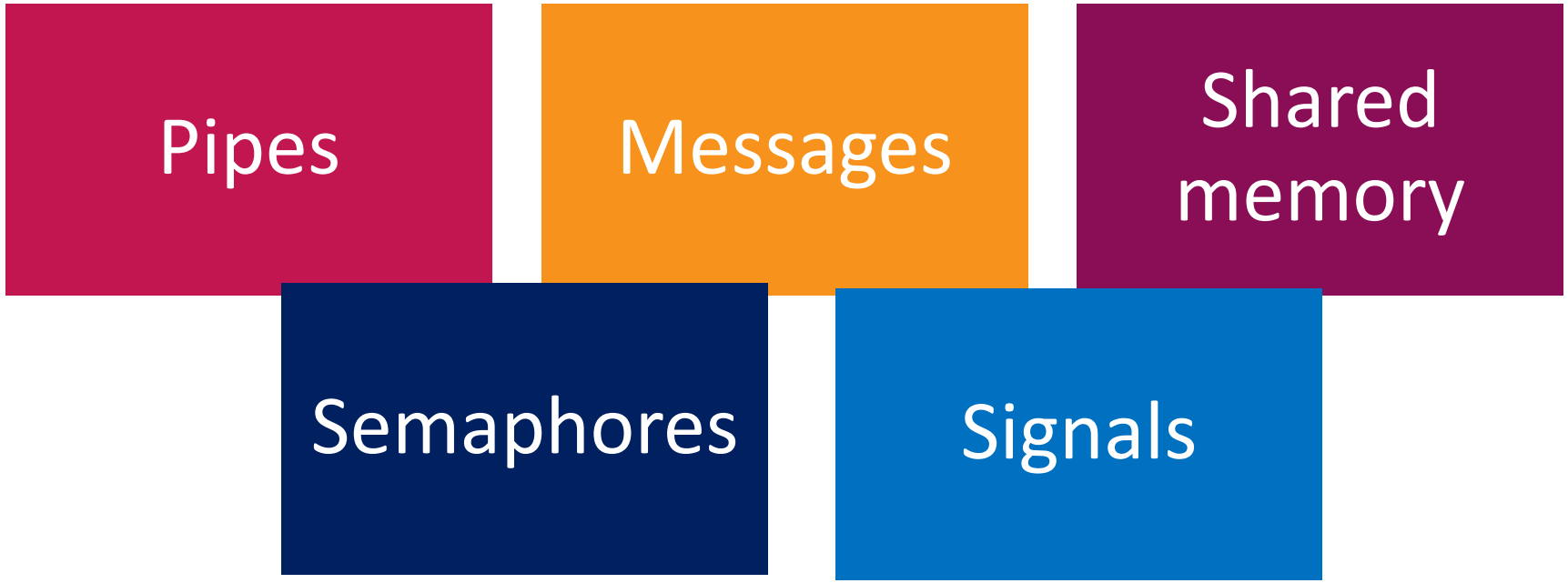
```
monitor dining_controller;
cond ForkReady[5];        /* condition  variable  for  synchronization  */
boolean fork[5] = {true}; /* availability status of each fork */
void get_forks(int pid) { /* pid is the philosopher id number */
      int left = pid; int right = (++pid) % 5;
      /*grant the left fork*/
      if (!fork[left]) cwait(ForkReady[left]); /* queue on condition variable */ fork(left) = false;
      /*grant the right fork*/
      if (!fork[right]) cwait(ForkReady[right]); /* queue on condition variable */ fork[right] = false:
}
void release_forks(int pid) {
      int left = pid; int right = (++pid) % 5;
      /*release the left fork*/
      if (empty(ForkReady[left]) /*no one is waiting for this fork */ fork[left] = true;
      else   /* awaken a process waiting on this fork */ csignal(ForkReady[left]);
      /*release the right fork*/
      if (empty(ForkReady[right])/*no one is waiting for this fork */ fork[right] = true;
      else   /* awaken a process waiting on this fork */ csignal(ForkReady[right]);
}

void philosopher[k=0 to 4] {/* the five philosopher clients */
      while (true) {
            <think>;
            get_forks(k); /* client requests two forks via monitor */
            <eat spaghetti>;
            release_forks(k); /* client releases forks via the monitor */
      }

}
```

UNIX mehanizmi istodobnosti

- UNIX pruža niz mehanizama za međuprocesorsku komunikaciju i sinkronizaciju uključujući:



Cijevi (Pipes)

- Kružni međuspremnici koji omogućuju komunikaciju dvaju procesa na modelu proizvođač-potrošač
  - Red "prvi ušao-prvi izašao (FIFO)", zapisan od strane jednog procesa i čitan od strane drugog

Dva ripa:

- Imenovani
- Neimenovani

Poruke

- Blok bajtova s pripadajućim tipom
- UNIX pruža msgsnd i msgrcv sistemske pozive za procese da se uključe u prosljeđivanje poruka
- Uz svaki proces povezan je red poruka, koji funkcionira kao poštanski sandučić

Zajednička memorija

- Najbrži oblik međuprocesne komunikacije
- Zajednički blok virtualne memorije koju dijeli više procesa
- Dopuštenje je samo za čitanje ili za čitanje-pisanje za proces
- Ograničenja međusobnog isključivanja nisu dio mogućnosti dijeljene memorije, ali ih moraju osigurati procesi koji koriste dijeljenu memoriju

Semafori

- Generalizacija primitiva semWait i semSignal
  - Niti jedan drugi proces ne može pristupiti semaforu dok se sve operacije ne dovrše

Sastoji se od:
• Trenutna vrijednost semafora • ID procesa posljednjeg procesa koji radi na semaforu • Broj procesa koji čekaju da vrijednost semafora bude veća od trenutne vrijednosti • Broj procesa koji čekaju da vrijednost semafora bude nula

Signali

- Softverski mehanizam koji obavještava proces o nastanku asinkronih događaja
  - Slično hardverskom prekidu, ali ne koristi prioritete
- Signal se isporučuje ažuriranjem polja u tablici procesa za proces kojem se signal šalje
- Proces može odgovoriti na signal:
  - Izvođenje neke zadane radnje
  - Izvršavanje funkcije rukovatelja signalom
  - Ignoriranje signala

Signali u stvarnom vremenu (RT).

- Linux uključuje sve mehanizme konkurentnosti koji se nalaze u drugim UNIX sustavima
- Linux također podržava signale u stvarnom vremenu (RT).
- RT signali se razlikuju od standardnih UNIX signala na tri osnovna načina:
  - Podržana je isporuka signala prioritetnim redoslijedom
  - Više signala se može staviti u red čekanja
  - Sa standardnim signalima, nikakva vrijednost ili poruka ne mogu se poslati ciljnom procesu – samo je obavijest s RT signalima moguće poslati vrijednost zajedno sa signalom

Atomarne operacije

- Atomarne operacije se izvode bez prekida i smetnji
- Najjednostavniji pristup sinkronizaciji kernela
- Dvije vrste:

Cjelobrojne operacije

Rad s cjelobrojnom varijablom

Obično se koristi za implementaciju brojača

Bitmap operacije

Radi na jednom od niza bitova na proizvoljnom memorijskom mjestu označenom varijablom pokazivača

Spinlocks

- Najčešća tehnika za zaštitu kritičnog odjeljka u Linuxu
- Može se nabaviti samo jednom niti u isto vrijeme
  - Bilo koja druga nit će nastaviti pokušavati (vrtjeti) sve dok ne postigne zaključavanje
- Izgrađen na cjelobrojnoj lokaciji u memoriji koju provjerava svaka nit prije nego što uđe u svoj kritični dio
- Učinkovito u situacijama u kojima se očekuje da će vrijeme čekanja za preuzimanje brave biti vrlo kratko
  - Zastoj:
  - Zaključa dretve koje traže pristup resursu i nastavlja se izvršavati

Semafori

- Korisnička razina:
  - Linux pruža semaforsko sučelje koje odgovara onom u UNIX SVR4
- Interno:
  - Implementirane kao funkcije unutar kernela i učinkovitije su od semafora vidljivih korisniku
- Tri vrste semafora kernela:
  - Binarni semafori
  - Brojanje semafora
  - Semafori čitač-pisač

Čitanje-kopiranje-ažuriranje (RCU)

- RCU mehanizam je napredni lagani mehanizam za sinkronizaciju koji je integriran u jezgru Linuxa 2002.
- RCU se široko koristi u Linux kernelu
- RCU također koriste drugi operativni sustavi
- Postoji RCU biblioteka korisničkog prostora koja se zove liburcu
- Zajedničkim resursima koje štiti mehanizam RCU mora se pristupiti preko pokazivača
- Mehanizam RCU omogućuje pristup za više čitatelja i pisaca zajedničkom izvoru

Mehanizmi istodobnosti u Windows OS-u

- Windows pruža sinkronizaciju među nitima kao dio arhitekture objekta

Najvažnije metode su:

- Izvršni dispečer objekti
- Kritični odjeljci korisničkog načina rada
- Tanke brave za čitanje i pisanje
- Varijable uvjeta
- Radnje bez zaključavanja

Kunkcija čekaj

Dozvoljava dretvi da se sama zaustavi

Nema vraćanja dok se ne ispune kriteriji

Vrsta funkcije čekanja određuje skup kriterija

Kritični odsječak

- Sličan mehanizam kao mutex, osim što kritične sekcije mogu koristiti samo niti jednog procesa
- Ako je sustav višeprocesorski, kod će pokušati postići spin-lock
  - Kao posljednje sredstvo, ako se spinlock ne može dobiti, objekt dispečera se koristi za blokiranje niti tako da kernel može poslati drugu nit na procesor

Tanke brave za čitanje i pisanje

- Windows Vista dodao je čitač-pisač u korisničkom načinu rada
- Zaključavanje čitač-pisač ulazi u kernel radi blokiranja tek nakon pokušaja korištenja spin-locka
- Tanak je u smislu da obično zahtijeva samo dodjelu jednog dijela memorije veličine pokazivača

Varijable uvjeta

- Windows također ima varijable uvjeta
- Proces mora deklarirati i inicijalizirati CONDITION_VARIABLE
- Koristi se s kritičnim dijelovima ili SRW bravama
- Koristi se na sljedeći način:
  - acquire exclusive lock
  - while (predicate()==FALSE)SleepConditionVariable()
  - perform the protected operation
  - release the lock

Sinkronizacija bez zaključavanja

- Windows se također uvelike oslanja na međusobno zaključane operacije za sinkronizaciju
- Isprepletene operacije koriste hardverske mogućnosti kako bi se jamčilo da se memorijske lokacije mogu čitati, mijenjati i zapisivati u jednoj atomskoj operaciji

"Bez zaključavanja"

- Sinkronizacija bez softverskog zaključavanja
- Dretva se nikada ne može isključiti iz procesora dok je resurs zaključan

Android međuprocesna komunikacija

- Android dodaje kernelu novu mogućnost poznatu kao Binder
  - Binder pruža mogućnost laganog poziva udaljenog postupka (RPC) koja je učinkovita u smislu zahtjeva za memorijom i procesiranjem
  - Također se koristi za posredovanje u svim interakcijama između dva procesa
  - RPC mehanizam radi između dva procesa na istom sustavu, ali radi na različitim virtualnim strojevima
  - Metoda koja se koristi za komunikaciju s Binderom je ioctl sistemski poziv
  - Poziv ioctl je sistemski poziv opće namjene za I/O operacije specifične za uređaj

Sažetak

- Principi zastoja
  - Resursi koji se mogu ponovno koristiti/potrošni
  - Grafovi raspodjele resursa
  - Uvjeti za zastoj
- Prevencija zastoja
  - Međusobno isključivanje
  - Sačekaj i čekaj
  - Bez prevencije
  - Kružno čekanje
- Izbjegavanje zastoja
  - Odbijanje pokretanja procesa
  - Uskraćivanje dodjele resursa
- Detekcija zastoja
  - Algoritam detekcije zastoja
  - Oporavak
- Integrirana strategija zastoja
- UNIX mehanizmi istodobnosti
  - Cijevi
  - Poruke
  - Zajednička memorija
  - Semafori
  - Signali
- Mehanizmi istodobnosti jezgre Linuxa
  - Atomske operacije
  - Spinlocks
  - Semafori
  - Barijere
- Mehanizmi istodobnosti sustava Windows
  - Funkcije čekanja
  - Dispečerski objekti
  - Kritični dijelovi
  - Tanke brave za čitanje i pisanje
  - Sinkronizacija bez zaključavanja
- Android međuprocesna komunikacija

Thank you for your attention!