7. Višedretvena sinkronizacija i komunikacija

Semafori, monitori, potpuni zastoj,
rekurzivno zaključavanje, inverzija prioriteta,
signali, zajednička memorija, poruke, cjevovod

Semafori (1)

- vrlo jednostavan i učinkovit mehanizam
- ali samo za jednostavne primjene
  - ostvarenje kritična odsječka
  - brojanje resursa
- Osnovno sučelje za rad sa semaforima (prema POSIX-u)
  - `int sem_init(sem_t *sem, int pshared, unsigned init_value);`
  - `int sem_post(sem_t *sem);`
  - `int sem_wait(sem_t *sem);`

  - `int sem_trywait(sem_t *sem);` //pokušaj smanjiti ili vrati grešku
  - `int sem_timedwait(sem_t *sem, const struct timespec *max_wait);`
    - blokiraj ako treba, ali ne dulje od *max_wait*

Primjer korištenja semafora

- kodovi su uz predavanja na:
  - https://www.zemris.fer.hr/~leonardo/srsv/skripta/src/
- U primjerima na webu koriste se makroi
  - zašto?
  - bez makroa kod je čitljiviji (pogledati u *without_macro*)
  - ALI u „produkcijskoj” aplikaciji bi trebalo provjeravati povratne vrijednosti SVIH funkcija
  - ne smije se pretpostaviti da će sve biti uvijek u redu
  - to posebice vrijedi za SRSV-e
  - s makroima kod može ostati kraći, uz provjere

```
#define ACT_WARN      0
#define ACT_STOP      1
#define CALL(ACT,FUNC,...)          \
do {                                \
    if (FUNC(__VA_ARGS__)) {        \
        perror(#FUNC);              \
        if (ACT == ACT_STOP)        \
            exit(1);                \
    }                               \
} while (0)

/* umjesto:
 *    if (pthread_create(&t1,NULL, worker, (void *) p)) {
 *        perror("pthread_create");
 *        exit(-1);
 *    }
 * koristiti makro:
 *    CALL(ACT_STOP, pthread_create, &t1, NULL, worker, (void *) 1);
 *
 *---------------------------------------------------------------- */
```

(poželjno bi bilo dodati i informaciju o datoteci i i liniji koda gdje se to pozvalo)

```c
static sem_t sem;
static int work_in_progress = 1;

int main () {
   long i;
   pthread_t thr[THREADS];
   struct timespec t = {50, 0};

   CALL(ACT_STOP, sem_init, &sem, 0, 1);

   for(i = 0; i < THREADS; i++)
      CALL(ACT_STOP, pthread_create, &thr[i],
      NULL, worker, (void *) i+1);

   CALL(ACT_WARN, nanosleep, &t, NULL);

   work_in_progress = 0;

   for(i = 0; i < THREADS; i++)
      CALL(ACT_WARN, pthread_join, thr[i], NULL);

   return 0;
}
```

```c
static void *worker(void *p) {
   long id = (long) p;
   struct timespec t = { id, 0 };
   printf("Thread %ld starting\n", id);

   while(work_in_progress) {
      CALL(ACT_STOP, sem_wait, &sem);

      printf("Thread %ld inside C.S.\n", id);

      CALL(ACT_WARN, nanosleep, &t, NULL);

      printf("Thread %ld leaving C.S.\n", id);

      CALL(ACT_STOP, sem_post, &sem);

      CALL(ACT_WARN, nanosleep, &t, NULL);
   }

   printf("Thread %ld exiting\n", id);

   return NULL;
}
```

Semafori (2)

- u složenijim primjenama problemi pri upotrebi semafora
  - treba više semafora
  - nije ih jednostavno uskladiti
  - problem **potpuna zastoja**
  - problem **rekurzivnog zaključavanja**
  - problem **inverzije prioriteta**

Potpuni zastoj

- primjer: problem pet filozofa (obrađen u OS-u)
- vrlo je teško izbjeći problem potpuna zastoja ako imamo više od jednog semafora

```
dretva proizvođač
    ponavljaj
        p = proizvedi_poruku()
        ČekajSemafor(prazna)
        ČekajSemafor(ko)
        međuspremnik[ulaz] = p
        ulaz = (ulaz+1) mod N
        PostaviSemafor(ko)
        PostaviSemafor(puna)
```

```
dretva potrošač
    ponavljaj
        ČekajSemafor(ko)
        ČekajSemafor(puna)
        r = međuspremnik[izlaz]
        izlaz = (izlaz+1) mod N
        PostaviSemafor(prazna)
        PostaviSemafor(ko)
        potroši_poruku(r)
```

Monitori

- monitor je „samo alat”, stanje sustava definirati kroz dodatne varijable
  - semafori imaju vrijednost koju se može iskoristiti za „brojanje”, monitori ne
- kad je sinkronizacija složenija, lakše je koristiti monitore nego semafore
- sučelja:

```
int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *attr);
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
int pthread_cond_init(pthread_cond_t *cond, const pthread_condattr_t *attr);
int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex);
int pthread_cond_signal(pthread_cond_t *cond);
int pthread_cond_broadcast(pthread_cond_t *cond);
int pthread_mutex_trylock(pthread_mutex_t *mutex);
int pthread_mutex_timedlock(pthread_mutex_t *mutex, const struct timespec *abstm);
int pthread_cond_timedwait(pthread_cond_t *cond, pthread_mutex_t *mutex,
    const struct timespec *abstm);
```

Primjer

- https://www.zemris.fer.hr/~leonardo/srsv/skripta/src/without_macro/pthread_mutex.c

Rekurzivno zaključavanje (1)

```
funkcija_1()
    Uđi_u_monitor(m)
    nešto_radi
    funkcija_2()
    još_radi
    Izađi_iz_monitora(m)
```

```
funkcija_2()
    Uđi_u_monitor(m)
    nešto_drugo_radi
    Izađi_iz_monitora(m)
```

- dretva koja je već ušla u KO zaštićen semaforom ili monitorom opet pokušava zaključati isti objekt
  - može biti greška
  - može biti namjerno, očekujući mehanizam rekurzivnog zaključavanja
    - *POSIX mutexi* sa *PTHREAD_MUTEX_RECURSIVE* postavkom
- može se izbjeći drukčijim programiranjem

Rekurzivno zaključavanje (1)

- primjer izbjegavanja rekurzivnog zaključavanja:

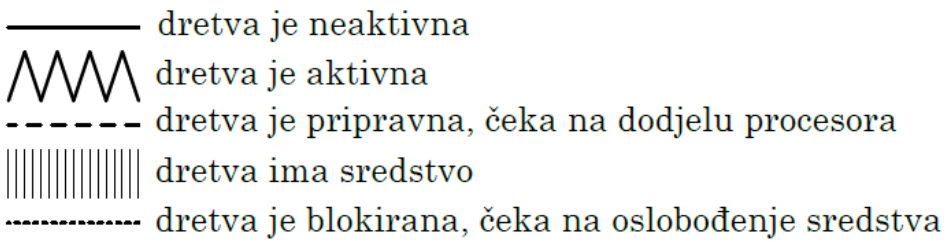
```
funkcija_1()
    Uđi_u_monitor(m)
    nešto_radi
    funkcija_3()
    još_radi
    Izađi_iz_monitora(m)
```

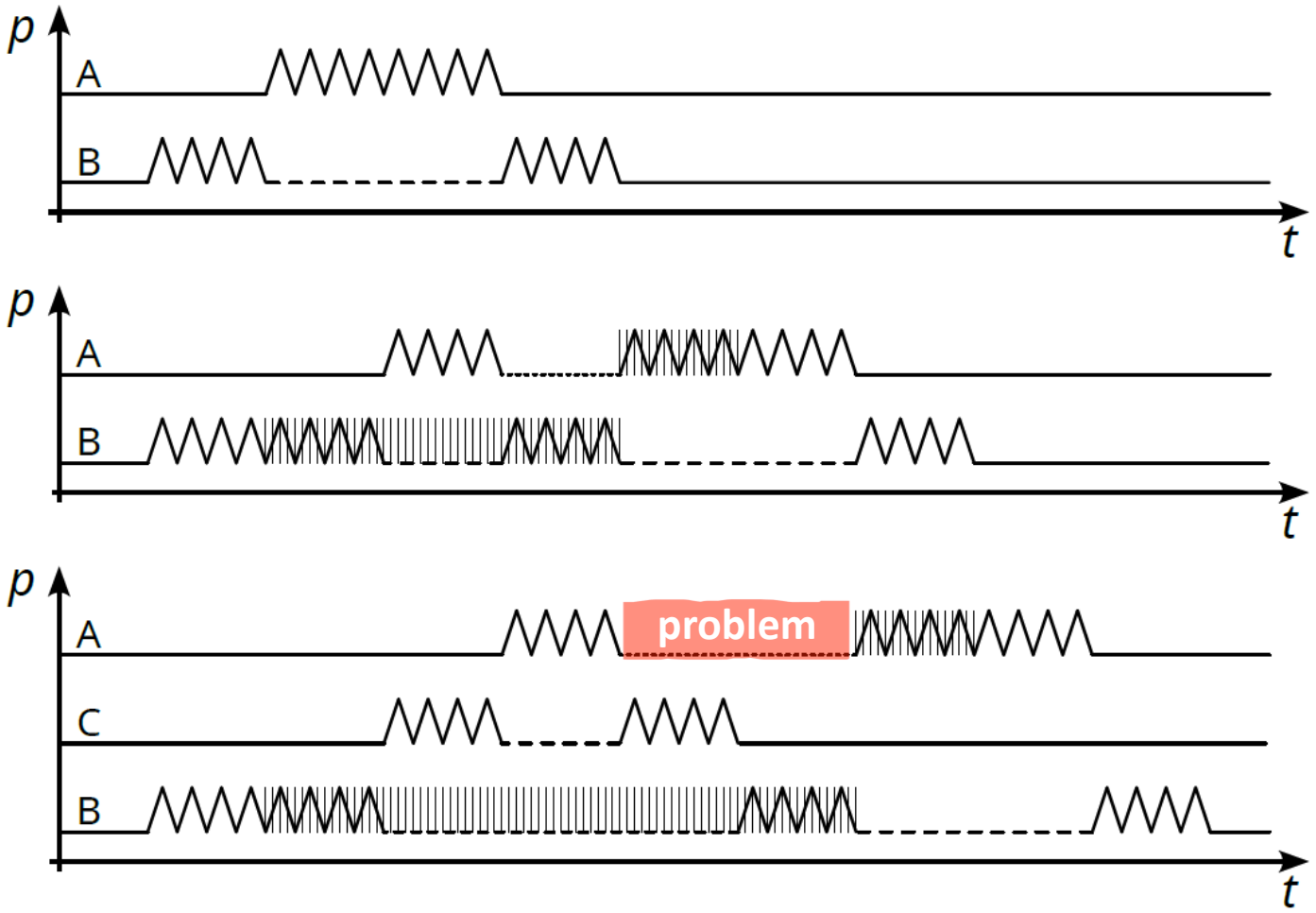
```
funkcija_2()
    Uđi_u_monitor(m)
    funkcija_3()
    Izađi_iz_monitora(m)
```

```
funkcija_3()
    nešto_drugo_radi
```

Problem inverzije prioriteta (1)

- očekivano ponašanje raspoređivanja dretvi:
  - dretva većeg prioriteta radi dok one manjeg čekaju
- sinkronizacijski mehanizmi mogu utjecati na to ponašanje!



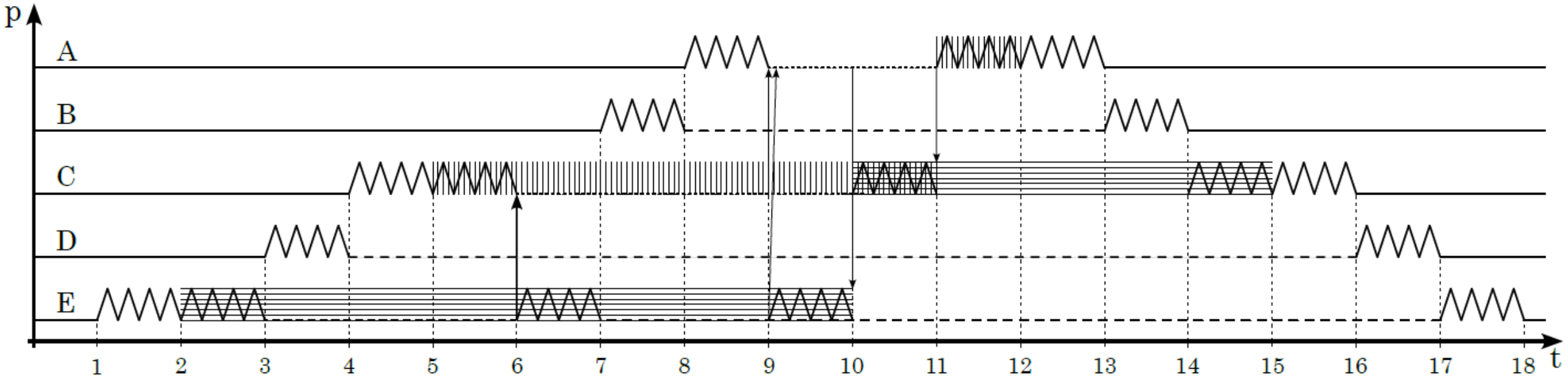


„Rješavanje problema”

- problem se ne može riješiti, ali se mogu ublažiti njegove posljedice, tj. skratiti vrijeme blokiranja prioritetne dretve
- dva najčešća protokola:
  - protokol nasljeđivanja prioriteta (engl. *priority inheritance protocol*)
  - protokol stropnog prioriteta (engl. *priority ceiling protocol*)

Protokol nasljeđivanja prioriteta

- ideja: u trenutku blokiranja dretve većeg prioriteta na semaforu (ili monitoru), dretvi koja je zauzela semafor privremeno povećati prioritet



- povećanje raditi i tranzitivno
- primjer u kodu

Protokol stropnog prioriteta

- za svaki semafor odrediti „stropni prioritet” – prioritet najprioritetnije dretve koja koristi taj semafor
- dvije inačice protokola:
  - pojednostavljeni protokol stropnog prioriteta (uglavnom podržan u OSu)
  - izvorni protokol stropnog prioriteta

Pojednostavljeni protokol stropnog prioriteta

- za semafor/monitor se definira stropni prioritet (sučeljem)
- kad dretva zauzme sredstvo odmah joj se povećava prioritet
- ostali nazivi:
  - izravni protokol stropnog prioriteta (engl. *immediate ceiling priority protocol*)
  - protokol zaštite prioritetom (engl. *priority protect protocol*) kod POSIX standarda
  - oponašanje protokola stropnog prioriteta (engl. *priority ceiling emulation*) kod programskog jezika Java.

Izvorni protokol stropnog prioriteta (1)

- algoritam iz „Rajkumar, 1991”
- sprječava i problem potpuna zastoja
- ukratko:
  - kako i kod protokola nasljeđivanja prioriteta u trenutku blokiranja prioritetnije dretve dretva koja je uzrok blokiranja nasljeđuju prioritet
  - ali ponekad se dretva blokira iako semafor nije zauzet, ali neki drugi jest
    - na taj način izbjeći potpuni zastoj

Izvorni protokol stropnog prioriteta (2)

Osnovne pretpostavke:

1. Razmatra se **skup dretvi** $\{\mathcal{D}_1, \mathcal{D}_2, \ldots \mathcal{D}_N\}$ koje u dijelovima svog izvođenja upotrebljavaju razna sredstva zaštićena binarnim semaforima $\{\mathcal{S}_1, \mathcal{S}_2, \ldots \mathcal{S}_M\}$. Za svaku je dretvu **unaprijed poznato koje će semafore možda** trebati tijekom rada, odnosno za svaki je semafor poznato koje se dretve mogu njime koristiti. Za svaki se semafor izračunava **stropni prioritet** $p(S_i)$ tako da se odabere najveća vrijednost među prioritetima dretvi koje se koriste tim semaforom.
2. Primjenjuje se prioritetno raspoređivanje – pripravna dretva najvećeg prioriteta se izvodi.

1. **Dretva $\mathcal{D}_i$ poziva *ČekajSemafor(S_x)*:**

   a) Niti jedan semafor nije zauzet – svi su prolazni.

   - Dretva $\mathcal{D}_i$ zauzima semafor S_x i nastavlja s radom (poziv radi uobičajeno).

   b) Semafor S_x je već zauzela dretva $\mathcal{D}_j$:

   - Poziv blokira dretvu $\mathcal{D}_i$ te dretva $\mathcal{D}_j$ nasljeđuje prioritet od $\mathcal{D}_i$: $p(\mathcal{D}_j) = \max\{p(\mathcal{D}_j), p(\mathcal{D}_i)\}$.
   - Nasljeđivanje prioriteta je tranzitivno: ako je dretva $\mathcal{D}_j$ blokirana zbog semafora S_y i dretve $\mathcal{D}_k$, onda i dretva $\mathcal{D}_k$ nasljeđuje prioritet od $\mathcal{D}_i$. I tako dalje.

   c) Semafor S_x nije zauzet, ali neki drugi semafori jesu:

   - Neka S^* označava semafor **najvećeg stropnog prioriteta** $p(S^*)$ koji je **zaključala** dretva $\mathcal{D}_k$.
   - Ako je $\mathcal{D}_k = \mathcal{D}_i$ ili $p(\mathcal{D}_i) > p(S^*)$, tada dretva $\mathcal{D}_i$ zauzima semafor S_x i nastavlja s radom.
   - Inače, ako je $p(\mathcal{D}_i) \leq p(S^*)$, poziv će blokirati dretvu $\mathcal{D}_i$ te dretva $\mathcal{D}_k$ nasljeđuje prioritet od $\mathcal{D}_i$: $p(\mathcal{D}_k) = \max\{p(\mathcal{D}_k), p(\mathcal{D}_i)\}$.
   - Nasljeđivanje prioriteta je tranzitivno.

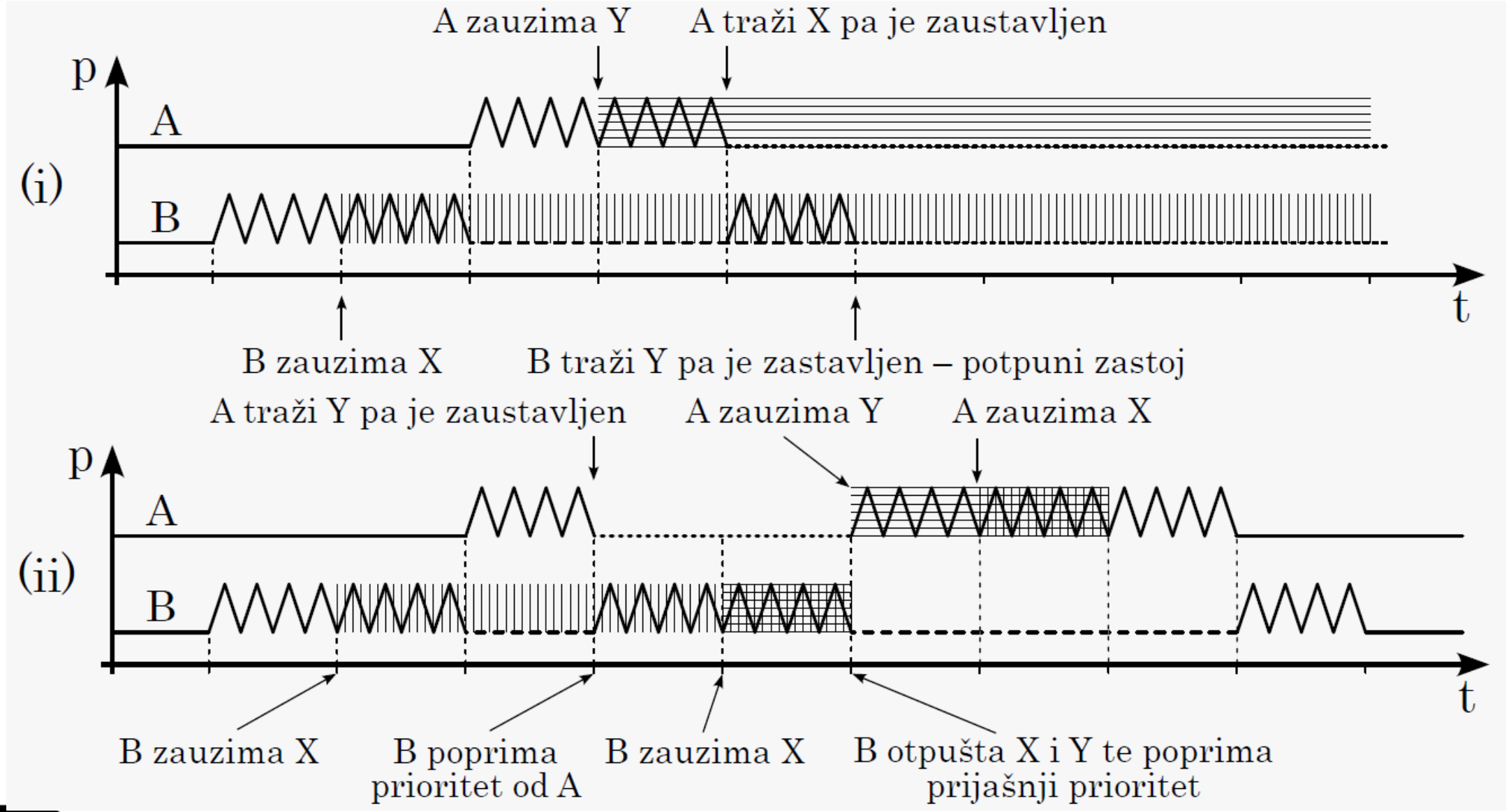
2. **Dretva $\mathcal{D}_i$ poziva *PostaviSemafor*(S_x)**.

   a) Ako je dretvi $\mathcal{D}_i$ bio povećan prioritet radi korištenja semafora S_x, prioritet joj se vraća na prijašnju vrijednost.

   b) Semafor S_x postavlja se u prolazno stanje.

   c) Ako ima blokiranih dretvi zbog toga što je dretva $\mathcal{D}_i$ imala semafor S_x onda se najprioritetnija od njih odblokira te ona ponovno izvodi svoj poziv *ČekajSemafor*(S_y) prema ovom protokolu.

Izbjegavanje potpunog zastoja



Primjer 7.5. Rad izvornog protokola stropnog prioriteta

Tablica 7.1. Sustav zadataka

Dretva	Prioritet	Sredstva
$\mathcal{D}_1$	20	S_1, S_2
$\mathcal{D}_2$	15	S_2, S_3
$\mathcal{D}_3$	10	S_3, S_4
$\mathcal{D}_4$	5	S_1, S_3

Tablica 7.2. Stropni prioriteti

Sredstvo	Stropni prioritet
S_1	20
S_2	20
S_3	15
S_4	10

Primjer 7.5. Rad izvornog protokola stropnog prioriteta

1. $\mathcal{D}_4$ se pojavljuje te kao dretva najvećeg prioriteta (jedina) odmah kreće s izvođenjem.
2. $\mathcal{D}_4$ poziva *ČekajSemafor*(S_3).

   S obzirom na to da niti jedan semafor nije još zaključan (ne postoji S^*), a i S_3 je prolazan, poziv neće blokirati te će dretva $\mathcal{D}_4$ nastaviti s radom.

3. $\mathcal{D}_2$ se pojavljuje te kao dretva najvećeg prioriteta istiskuje $\mathcal{D}_4$ i izvodi se.
4. $\mathcal{D}_2$ poziva *ČekajSemafor*(S_2).

   S^* (zaključani semafor najvećeg prioriteta) je S_3 sa stropnim prioritetom 15.

   $\mathcal{D}_2$ nema veći prioritet od 15 (ima 15), pa se blokira.

   $\mathcal{D}_4$ zbog toga (blokiranje $\mathcal{D}_2$) poprima prioritet od $\mathcal{D}_2$ (15) i nastavlja s radom.

Primjer 7.5. Rad izvornog protokola stropnog prioriteta

5. $\mathcal{D}_1$ se pojavljuje te kao dretva najvećeg prioriteta, istiskuje $\mathcal{D}_4$ te se izvodi.
6. $\mathcal{D}_1$ poziva *ČekajSemafor*(S_1).

   S^* (zaključani semafor najvećeg prioriteta) je S_3 sa stropnim prioritetom 15.

   $\mathcal{D}_1$ ima veći prioritet od 15 (ima 20), pa prolazi – zaključava S_1 i nastavlja s radom.

   S^* je sada S_1.

7. $\mathcal{D}_1$ se blokira na nekom drugom redu (npr. čeka dovršetak neke ulazne operacije).

   $\mathcal{D}_4$ nastavlja s radom.

Primjer 7.5. Rad izvornog protokola stropnog prioriteta

8. $\mathcal{D}_4$ otpušta sredstvo, tj. poziva *PostaviSemafor*(S_3).

   Semafor S_3 se otključava.

   $\mathcal{D}_4$ poprima prijašnji prioritet, tj. 5.

   Provjerava se mogućnost propuštanja blokiranih dretvi, tj. dretve $\mathcal{D}_2$.

      S obzirom na to da je stropni prioritet od $S^* = S_1$ i dalje veći od prioriteta $\mathcal{D}_2$, $\mathcal{D}_2$ ostaje blokirana.

   $\mathcal{D}_4$ nastavlja s radom.

POSIX funkcije za rješavanje problema inverzije prioriteta

- podržan je mehanizam monitora
- `int pthread_mutexattr_setprotocol(pthread_mutexattr_t *attr, int protocol);`

Protokol se odabire drugim parametrom. Mogućnosti su:

- `PTHREAD_PRIO_NONE` – bez upotrebe ijednog protokola
- `PTHREAD_PRIO_INHERIT` – upotreba nasljeđivanja prioriteta
- `PTHREAD_PRIO_PROTECT` – upotreba stropnog prioriteta.

- `int pthread_mutex_setprioceiling(pthread_mutex_t *mutex, int prioceiling, int *old_ceiling);`

Ostali mehanizmi sinkronizacije

- Zaključavanje čitaj/piši
- Zaključavanje radnim čekanjem
- Barijera

Signali (1)

- dojava događaja dretvi signalima
  - ona privremeno prekida što je tada radila, obrađuje događaj, vraća se i nastavlja s prijašnjim poslom
- sučeljem OS-a postavlja se ponašanje dretve za signal
  - prihvati signal i obradi ga **pretpostavljenom funkcijom** (definiranom u bibliotekama koje koristi program; najčešće je to prekid izvođenja dretve)
  - prihvati signal i obradi ga **zadanom funkcijom** (postavljenoj pri inicijalizaciji prihvata signala u programu)
  - **privremeno blokiraj signal** (signal ostaje na čekanju)
  - **odbaci signal** (ne poduzimaj nikakve akcije na njega, niti ga pamti)

Signali (2)

- `int sigaction(int sig, const struct sigaction *act, struct sigaction *oact);`
- u **act** se postavi funkcija za obradu
  - `void obrada_signala(int signum, siginfo_t *info, void *context) {}`
- primjer

- signali nisu baš čest mehanizam za SRSV
  - umjesto njih uglavnom se tamo koriste mehanizam poruka

Ostvarivanje međudretvene komunikacije

- zajednička memorija
- poruke
- cjevovodi

Zajednička memorija

- prednost: vrlo brz pristup, veličina podataka
- nedostatak: treba koristiti i sinkronizacijske mehanizme

- primjer (POSIX)

Redovi poruka

- poruka:
  - kratka informacija
  - može imati i prioritet (ili oznaku)
- način rada:
  - jedna dretva pošalje poruku u red
  - druga dretva uzima poruku iz reda
  - ako je red pun, dretva koja želi poslati se blokira
  - ako nema poruka u redu, dretva koja želi uzeti poruku se blokira
  - moguće je izbjeći čekanja posebnim zastavicama (onda funkcije javljaju grešku umjesto blokiranja)
- poruke su uobičajeni mehanizam komunikacije u RTOS-evima
- primjer (POSIX)

Cjevovodi

- slično redu poruka, ali:
  - nema granulacije podataka, novi podatci koji se pošalju u cijev budu „spojeni” s prethodnima – više se ne vidi granica
  - koristi se sučelje za rad s datotekama
    - ali čitanje je destruktivno (miču se podatci iz cijevi)
    - nema pomicanja po podatcima (*seek*)
- cjevovodi su bolji od poruka za prijenos više informacija (npr. datoteka), ali lošiji kad treba slati kratke informacije kod kojih je bitna granulacija između informacija (poruka)

- primjer (POSIX)

Uporaba višedretvenosti u SRSV-ima – sažetak (1)

- korištenje dretvi ima prednosti (već opisano u 3. i 4. poglavlju)
- mehanizmi OS-a na koje treba obratiti pažnju (odabrati postavke, ...):
  - raspoređivanje dretvi
  - sinkronizacija između dretvi (i mogući problemi)
  - komunikacija među dretvama
  - korištenje zajedničkih varijabli (zaštititi!!!)
  - trošak (overhead) jezgre za sve gornje mehanizme

Uporaba višedretvenosti u SRSV-ima – sažetak (2)

Tablica 7.3. Neki od mehanizama operacijskog sustava za potporu višedretvenosti

mehanizam	uobičajeni odabir	dodatne mogućnosti
raspoređivanje	raspoređivanje prema prioritetu SCHED_FIFO SCHED_RR	raspoređivanje prema krajnjim trenucima dovršetaka – SCHED_DEADLINE raspoređivanje sporadičnih zadataka – SCHED_SPORADIC
sinkronizacija	semafori monitori	operacije *probaj_čekati*, *čekaj_ograničeno* (npr. `sem_trywait`, `sem_timedwait`) rekurzivno zaključavanje nasljeđivanje prioriteta stropni prioritet
komunikacija	zajednički spremnik redovi poruka cjevovodi	signali datoteke mrežna komunikacija