Međusobno isključivanje i sinkronizacija

Višestruki procesi

- Dizajn operacijskog sustava bavi se upravljanjem procesima i dretvama:
  - Multiprogramiranje
    - Upravljanje višestrukim procesima unutar jednoprocesorskog sustava
  - Multiprocesiranje
    - Upravljanje višestrukim procesima unutar višeprocesora
  - Distribuirana obrada
    - Upravljanje višestrukim procesima koji se izvode na više, distribuiranih računalnih sustava
    - Nedavna proliferacija klastera izvrstan je primjer ovog tipa sustava

Istodobnost nastaje u tri različita konteksta:

Višestruke aplikacije

Nastao kako bi omogućio dijeljenje vremena obrade među aktivnim aplikacijama

Strukturirane aplikacije

Proširenje modularnog dizajna i strukturiranog programiranja

Struktura operaciskog sustava

OS implementiran kao skup procesa ili dretvi

Ključni pojmovi vezani uz istodobnost

Atomska operacija	Funkcija ili radnja implementirana kao slijed jedne ili više instrukcija koja se čini nedjeljiva; to jest, nijedan drugi proces ne može vidjeti međustanje ili prekinuti operaciju. Zajamčeno je da će se slijed instrukcija izvršiti kao grupa, ili se uopće neće izvršiti, bez vidljivog učinka na stanje sustava. Atomičnost jamči izolaciju od istodobnih procesa.
Kritični odjeljak	Dio koda unutar procesa koji zahtijeva pristup zajedničkim resursima i koji se ne smije izvršiti dok drugi proces pristupa istom resursu.
Zastoj (Deadlock)	Situacija u kojoj dva ili više procesa ne mogu nastaviti jer svaki čeka da jedan od ostalih nešto učini.
Zastoj (Livelock)	Situacija u kojoj dva ili više procesa kontinuirano mijenjaju svoja stanja kao odgovor na promjene u drugim procesima bez obavljanja korisnog rada.
Uzajamno isključivanje	Kada je jedan proces u kritičnom dijelu i pristupa dijeljenim resursima, nijedan drugi proces ne smije biti u dijelu koda koji pristupa bilo kojem od tih zajedničkih resursa.
Utrkivanje	Situacija u kojoj više dretvi ili procesa čitaju i pišu isti podatak, a konačni rezultat ovisi o relativnom vremenu njihovog izvršenja.
Izgladnjivanje	Situacija u kojoj modul za raspodjelu procesorskog vremena duže vremena ne dodjeljuje vrijeme procesu koji je spreman i može se pokrenuti.

Međusobno isključenje - Software

- Softverski pristupi može se implementirati za istodobne procese koji se izvode na jednom ili višeprocesorskom računalu sa zajedničkom glavnom memorijom
- Ovaj pristup obično pretpostavljaju elementarno međusobno isključivanje na razini pristupa memoriji
  - Simultani pristupi (čitanje i/ili pisanje) istoj lokaciji u glavnoj memoriji serijalizira neka vrsta memorijskog arbitra, iako redoslijed dodjele pristupa nije preciziran unaprijed
  - Ne pretpostavlja se nikakva podrška u hardveru, operativnom sustavu ili programskom jeziku
- Prve algoritmi za međusobno isključivanje za dva procesa dizajnirao je nizozemski matematičar Dekker
  - Kroz razvoj algoritma možemo uočiti greške na koje treba paziti

Međusobno isključenje – 1. pokušaj

```
/* PROCESS 0 */
.
.
while (turn != 0)
       /* do nothing */;
/* critical section*/;
turn = 1;
.
.
```

```
/* PROCESS 1 */
.
.
while (turn != 1)
       /* do nothing */;
/* critical section*/;
turn = 0;
.
.
```

Međusobno isključenje – 2. pokušaj

```
/* PROCESS 0 */
.
.
while (flag[1])
       /* do nothing */;
flag[0] = true;
/* critical section*/;
flag[0] = false;
.
.
```

```
/* PROCESS 1 */
.
.
while (flag[0])
       /* do nothing */;
flag[1] = true;
/* critical section*/;
flag[1] = false;
.
.
```

Međusobno isključenje – 3. pokušaj

```
/* PROCESS 0 */
.
.
flag[0] = true;
while (flag[1])
       /* do nothing */;
/* critical section*/;
flag[0] = false;
.
.
```

```
/* PROCESS 1 */
.
.
flag[1] = true;
while (flag[0])
       /* do nothing */;
/* critical section*/;
flag[1] = false;
.
.
```

Međusobno isključenje – 4. pokušaj

```
/* PROCESS 0 */
.
.
flag[0] = true;
while (flag[1]) {
       flag[0] = false;
       /* delay */;
       flag[0] = true;
}
/* critical section*/;
flag[0] = false;
.
.
```

```
/* PROCESS 0 */
.
.
flag[1] = true;
while (flag[0]) {
       flag[1] = false;
       /* delay */;
       flag[1] = true;
}
/* critical section*/;
flag[1] = false;
.
.
```

Dekkerov algoritam

```
/* PROCESS 0 */
.
flag[0] = false;
flag[1] = false;
turn = 1;
while (true) {
	flag[0] = true;
	while (flag[1]) {
		if (turn == 1) {
			flag[0] = false;
			while (turn == 1) /* do nothing */;
			flag[0] = true;
		}
	}
	/* critical section*/;
	turn == 1
	flag[0] = false;
	/* remainder */;
}
```

```
/* PROCESS 1 */
.
flag[0] = false;
flag[1] = false;
turn = 1;
while (true) {
	flag[1] = true;
	while (flag[0]) {
		if (turn == 0) {
			flag[1] = false;
			while (turn == 0) /* do nothing */;
			flag[1] = true;
		}
	}
	/* critical section*/;
	turn == 0
	flag[1] = false;
	/* remainder */;
}
```

Petersonov algoritam za dva procesa

```
/* PROCESS 0 */
.
flag[0] = false;
flag[1] = false;
while (true) {
     flag[0] = true;
     turn = 1;
     while (flag [1] && turn == 1) /* do nothing */;
     /* critical section */;
     flag [0] = false;
     /* remainder */;}
}
```

```
/* PROCESS 1 */
.
flag[0] = false;
flag[1] = false;
while (true) {
     flag[1] = true;
     turn = 0;
     while (flag [0] && turn == 0) /* do nothing */;
     /* critical section */;
     flag [1] = false;
     /* remainder */;}
}
```

Načela konkurentnosti

- Preplitanje i preklapanje
  - Mogu se promatrati kao primjeri istodobne obrade
  - Obje predstavljaju iste probleme
- Jednoprocesor – ne može se predvidjeti relativna brzina izvođenja procesa
  - Ovisi o aktivnostima drugih procesa
  - Način na koji OS rješava prekide
  - Algoritmi dodjele procesorskog vremena

Poteškoće istodobnosti

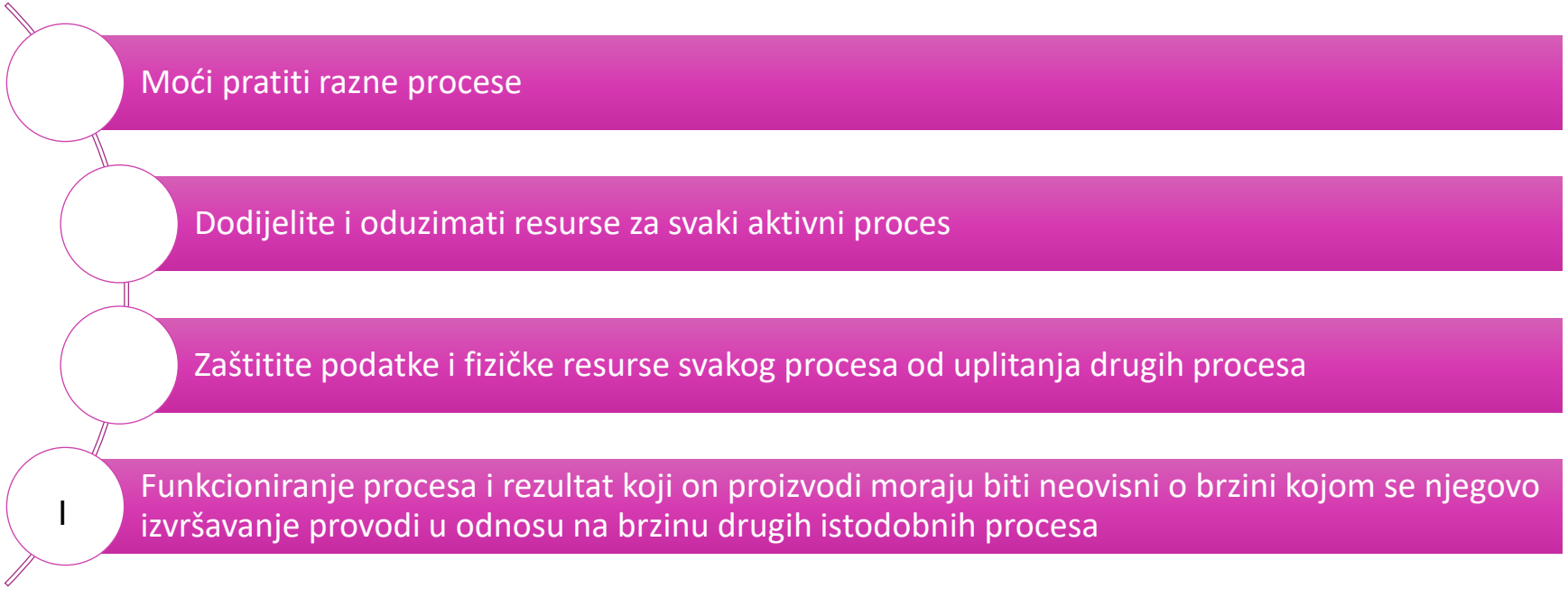
- Dijeljenje globalnih resursa
- OS-u je teško optimalno upravljati dodjelom resursa
- Teško je locirati programske pogreške jer rezultati nisu deterministički i ponovljivi

Utrkivanje

- Pojavljuje se kada više procesa ili dretvi čitaju i pišu podatak na istu lokaciju
- Konačni rezultat ovisi o redoslijedu izvršenja
  - “Dobitnik” utrke je proces koji se posljednji ažurira i koji će odrediti konačnu vrijednost varijable

Izazovi operaciskog sustava

- Problemi dizajna i upravljanja izazvani postojanjem istodobnosti:
  - OS mora:



Interakcija procesa

Stupanj svjesnosti	Odnos	Utjecaj koji jedan proces ima na drugi	Potencijalni problemi s kontrolom
Procesi koji nisu svjesni jedni drugih	Natjecanje	• Rezultati jednog procesa neovisni o djelovanju drugih • Može utjecati na vrijeme procesa	• Međusobno isključivanje • Zastoj • Izgladnjivanje
Procesi koji su neizravno svjesni jedni drugih (npr. zajednički objekt)	Suradnja dijeljenjem	• Rezultati jednog procesa mogu ovisiti o informacijama dobivenim od drugih • Može utjecati na vrijeme procesa	• Međusobno isključivanje • Zastoj • Izgladnjivanje • Koherentnost podataka
Procesi izravno svjesni jedni drugih (imaju na raspolaganju komunikacijske kanale)	Suradnja komunikacijom	• Rezultati jednog procesa mogu ovisiti o informacijama dobivenim od drugih • Može utjecati na vrijeme procesa	• Zastoj • Izgladnjivanje

Natjecanje za resurse

- Istodobni procesi dolaze u sukob kada se natječu za korištenje istog resursa
  - Na primjer: I/O uređaji, memorija, vrijeme procesora, sat

U slučaju konkurentskih procesa moraju se suočiti s tri problema kontrole:

- **Potreba za međusobnim isključivanjem**
- **Zastoj**
- **Izgladnjivanje**

Međusobno isključivanje

```
/* PROCESS 1 */
void P1 {
     while (true) {
          /* preceding code */;
          entercritical (Ra);
          /* critical section */;
          exitcritical (Ra);
          /* following code */;
     }
}
```

```
/* PROCESS 2 */
void P2 {
     while (true) {
          /* preceding code */;
          entercritical (Ra);
          /* critical section */;
          exitcritical (Ra);
          /* following code */;
     }
}
```

...

```
/* PROCESS n */
void Pn {
     while (true) {
          /* preceding code */;
          entercritical (Ra);
          /* critical section */;
          exitcritical (Ra);
          /* following code */;
     }
}
```

Suradnja među procesima dijeljenjem

Obuhvaća procese koji su u interakciji s drugim procesima, a da nisu eksplicitno svjesni toga

Procesi mogu koristiti i ažurirati dijeljene podatke bez upućivanja na druge procese, ali znaju da drugi procesi mogu imati pristup istim podacima

Procesi moraju surađivati kako bi osigurali da se podacima koje dijele pravilno upravlja

Kontrolni mehanizmi moraju osigurati integritet zajedničkih podataka

Budući da se podaci drže na resursima (uređajima, memoriji), opet su prisutni kontrolni problemi međusobnog isključivanja, zastoja i gladovanja

Jedina je razlika u tome što se podacima može pristupiti na dva različita načina, čitanje i pisanje, a samo operacije pisanja moraju se međusobno isključivati

Suradnja među procesima putem komunikacije

- Različiti procesi sudjeluju u zajedničkom naporu koji povezuje sve procese
- Komunikacija pruža način za sinkronizaciju ili koordinaciju različitih aktivnosti
- Obično se komunikacija može okarakterizirati kao da se sastoji od poruka
- Mehanizmi za slanje i primanje poruka mogu biti osigurani kao dio programskog jezika ili osigurani od strane jezgre OS-a
- Međusobno isključenje nije uvjet kontrole
- Problemi zastoja i izgladnjivanja i dalje su prisutni

Zahtjevi za međusobno isključivanje

- Svaki objekt ili sposobnost koja treba pružiti potporu za međusobno isključivanje treba ispunjavati sljedeće zahtjeve:
  - Međusobno isključivanje mora se provoditi na način da je samo jednom proces dopušten ulazak u kritični odsječak među svim procesima koji imaju kritične odsječke koji pristupaju istom resursu ili zajedničkom objektu
  - Proces koji se zablokira mora to učiniti bez uplitanja u druge procese
  - Ne smije biti moguće da se proces koji zahtijeva pristup kritičnom dijelu odgađa na neodređeno vrijeme: nema zastoja ili izgladnjivanja
  - Kada nijedan proces nije u kritičnom dijelu, svakom procesu koji zahtijeva ulazak u svoj kritični dio mora se dopustiti da uđe bez odlaganja
  - Nisu napravljene nikakve pretpostavke o relativnim brzinama procesa ili broju procesa
  - Proces ostaje unutar svog kritičnog dijela samo određeno vrijeme

Međusobno isključenje - Hardware

Onemogućavanje prekida

- U jednoprocesorskom sustavu, istodobni procesi ne mogu biti preklapajući; mogu se samo ispreplitati
- Proces će se nastaviti izvoditi sve dok ne pozove neku uslugu OS-a ili dok se ne prekine
- Stoga, da bi se zajamčilo međusobno isključivanje, dovoljno je spriječiti prekid procesa
- Ova se mogućnost može pružiti u obliku definiranih od strane OS kernela za onemogućavanje i omogućavanje prekida

Nedostaci:

- Učinkovitost izvršenja mogla bi biti osjetno smanjena jer je procesor ograničen u svojoj sposobnosti preplitanja procesa
- Ovaj pristup neće raditi u višeprocesorskoj arhitekturi

Međusobno isključenje - Hardware

- Usporedi i zamijeni instrukcije
  - Naziva se i "compare and exchange instruction"
  - Usporedba se vrši između memorijske vrijednosti i testne vrijednosti
  - Ako su vrijednosti iste dolazi do zamjene
  - Izvodi se atomski (ne podliježe prekidima)

Podrška za međusobno isključivanje

```
/* program mutualexclusion - use of this instruction */
const int n = /* number of processes */;
int bolt;
void P(int i) {
     while (true) {
          while (compare_and_swap(bolt, 0, 1) == 1)
               /* do nothing */;
          /* critical section */;
          bolt = 0;
          /* remainder */;
     }
}
void main() {
     bolt = 0;
     parbegin (P(1), P(2), ... ,P(n));
}
```

```
/* program mutualexclusion - exchange instruction */
int const n = /* number of processes */;
int bolt;
void P(int i) {
     while (true)
          int keyi = 1;
          do exchange (&keyi, &bolt)
          while (keyi != 0);
               /* critical section */;
               bolt = 0;
               /* remainder */;
     }
}
void main() {
     bolt = 0;
     parbegin (P(1), P(2), ..., P(n));
}
```

Posebne strojne instrukcije: Prednosti

- Primjenjivo na bilo koji broj procesa na jednom ili više procesora koji dijele glavnu memoriju
- Jednostavno i lako za provjeru
- Može se koristiti za podršku više kritičnih sekcija; svaki kritični dio može se definirati svojom varijablom

Posebne strojne instrukcije: Nedostaci

- Čekanje
  - Stoga, dok proces čeka na pristup kritičnom dijelu, nastavlja trošiti procesorsko vrijeme
- Izgladnjivanje je moguće
  - Kada proces napusti kritični dio i više od jednog procesa čeka, odabir procesa čekanja je proizvoljan; nekom procesu može biti odbijen pristup na duže vrijeme
- Zastoj je moguć

Uobičajeni mehanizmi istodobnosti

Semafor	Cjelobrojna vrijednost koja se koristi za signalizaciju među procesima. Na semaforu se mogu izvesti samo tri operacije, a sve su atomske: inicijaliziranje, smanjivanje i povećanje. Operacija smanjivanja može rezultirati blokiranjem procesa, a operacija povećanja može rezultirati deblokiranjem procesa. Također poznat kao semafor za brojanje ili opći semafor.
Binarni semafor	Semafor koji poprima samo vrijednosti 0 i 1.
Mutex	Slično binarnom semaforu. Ključna razlika između njih je ta da proces koji zaključava mutex (postavlja vrijednost na 0) mora biti onaj koji će ga otključati (postavlja vrijednost na 1).
Varijabla uvjeta	Vrsta podataka koja se koristi za blokiranje procesa ili dretvii dok se određeni uvjet ne ispuni.
Monitor	Konstrukcija programskog jezika koja enkapsulira varijable, procedure pristupa i inicijalizacijski kod unutar apstraktnog tipa podataka. Varijabli monitora može se pristupiti samo putem njegovih pristupnih procedura i samo jedan proces može aktivno pristupati monitoru u bilo kojem trenutku. Postupci pristupa su kritični dijelovi. Monitor može imati red procesa koji čekaju da mu pristupe.
Zastavice događaja	Memorijska riječ koja se koristi kao mehanizam za sinkronizaciju. Aplikacijski kod može svakom bitu u zastavi pridružiti drugačiji događaj. Dretva može čekati ili jedan događaj ili kombinaciju događaja provjeravanjem jednog ili više bitova u odgovarajućoj zastavici. Dretva je blokirana dok se ne postave svi potrebni bitovi (AND) ili dok se ne postavi barem jedan od bitova (ILI).
Poruke	Sredstva za dva procesa za razmjenu informacija i koja se mogu koristiti za sinkronizaciju.
Spinlocks	Mehanizam međusobnog isključivanja u kojem se proces izvršava u beskonačnoj petlji čekajući vrijednost varijable zaključavanja da ukaže na dostupnost.

Semafor

1. Semafor se može inicijalizirati na nenegativnu cjelobrojnu vrijednost
2. Operacija semWait smanjuje vrijednost semafora
3. Operacija semSignal povećava vrijednost semafora

Varijabla koja ima cjelobrojnu vrijednost na kojoj su definirane samo tri operacije:

- Ne postoji način da se pregleda ili manipulira semaforima osim ove tri operacije

Posljedice

Ne postoji način da se zna prije nego što proces smanji semafor hoće li blokirati ili ne

Ne postoji način da se zna koji će se proces odmah nastaviti na jednoprocesorskom sustavu kada se dva procesa izvode istovremeno

Ne znate čeka li drugi proces pa broj deblokiranih procesa može biti nula ili jedan

Definicija semafora

```
struct semaphore {
     int count;
     queueType queue;
};
void semWait(semaphore s) {
     s.count--;
     if (s.count < 0) {
          /* place this process in s.queue */;
          /* block this process */;
     }
}
void semSignal(semaphore s) {
     s.count++;
     if (s.count<= 0) {
          /* remove a process P from s.queue */;
          /* place process P on ready list */;
     }
}
```

Definicija binarnog semafora

```
struct binary_semaphore {
     enum {zero, one} value;
     queueType queue;
};
void semWaitB(binary_semaphore s) {
     if (s.value == one)
          s.value = zero;
     else {
          /* place this process in s.queue */;
          /* block this process */;
     }
}
void semSignalB(semaphore s) {
     if (s.queue is empty())
          s.value = one;
     else {
          /* remove a process P from s.queue */;
          /* place process P on ready list */;
     }
}
```

Jaki/slabi semafori

- Red se koristi za držanje procesa koji čekaju na semaforu

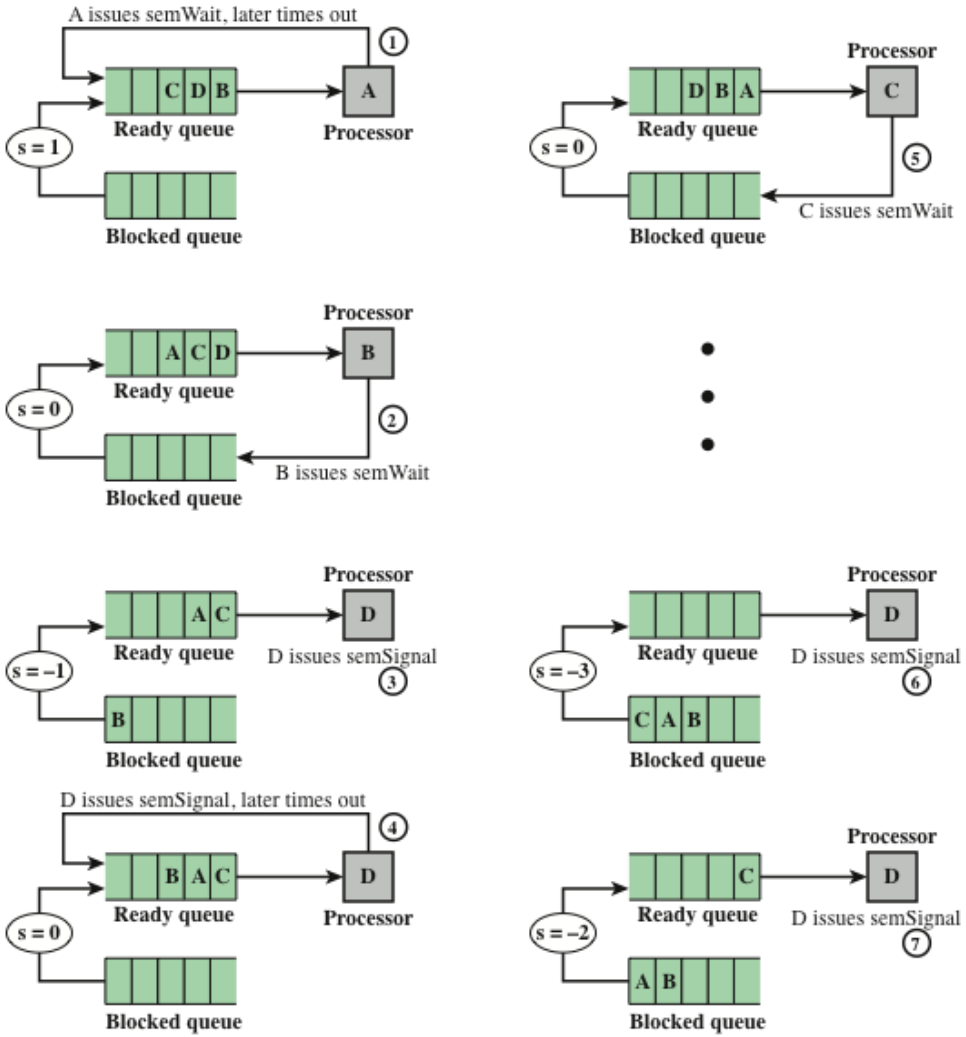
Jaki semafori

- Proces koji je najduže blokiran prvi se oslobađa iz reda čekanja (FIFO)

Slabi semafori

- Nije naveden redoslijed kojim se procesi uklanjaju iz reda čekanja

Semaphore Mechanism



Međusobno isključivanje korištenjem semafora

```
/* program mutualexclusion */
const int n = /* number of processes */;
semaphore s = 1;
void P(int i) {
     while (true) {
          semWait(s);
          /* critical section */;
          semSignal(s);
          /* remainder */;
     }
}
void main() {
     parbegin (P(1), P(2), . . . , P(n));
}
```

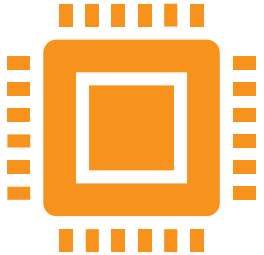
Problem proizvođača/potrošača

Osnovna poruka:

Jedan ili više proizvođača generiraju podatke i stavljaju ih u međuspremnik

Jedan potrošač vadi podatke iz međuspremnika jednu po jednu

Samo jedan proizvođač ili potrošač može pristupiti međuspremniku u bilo kojem trenutku



Izazov:

Osigurajte da proizvođač neće pokušati dodati podatke u međuspremnik ako je pun i da potrošač neće pokušati ukloniti podatke iz praznog međuspremnika

Netočno rješenje

```
/* program producerconsumer */
int n;
binary_semaphore s = 1, delay = 0;
void producer() {
     while (true) {
          produce();
          semWaitB(s);
          append();
          n++;
          if (n==1) semSignalB(delay);
          semSignalB(s);
     }
}
```

```
void consumer() {
     semWaitB(delay);
     while (true) {
          semWaitB(s);
          take();
          n--;
          semSignalB(s);
          consume();
          if (n==0) semWaitB(delay);
     }
}
void main() {
     n = 0;
     parbegin (producer, consumer);
}
```

Mogući scenarij za program

	Producer	Consumer	s	*n*	Delay
1			1	0	0
2	semWaitB(s)		0	0	0
3	n++		0	1	0
4	**if** (n==1) (semSignalB(delay))		0	1	1
5	semSignalB(s)		1	1	1
6		semWaitB(delay)	1	1	0
7		semWaitB(s)	0	1	0
8		n--	0	0	0
9		semSignalB(s)	1	0	0
10	semWaitB(s)		0	0	0
11	n++		0	1	0
12	**if** (n==1) (semSignalB(delay))		0	1	1
13	semSignalB(s)		1	1	1
14		**if** (n==0) (semWaitB(delay))	1	1	1
15		semWaitB(s)	0	1	1
16		n--	0	0	1
17		semSignalB(s)	1	0	1
18		**if** (n==0) (semWaitB(delay))	1	0	0
19		semWaitB(s)	0	0	0
20		n--	0	-1	0
21		semSignalB(s)	1	-1	0

Ispravno rješenje

```
/* program producerconsumer */
int n;
binary_semaphore s = 1, delay = 0;
void producer() {
     while (true) {
          produce();
          semWaitB(s);
          append();
          n++;
          if (n==1) semSignalB(delay);
          semSignalB(s);
}
```

```
void consumer() {
     int m; /* a local variable */
     semWaitB(delay);
     while (true) {
          semWaitB(s);
          take();
          n--;
          m = n;
          semSignalB(s);
          consume();
          if (m==0) semWaitB(delay);
     }
}
void main() {
     n = 0;
     parbegin (producer, consumer);
}
```

Implementacija semafora

- Imperativ da se operacije semWait i semSignal implementiraju kao atomske
- Može se implementirati u hardware ili firmware
- Mogu se koristiti softverske sheme kao što su Dekkerovi ili Petersonovi algoritmi
- Druga alternativa je korištenje jedne od hardverski podržanih shema za međusobno isključivanje

Monitori

- Konstrukcija programskog jezika koja pruža ekvivalentnu funkcionalnost semaforima i lakša je za kontrolu
- Implementirano u više programskih jezika
  - Istodobni Pascal, Pascal-Plus, Modula-2, Modula-3, Java
  - Također je implementirana kao programska biblioteka
  - Softverski modul koji se sastoji od jedne ili više procedura, sekvence inicijalizacije i lokalnih podataka

Karakteristike monitora

Lokalne varijable podataka dostupne su samo postupcima monitora, a ne bilo kojim vanjskim procedurama

Proces ulazi u monitor pozivanjem jedne od njegovih procedura

Na monitoru se istovremeno može izvršavati samo jedan proces

Sinkronizacija

- Monitor podržava sinkronizaciju korištenjem varijabli uvjeta koje su sadržane unutar monitora i dostupne samo unutar monitora
  - Varijable uvjeta posebna su vrsta podataka u monitorima kojima upravljaju dvije funkcije:
    - cwait(c): obustaviti izvođenje procesa poziva pod uvjetom c
    - csignal(c): nastavak izvođenja nekog procesa blokiranog nakon čekanja pod istim uvjetom

Slanje poruka

- Kada se procesi međusobno izvršavajuu, moraju biti zadovoljena dva temeljna zahtjeva:

Sinkronizacija	Komunikacija
• Za provođenje međusobnog isključivanja	• Za razmjenu informacija

- Prosljeđivanje poruka je jedan pristup pružanju obje ove funkcije
  - Radi s distribuiranim sustavima, multiprocesorskim i jednoprocesorskim sustavima dijeljene memorije

Slanje poruka

- Stvarna funkcija se obično daje u obliku para funkcija:

```
send (odredište, poruka)
receive (izvor, poruka)
```

- Proces šalje informacije u obliku poruke drugom procesu definiranjem odredišta
- Proces prima informacije izvršavanjem primanja, navodeći izvor

Sinkronizacija

Komunikacija poruke između dva procesa podrazumijeva sinkronizaciju između njih

Primatelj ne može primiti poruku dok je ne pošalje drugi proces

Kada se funkcija primanja izvrši u procesu, postoje dvije mogućnosti:

Ako nema poruke na čekanju, proces je blokiran dok poruka ne stigne ili dok se proces ne nastavi izvršavati, napuštajući pokušaj primanja

Ako je poruka već poslana, poruka se prima i izvršavanje se nastavlja

Blokiranje slanja, blokiranje primanja

- I pošiljatelj i primatelj su blokirani dok se poruka ne dostavi
- Ponekad se naziva sastanak
- Omogućuje potpunu sinkronizaciju između procesa

Slanje bez blokiranja

Slanje bez blokade, blokiranje primanja

- Pošiljatelj nastavlja, ali primatelj je blokiran dok ne stigne tražena poruka
- Najkorisnija kombinacija
- Šalje jednu ili više poruka na različita odredišta što je brže moguće
- Primjer - uslužni proces koji postoji za pružanje usluge ili resursa drugim procesima

Slanje bez blokade, primanje bez blokade

- Nijedna strana ne čeka

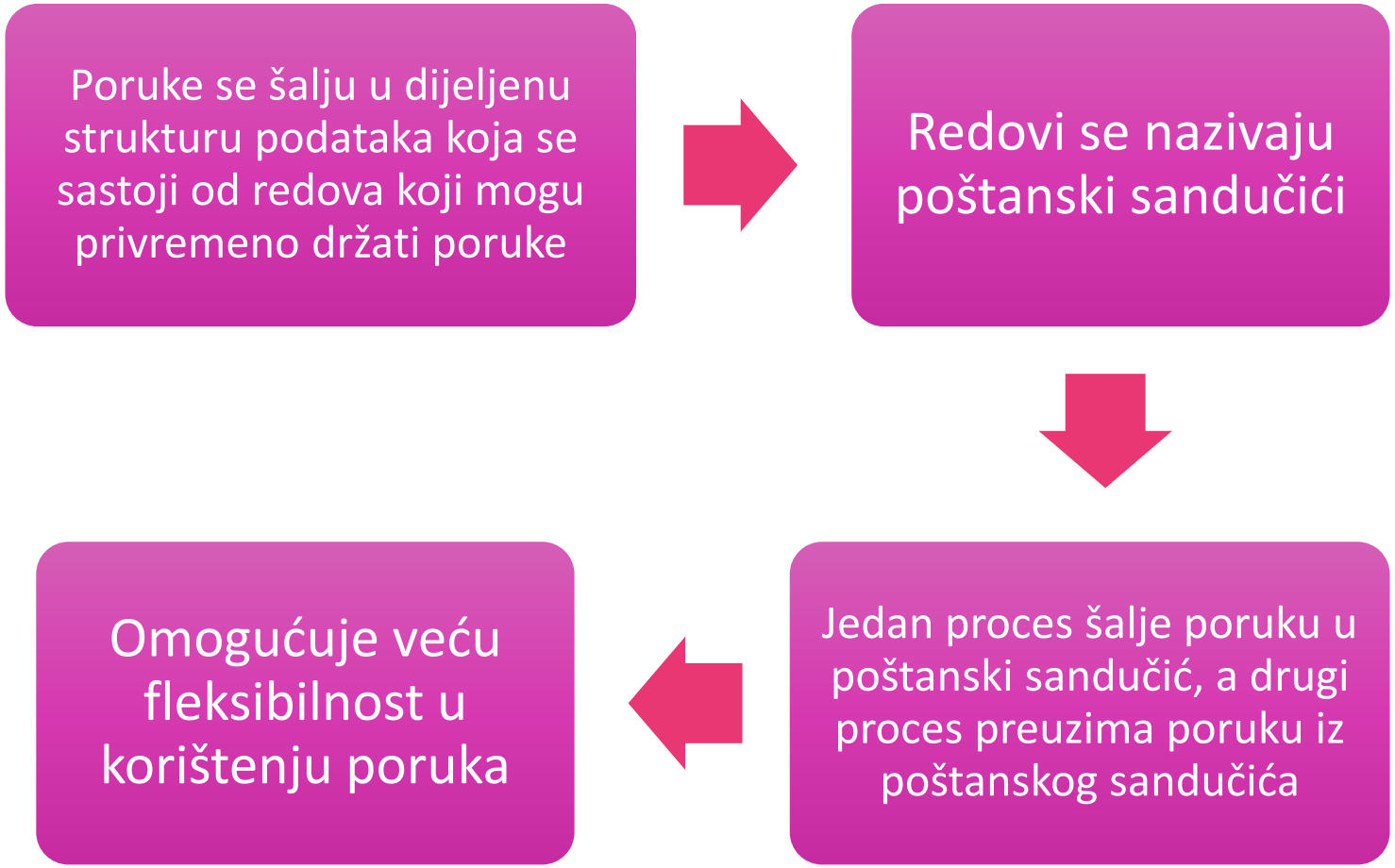
Adresiranje

- Sheme za određivanje procesa u funkcijama slanja i primanja padaju u dvije kategorije:

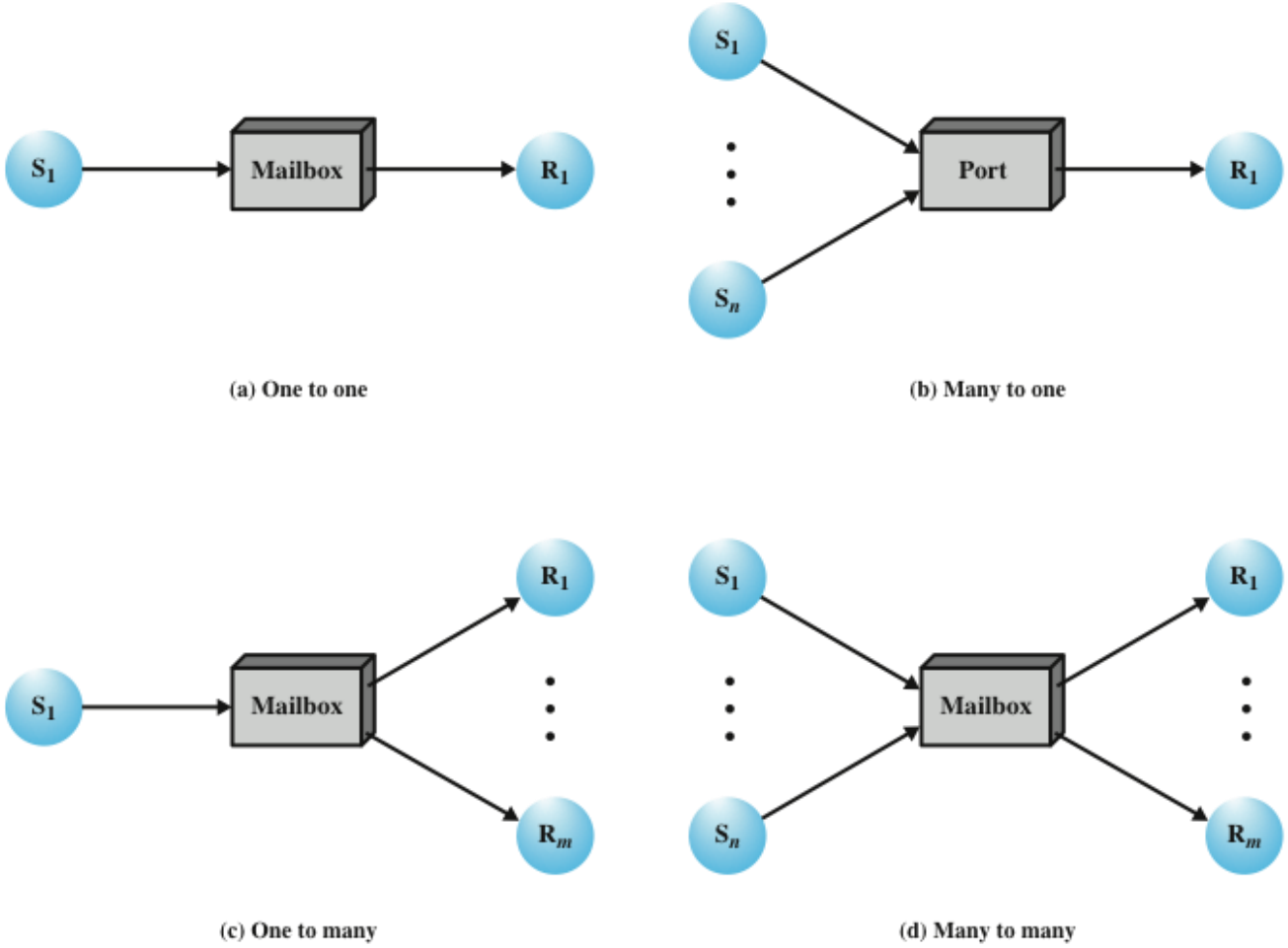
Izravno adresiranje

- Funkcija za slanje uključuje određeni identifikator odredišnog procesa
- Funkcija primanja može raditi na jedan od dva načina:
  - Zahtijevajte da proces eksplicitno odredi proces slanja
    - Učinkovito za suradnju u istodobnim procesima
  - Implicitno adresiranje
    - Izvorni parametar funkcije primanja ima vrijednost vraćenu kada se izvrši operacija primanja

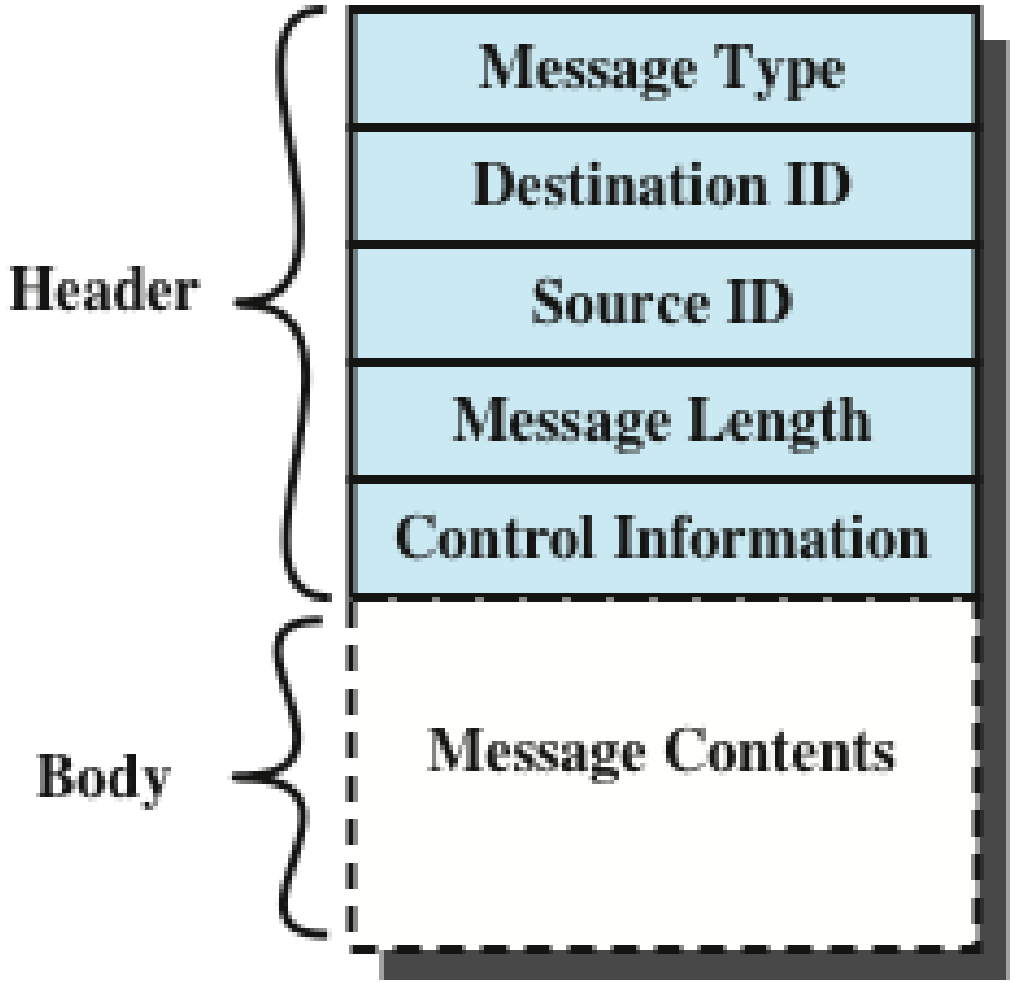
Neizravno adresiranje



Odnos između pošiljatelja i primatelja



Format poruke



Raspored u redu čekanja

- Najjednostavnija disciplina čekanja je prvi ušao-prvi izašao (FIFO)
  - To možda neće biti dovoljno ako su neke poruke hitnije od drugih
- Ostale alternative su:
  - Omogućiti određivanje prioriteta poruke, na temelju vrste poruke ili prema parametru pošiljatelja
  - Omogućiti primatelju da pregleda red poruka i odabere koju će poruku primiti sljedeću

Problem čitatelja/pisca

- Područje podataka dijeli se među mnogim procesima
  - Neki procesi samo čitaju podatkovno područje (čitači), a neki samo pišu u podatkovno područje (pisači)
- Uvjeti koji moraju biti zadovoljeni:
  - Bilo koji broj čitatelja može istovremeno čitati datoteku
  - Samo jedan po jedan pisač može pisati u datoteku
  - Ako pisač piše u datoteku, nijedan je čitatelj ne smije čitati dok ne završi s pisanjem

Sažetak

- Međusobno isključenje: softverski pristupi
  - Dekkerov algoritam
  - Petersonov algoritam
- Načela istodobnosti
  - Utrkivanje
  - OS problemi
  - Interakcija procesa
  - Zahtjevi za međusobno isključivanje
- Međusobno isključenje: hardverska podrška
  - Onemogućavanje prekida
  - Posebne strojne instrukcije
- Semafori
  - Međusobno isključivanje
  - Problem proizvođača/potrošača
  - Implementacija semafora
- Monitori
  - Monitor sa signalom
  - Alternativni model monitora s obavijesti i slanjem
- Slanje poruka
  - Sinkronizacija
  - Adresiranje
  - Format poruke
  - Raspored u redu čekanja
  - Međusobno isključivanje
- Problem čitatelja/pisca
  - Prioriteti
  - Raspored poruka

Thank you for your attention!