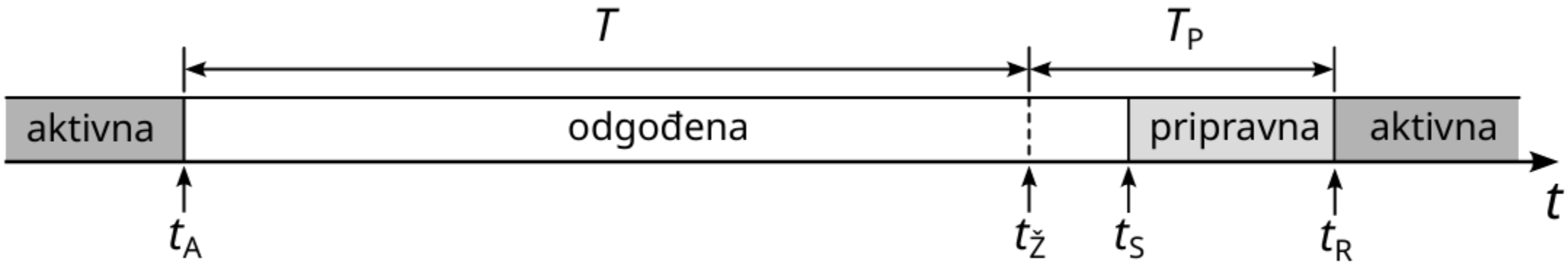
6. Upravljanje vremenom

Satovi, alarmi

POSIX sučelje

Problem odgoda zbog drugih događaja



Slika 6.1. Stanje dretve pri odgodi

- problemi koji mogu uzrokovati (nepredviđene, neuračunate) odgode:
  - obrade prekida i drugi poslovi jezgre
  - druge prioritetnije dretve
  - nepreciznost satnih mehanizama
- treba uzeti u obzir ove probleme i prilagoditi program

6.1. Korištenje satnih mehanizama u operacijskim sustavima

- mehanizmi u granulaciji sekunde (*time(), sleep(), alarm()*) nam nisu zanimljivi
- mehanizmi u granulaciji mikrosekunde i nanosekunde su bolji izbor
  - u teoriji su nam oni dostatni
  - ali, iako jedinice koje sučelja primaju jesu izražena u npr. nano sekundama, stvarna granulacija koju OS koristi pri tim operacijama može biti značajno krupnija, npr. čak i milisekunda!
  - prije korištenja potrebno je ispitati granulaciju sata
    - OS ponekad nudi sučelje kojim se to može izravno dohvatiti
      npr. **clock_getres()**
      - ali to je samo preciznost, ne nužno i granulacija operacija (odgode, alarma)
    - ali i same operacije sa satom traju
      - npr. dva put zaredom pozvati *clock_gettime()* i usporediti dobiveno

Uobičajeni satovi: CLOCK_REALTIME

- ostvaruje „stvarno vrijeme”, „sat”
  - pomoću njega se može doznati koliko je sati, npr. 12:23:43.123
  - koristi se kada neke akcije treba pokretati u trenucima usklađenim sa satom
    - npr. na svaki puni sati; svakih 15 minuta (u 00, 15, 30, 45)
  - OS povremeno usklađuje ovaj sat s udaljenim poslužiteljem
    - ovo može stvoriti probleme ako se ovaj sat koristi za neke periodičke akcije (npr. svakih 50 ms i slično)
    - zato se ovaj sat ne preporuča za to

Uobičajeni satovi: CLOCK_MONOTONIC

- sat koji odbrojava istim tempom kao i CLOCK_REALTIME, ali ne predstavlja sat sustava
  - najčešće je njegova vrijednost nula u trenutku pokretanja
- on se ne ažurira, on uvijek samo odbrojava istim tempom
- zato se on koristi za periodičke aktivnosti gdje nije bitno „koliko je sati” već samo protok vremena (npr. „svakih 35 ms radit to”)
- također se može koristiti za mjerenje protoka vremena: „koliko je proteklo od tada do tada”

Sučelja

- *int clock_gettime(clockid_t clock_id, struct timespec *tp);*
  - dohvati sat
- *int clock_nanosleep(clockid_t clock_id, int flags, const struct timespec *rqtp, struct timespec *rmtp);*
  - odgodi izvođenje dretve
  - flags: 0 ili *TIMER_ABSTIME*
  - *rmtp* – ako je prekinut prije vrati neprospavano vrijeme

Primjer

```
int main ()
{
        long i;
        pthread_t tid[BROJ_DRETVI];

        clock_gettime(CLOCK_REALTIME, &t0); /* vrijeme početka */
        clock_gettime(CLOCK_MONOTONIC, &tx0);

        for (i = 0; i < BROJ_DRETVI; i++) {
                if (pthread_create(&tid[i], NULL, posao_dretve, (void *) (i+1)))
                {
                        perror("Error: pthread_create");
                        return 1;
                }
        }

        for (i = 0; i < BROJ_DRETVI; i++)
                pthread_join(tid[i], NULL);

        return 0;
}
```

```
static void *posao_dretve(void *param)
{
        long id = (long) param;
        int iter;
        struct timespec iduca_aktivacija, period;
        unsigned long long i;

        /* period = 500 ms * id */
        period.tv_sec = id / 2;
        period.tv_nsec = (id % 2) * 500000000;

        iduca_aktivacija = tx0; //MONOTONIC

        for (iter = 0; iter < 100; iter++)
        {
                timestamp(id, "POCETAK", iter);
                for (i = 0; i < id * BROJAC; i++)               //#define BROJAC 30000000ULL
                        asm volatile ("":::"memory");
                timestamp(id, "KRAJ", iter);

                timespec_add(&iduca_aktivacija, &period);
                clock_nanosleep(CLOCK_MONOTONIC, TIMER_ABSTIME,
                                                &iduca_aktivacija, NULL);
        }

        return NULL;
}
```

```
void timespec_add(struct timespec *A, struct timespec *B)
{
        A->tv_sec += B->tv_sec;
        A->tv_nsec += B->tv_nsec;
        if (A->tv_nsec >= 1000000000) {
                A->tv_sec++;
                A->tv_nsec -= 1000000000;
        }
}
void timespec_sub(struct timespec *A, struct timespec *B)
{
        A->tv_sec -= B->tv_sec;
        A->tv_nsec -= B->tv_nsec;
        if (A->tv_nsec < 0) {
                A->tv_sec--;
                A->tv_nsec += 1000000000;
        }
}
void timestamp(long id, char *msg, int iter)
{
        struct timespec t;
        pthread_mutex_lock(&monitor);
        clock_gettime(CLOCK_REALTIME, &t);
        timespec_sub(&t, &t0);
        printf("[%02ld:%06ld] Dretva %ld: %s iteracija=%d\n",
                t.tv_sec % 100, t.tv_nsec/1000, id, msg, iter);
        pthread_mutex_unlock(&monitor);
}
```

```
Primjer pokretanja:
$ gcc periodic-tasks-1.c -pthread -Wall
$ ./a.out                                          # na jednoprocesorskom sustavu
[00:000140] Dretva 3: POCETAK iteracija=0
[00:000460] Dretva 2: POCETAK iteracija=0
[00:008245] Dretva 1: POCETAK iteracija=0
[00:246640] Dretva 1: KRAJ iteracija=0
[00:418966] Dretva 2: KRAJ iteracija=0
[00:500457] Dretva 1: POCETAK iteracija=1
[00:522505] Dretva 3: KRAJ iteracija=0
[00:613793] Dretva 1: KRAJ iteracija=1
[01:000499] Dretva 2: POCETAK iteracija=1
[01:004525] Dretva 1: POCETAK iteracija=2
...
$ ./a.out                                          # na drugom sustavu, višeprocesorskom, u WSL-u
[00:000087] Dretva 1: POCETAK iteracija=0
[00:000142] Dretva 2: POCETAK iteracija=0
[00:000200] Dretva 3: POCETAK iteracija=0
[00:053350] Dretva 1: KRAJ iteracija=0
[00:101359] Dretva 2: KRAJ iteracija=0
[00:150136] Dretva 3: KRAJ iteracija=0
[00:500129] Dretva 1: POCETAK iteracija=1
[00:579458] Dretva 1: KRAJ iteracija=1
[01:000133] Dretva 1: POCETAK iteracija=2
[01:000278] Dretva 2: POCETAK iteracija=1
[01:091997] Dretva 1: KRAJ iteracija=2
[01:151397] Dretva 2: KRAJ iteracija=1
[01:500144] Dretva 1: POCETAK iteracija=3
[01:500287] Dretva 3: POCETAK iteracija=1
```

Periodički alarm

```
int timer_create(clockid_t clockid, struct sigevent *evp, timer_t *timerid);
int timer_settime(timer_t timerid, int flags, const struct itimerspec *value,
                  struct itimerspec *ovalue);
int timer_gettime(timer_t timerid, struct itimerspec *value);
int timer_getoverrun(timer_t timerid);
int clock_getres(clockid_t clock_id, struct timespec *res);
int sigaction(int sig, const struct sigaction *act, struct sigaction *oact);
```

Postupak:

1. stvori alarm: *timer_create*
2. pokreni ga: *timer_settime*

❑ što napraviti u „aktivaciji” (*sigevent*)? mogućnosti:

- ➢ poslati signal dretvi
- ➢ stvoriti dretvu koja će obraditi događaj

Nadzorni alarm

- mnogi ugrađeni sustavi su „daleko”
- što napraviti ako nešto krene krivo?
- jedan od načina je „resetirati” sustav
- ali kako detektirati da je nešto pošlo krivo?
- često korišteni mehanizam: nadzorni alarm (*watchdog timer*)
- ideja je da se znaju trajanja pojedinih operacija te da ako nešto predugo traje to označava problem (nerješivi) i da treba resetirati sustav
- nadzorni alarm = brojilo koje odbrojava do nule i ako stigne do nule onda na *reset* ulaz procesora šalje signal
- zadatak programa je da periodički upisuje vrijednost u brojilo tako da ono nikad ne dođe do nule u normalnom radu

Primjer 6.1. Primjer korištenja nadzornog alarma

```
int main()
{
    uint16 volatile *brojilo = (uint16 volatile *) 0xFF0000;

    inicijalizacija();

    for (;;) {
        *brojilo = 10000;
        očitaj_stanje_senzora();
        izračunaj_i_pošalji_naredbe();
        zapiši_stanje_sustava();
    }
}
```