- `queue` – pokazivač na red u koji je potrebno dodati novi element

Povratna vrednost:
Nema.

```
int OSDequeue(OSQueue *queue);
```

Opis: Funkcija koja uklanja element iz reda. Kako je u pitanju prioritetni red, elementi će biti uklonjeni po proritetu, počevši od onih sa najvišim prioritetom. Ukoliko je red prazan, zadatak će biti blokiran dok se ne doda prvi element. Dobra praksa je da samo jedan zadatak ima mogućnost da pozove ovu funkciju.

Parametri:

- `queue` – pokazivač na red iz kojeg je potrebno ukloniti element

Povratna vrednost:
Element tipa `int16_t` koji je na početku reda.

22.4 Primeri aplikacija u ArdOS

U okviru ovog poglavlja će biti prikazani primeri aplikacija u *ArdOS* operativnom sistemu. Primeri su koncipirani na način da redom uvode nove servise. Za pokretanje svih primera je neophodan samo Arduino koji ima mogućnost serijske komunikacije sa računarom.

Osnovna aplikacija u ArdOS

Primer 22.4.1. Implementirati aplikaciju koja omogućava istovremeno treperenje diode frekvencijom od 1Hz i ispisivanje sistemskog vremena putem serijskog porta svakih 500ms.

Rešenje:

Kako je potrebno realizovati istovremeno izvršavanje dve funkcionalnosti, biće neophodno iskoristiti dva zadatka. Prvi zadatak će biti namenjen realizaciji treperenja diode. Funkcija koja će biti povezana sa ovim zadatkom je *task1*, a kao parametar ove funkcije je prosleđena perioda treperenja, koja u ovom slučaju iznosi 1000ms. Drugi zadatak će biti realizovan u okviru funkcije *task2*, koja će slati poruku u čijem je sadržaju sistemsko vreme svakih 500ms. Ovaj vremenski interval je, takođe, prosleđen kao parametar funkcije prilikom kreiranja zadatka. Rešenje zadatka je prikazano na listingu 22.9.

```
#include <stdio.h>
```

```
#include "kernel.h"
#include "pin.h"
#include "usart.h"

// Prvi zadatak
void task1(void *p)
{
    // Ucitavanje parametra
    uint16_t ms = (uint16_t)p;
    ms = ms/2;

    // Treperenje diode
    while(1)
    {
        pinSetValue(PORT_B, 5, HIGH);
        OSSleep(ms);
        pinSetValue(PORT_B, 5, LOW);
        OSSleep(ms);
    }
}

// Drugi zadatak
void task2(void *p)
{
    // Ucitavanje parametra
    int8_t msg[64];
    uint16_t ms = (uint16_t)p;

    // Slanje poruke putem serijskog porta
    while(1)
    {
        OSSleep(ms);
        sprintf(msg, "Task2: %ldms\r\n", OSticks());
        usartPutString(msg);
    }
}

int16_t main()
{
    // Inicijalizacija pina koji upravlja LED diodom i
        serijske komunikacije
    pinInit(PORT_B, 5, OUTPUT);
    usartInit(115200);

    OSInit(2); // Inicijalizacija operativnog sistema sa dva
        zadatka

    // Inicijalizacija zadataka
    OSCreateTask(0, task1, (void *) 1000);
    OSCreateTask(1, task2, (void *) 500);
```

```
    OSRun(); // Pokretanje operativnog sistema

    while(1);
}
```

Listing 22.9: Osnovna aplikacija u ArdOS

Nakon pokretanja aplikacije, u okviru serijskog terminala je moguće očitati poruke koje šalje Arduino. Poruke sadrže broj milisekundi proteklih od pokretanja operativnog sistema. Kako je u primeru zahtevano da se ove poruke šalju svakih 500ms, zadatak koji ovo izvršava je upravo "uspavan" u toku ovog vremenskog intervala. Međutim, ukoliko se uporede dve uzastopno dobijene poruke, prikazane na listingu 22.10, primetiće se da je vreme između dva slanja poruke 502ms. Ovo je posledica toga da se pored zadataka na mikrokontroleru, izvršava i operativni sistem, čije funkcije takođe zahtevaju određeno vreme za izvršavanje. Takođe, prilikom prelaska sa jednog zadatka na drugi vrši se zamena konteksta koja zahteva određeno vreme.

```
Task2: 501ms
Task2: 1003ms
Task2: 1505ms
Task2: 2007ms
```

Listing 22.10: Ispis osnovne aplikacije u ArdOS

Primena semafora u ArdOS

Primer 22.4.2. Implementirati aplikaciju koja čeka na prihvat poruke "*START*" putem serijskog porta, nakon čega započinje treptaj diode frekvencijom od 1Hz. Svakim novim prihvatom ispravne poruke je potrebno izvršiti novi treptaj. Dodatno, potrebno je omogućiti da se poruke mogu prihvatati u toku treperenja diode.

Rešenje:

Kako je potrebno realizovati istovremeno izvršavanje dve funkcionalnosti, biće neophodno iskoristiti dva zadatka. Prvi zadatak će biti namenjen realizaciji treperenja diode. Funkcija koja će biti povezana sa ovim zadatkom je *task1*, a kao parametar ove funkcije je prosleđena perioda treperenja koja je u ovom slučaju 1000ms. Drugi zadatak će biti realizovan u okviru funkcije *task2*, koja će biti zadužena za prihvat poruke, proveru da li je ona validna i signalizaciju da je potrebno izvršiti još jedan treptaj. Broj treptaja koji je potrebno izvršiti će biti realizovan pomoću brojačkog semafora. Semafor je inicijalizovan kao brojački, čija je početna vrednost 0. Drugi zadatak će prilikom prijema ispravne poruke svaki put osloboditi semafor, što će u njemu interno inkrementovati brojač. Za razliku od njega, prvi zadatak će čekati sve dok ne bude mogao da zauzme semafor, što će se desiti u situaciji kad interni

brojač semafora postane različit od nule. Kada prvi zadatak zauzme brojač, interni brojač semafora će se dekrementovati. Pri tome, prvi zadatak će moći da zauzme brojač onoliko puta koliko ga je puta drugi zadatak oslobodio. Kako se ova dva zadatka izvršavaju istovremeno, semafor će moći da bude oslobođen više puta u toku jedne periode treptaja, odnosno između dva uzastopna zauzimanja, što je i bilo potrebno realizovati. Rešenje zadatka je prikazano na listingu 22.11.

```
#include "kernel.h"
#include "sema.h"
#include "pin.h"
#include "usart.h"

OSSema sem; // Kreiranje objetka za semafor

void task1(void *p)
{
    // Ucitavanje parametra
    uint16_t ms = (uint16_t)p;
    ms = ms/2;
    uint8_t i;

    while(1)
    {
        OSTakeSema(&sem); // Zauzimanje semafora

        // Treptanje diode
        pinSetValue(PORT_B, 5, HIGH);
        OSSleep(ms);
        pinSetValue(PORT_B, 5, LOW);
        OSSleep(ms);
    }
}

void task2(void *p)
{
    int8_t msg[64];
    uint8_t msg_len;
    int8_t start_msg[] = "START";
    uint8_t flag;
    uint8_t i;

    while(1)
    {
        // Ucitavanje poruke
        usartPutString("Type␣\"START\"␣to␣enable␣LED␣pulsing!\
            r\n");
        while(usartAvailable() == 0)
            OSSleep(10);
        OSSleep(100);
```

```
        msg_len = usartGetString(msg);

        // Provera da li je poruka validna
        if(msg_len == 5)
        {
            flag = 0;
            for(i = 0; i < 5; i++)
            {
                if(msg[i] != start_msg[i])
                    flag = 1;
            }
            if(flag == 0)
            {
                usartPutString("SUCCESS!\r\n");
                OSGiveSema(&sem); // Oslobadjanje semafora
            }
            else
                usartPutString("FAIL!\r\n");
        }
        else
            usartPutString("FAIL!\r\n");
    }
}

int16_t main()
{
    // Inicijalizacija pina koji upravlja LED diodom i
       serijske komunikacije
    pinInit(PORT_B, 5, OUTPUT);
    usartInit(115200);
    usartPutString("Test!\r\n");

    OSInit(2); // Inicijalizacija operativnog sistema sa dva
       zadatka

    OSCreateSema(&sem, 0, 0); // Inicijalizacija semafora

    // Inicijalzziacija zadataka
    OSCreateTask(0, task1, (void *) 1000);
    OSCreateTask(1, task2, 0);

    OSRun(); // Pokretanje operativnog sistema

    while(1);
}
```

Listing 22.11: Semafori u ArdOS

U okviru ovog primera, prikazana je procedura za učitavanje poruka poslatih ka Arduino platformi. Razlika u odnosu na standardan način učitavanja poruke koji

se izvršava u običnoj aplikaciji je ta, da se, prilikom čekanja na prihvat poruke, poziva funkcija *OSSleep()*. Razlog ovoga jeste potreba da se omogući mikrokontroleru da izvršava druge zadatke. Ukoliko bi beskonačna petlja bila takva da nema telo, ni u jednom trenutku ne bi prepustila kontrolu drugom zadatku, što bi obesmislilo koncept operativnog sistema. Zbog toga se u okviru petlje zadatak "uspavljuje" u periodu od 10ms.

Primena muteksa u ArdOS

Primer 22.4.3. Implementirati aplikaciju koja omogućava slanje tri uzastopne poruke svakih 200ms putem serijskog porta od strane tri različita zadatka. Istovremeno omogućiti treperenje diode frekvencijom od 1Hz.

Rešenje:

Kako je potrebno realizovati istovremeno izvršavanje četiri funkcionalnosti, biće neophodno iskoristiti četiri zadatka. Prva tri zadataka će biti namenjena realizaciji slanja poruka preko serijskog porta. Funkcije koje će biti povezane sa ovim zadacima su *task1*, *task2* i *task3*. Četvrti zadatak će biti realizovan u okviru funkcije *task4*, koja će biti zadužena za implementaciju treptanja diode. Ideja ovog primera jeste da prikaže upotrebu muteksa u omogućavanju pojedinačnog pristupa nekom resursu od strane više različitih zadataka. Naime, kako se u ovom primeru zahteva da tri različita zadatka koriste isti resurs, u ovom slučaju serijski port, potrebno ih je ograničiti tako da u jednom trenutku samo jedan zadatak može da ga upotrebi. Ovo se postiže upravo primenom muteksa. Rešenje zadatka je prikazano na listingu 22.12.

```
#include "kernel.h"
#include "mutex.h"
#include "pin.h"
#include "usart.h"

// Konstanta koja omogucava upotrebu muteksa
#define USE_MUTEX

OSMutex mutex; // Kreiranje objekata za muteks

void task1(void *p)
{
    uint8_t i;
    int8_t msg[64];

    while(1)
    {
        #ifdef USE_MUTEX
        OSTakeMutex(&mutex); // Zauzimanje muteksa
        #endif
```

```
        // Ispis poruke
        for(i = 0; i < 3; i++)
        {
            sprintf(msg, "Task1: %d\r\n", i);
            usartPutString(msg);
            OSSleep(200);
        }

        #ifdef USE_MUTEX
        OSGiveMutex(&mutex); // Oslobadjanje muteksa
        #endif
    }
}

void task2(void *p)
{
    uint8_t i;
    int8_t msg[64];

    while(1)
    {
        #ifdef USE_MUTEX
        OSTakeMutex(&mutex); // Zauzimanje muteksa
        #endif

        // Ispis poruke
        for(i = 0; i < 3; i++)
        {
            sprintf(msg, "Task2: %d\r\n", i);
            usartPutString(msg);
            OSSleep(200);
        }

        #ifdef USE_MUTEX
        OSGiveMutex(&mutex); // Oslobadjanje muteksa
        #endif
    }
}

void task3(void *p)
{
    uint8_t i;
    int8_t msg[64];

    while(1)
    {
        #ifdef USE_MUTEX
        OSTakeMutex(&mutex); // Zauzimanje muteksa
        #endif
```

```
        // Ispis poruke
        for(i = 0; i < 3; i++)
        {
            sprintf(msg, "Task3: %d\r\n", i);
            usartPutString(msg);
            OSSleep(200);
        }

        #ifdef USE_MUTEX
        OSGiveMutex(&mutex); // Oslobadjanje muteksa
        #endif
    }
}

void task4(void *p)
{
    // Treptanje diode
    while(1)
    {
        pinSetValue(PORT_B, 5, HIGH);
        OSSleep(500);
        pinSetValue(PORT_B, 5, LOW);
        OSSleep(500);
    }
}

int16_t main()
{
    // Inicijalizacija pina koji upravlja LED diodom i
        serijske komunikacije
    pinInit(PORT_B, 5, OUTPUT);
    usartInit(115200);

    OSInit(4); // Inicijalizacija operativnog sistema sa
        cetiri zadatka

    // Inicijalizacija muteksa
    OSCreateMutex(&mutex);

    // Inicijalizacija zadataka
    OSCreateTask(0, task1, 0);
    OSCreateTask(1, task2, 0);
    OSCreateTask(2, task3, 0);
    OSCreateTask(3, task4, 0);

    OSRun(); // Pokretanje operativnog sistema

    while(1);
}
```

Listing 22.12: Muteksi u ArdOS

Rešenje zadatka prikazano na listingu 22.12 sadrži dodatnu makro konstantu. Ova konstanta je upotrebljena u cilju odabira načina kompajliranja aplikacije. Ukoliko je definisana, muteks će biti upotrebljen, a ukoliko nije, muteks će biti izostavljen. Pokretanjem aplikacije sa definisanom i nedefinisanom konstantom će rezultovati u različitom izvršavanju aplikacije. Ukoliko je konstanta definisana, zadatak će, nakon zauzimanja muteksa, imati mogućnost da ispiše sve tri poruke uzastopno, što će značiti da je on zauzeo resurs serijskog porta, a drugi zadaci će čekati da ga oslobodi. U slučaju da konstanta nije definisana, zadaci će se boriti oko posedovanja serijskog porta, što će prouzrokovati istovremeno ispisivanje od strane svih zadataka. U nekim situacijama ovo neće predstavljati problem. Međutim, ukoliko je to neki hardverski resurs, za čije upravljanje je potrebno izvesti odgovarajuću sekvencu događaja, istovremeni pokušaj izvršavanja različitih sekvenci od strane dva zadatka će napraviti koliziju, koja može dovesti do greške, odnosno kvara sistema. Zbog toga se najćešće upotrebljavaju muteksi, kako bi se ograničila upotreba resursa na samo jedan zadatak. Ispis putem serijskog porta za slučaj definisane konstante je prikazan u levoj koloni na listingu 22.13, a u slučaju kada konstanta nije definisana u desnoj.

```
Task1: 0
Task1: 1
Task1: 2
Task2: 0
Task2: 1
Task2: 2
Task1: 0
Task1: 1
Task1: 2
Task2: 0
Task2: 1
Task2: 2
```

```
Task1: 0
Task2: 0
Task3: 0
Task1: 1
Task2: 1
Task3: 1
Task1: 2
Task2: 2
Task3: 2
Task1: 0
Task2: 0
Task3: 0
```

Listing 22.13: Ispis primera za rad sa muteksima

Rezultati prikazani u desnoj koloni na listingu 22.13 su očekivani. Zadaci istovremeno pokušavaju da pristupe resursu, koji je u ovom slučaju serijski port, rezultujući u koliziji, tj. neizvršavanju funkcionalnosti onako kako je po specifikaciji definisano. Rešenje ovog problema bi trebalo da bude uvođenje muteksa, međutim, rezultat prikazan u levoj koloni na listingu 22.13 ukazuje na dodatne probleme koji se tom prilikom javljaju. Posmatrajući ove rezultate, može se primetiti da se treći zadatak nikada ne izvrši, odnosno nikada ne uđe u kritični deo koda ograničen muteksom, što se prikazuje nedostatkom poruka koje treba da ispiše. Razlog ove činjenice leži u prioritetnom izvršavanju zadataka, gde se nakon oslobađanja muteksa u prvom zadatku aktivira zauzimanje muteksa u spremnom zadatku najvišeg prioriteta, što je u ovom slučaju drugi zadatak. Slična priča se ponavlja nakon oslobađanja muteksa na kraju drugog zadatka. Međutim, u ovoj situaciji, zadatak najvišeg prioriteta je prvi. Opisani tok izvršavanja se nadalje ponavlja,

što dovodi do toga da treći zadatak nikad ne zauzme muteks. Ovaj problem se naziva "gladovanje" (eng. *starvation*), gde zadatak ne može da dobije kontrolu nad resursom.

Primena uslovnih promenljivih u ArdOS

Primer 22.4.4. Implementirati aplikaciju koja realizuje memorijski bafer sa mogućnošću skladištenja četiri bajta. Omogućiti neprestano dodavanje vrednosti u bafer u opsegu od 0 do 11 i istovremeno uklanjanje elemenata. Upotrebom funkcije *OSSleep()*, testirati različite situacije u kojima se bafer može naći. Dodatno, omogućiti treperenje diode frekvencijom od 1Hz.

Rešenje:

Kako je potrebno realizovati istovremeno izvršavanje tri funkcionalnosti, biće neophodno iskoristiti tri zadatka. Prva dva zadataka će biti namenjena pristupu baferu. Prvi zadatak će implementirati dodavanje novih elemenata u bafer, a drugi uklanjanje elemenata. Prilikom svakog dodavanja i uklanjanja, ispisuje se poruka putem serijskog porta o tome koja operacija je izvršena i koja je vrednost elementa. Kako je bafer specifičan resurs, potrebno je obezbediti da u jednom trenutku samo jedan zadatak može da mu pristupi. Ovo se postiže upotrebom muteksa. Međutim, problem se javlja ukoliko muteks postane zauzet od strane zadatka koji uklanja elemente iz bafera, a ne postoji ni jedan element u njemu. U ovoj situaciji potrebno je omogućiti čekanje na upis prvog elementa. Čekanje se obezbeđuje upotrebom petlje koja proverava da li je unet novi element. Međutim, budući da je ovaj zadatak zauzeo muteks, drugi zadatak koji treba da doda novi element neće moći to da izvrši. Kako bi se ovaj problem prevazišao, upotrebljavaju se uslovne promenljive. Naime, u okviru petlje, koja proverava da li je bafer prazan, smešta se funkcija koja čeka na pojavu signala. Pozivom ove funkcije, muteks se oslobađa, a zadatak koji uklanja element iz bafera se zaustavlja. Istovremeno se omogućava pokretanje zadatka koji dodaje element u bafer. Kada novi element bude dodat, šalje se signal zadatku koji čeka na upis prvog elementa. On biva aktiviran, i omogućen mu je dalji nastavak rada. Sličan postupak je realizovan i za problem dodavanja novog elementa ukoliko je bafer popunjen. U okviru ovog primera su upotrebljeni muteks, koji omogućava jedinstven pristup baferu, i dve uslovne promenljive: jedna za signalizaciju da je novi element dodat, a druga koja signalizira da je element uklonjen. Prva se upotrebljava prilikom provere da li je bafer popunjen, dok se druga koristi za proveru da li je bafer prazan. Rešenje zadatka je prikazano na listingu 22.14.

```
#include <stdio.h>
#include "kernel.h"
#include "mutex.h"
#include "pin.h"
#include "usart.h"

// Kreiranje objekata za muteks i uslovne promenljive
```

```
OSMutex mutex;
OSCond cv_full, cv_empty;

// Kreiranje buffera
#define BUFFER_SIZE 4
int8_t buffer[BUFFER_SIZE];
uint8_t head_ptr = 0, back_ptr = 0;
uint8_t count = 0;

// Deklaracije funkcija za upravljanje baferom
void add(int8_t c);
int8_t remove();

void task1(void *p)
{
    int8_t msg[64];
    int8_t i;

    while(1)
    {
        for(i = 0; i < 12; i++)
        {
            OSTakeMutex(&mutex); // Zauzimanje muteksa

            // Cekanje na pojavu signala da je element
                uklonjen
            while(count == BUFFER_SIZE)
            OSWait(&cv_full, &mutex);

            add(i); // Dodavanje elementa u bafer

            // Ispis dodate vrednosti
            sprintf(msg, "Added␣%d\r\n", i);
            usartPutString(msg);

            OSSignal(&cv_empty); // Signali da je element
                dodat

            OSGiveMutex(&mutex); // Oslobadjanje muteksa

            OSSleep(200); // Pauza za potrebe testiranja
        }

        OSSleep(2000); // Pauza za potrebe testiranja
    }
}

void task2(void *p)
{
    int8_t c;
```

```
    int8_t msg[64];
    int8_t i = 0;

    OSSleep(3000);  // Pauza za potrebe testiranja

    while(1)
    {
        OSTakeMutex(&mutex); // Zauzimanje muteksa

        // Cekanje na pojavu signala da je element dodat
        while(count == 0)
        OSWait(&cv_empty, &mutex);

        c = remove(); // Uklanjanje elementa iz bafera

        // Ispis uklonjenje vrednosti
        sprintf(msg, "Removed␣%d\r\n", c);
        usartPutString(msg);

        OSSignal(&cv_full); // Signali da je element uklonjen

        OSGiveMutex(&mutex); // Oslobadjanje muteksa

        // Pauze za potrebe testiranja
        i++;
        if(i % 7 == 0)
            OSSleep(2000);

        OSSleep(100);
    }
}

void task3(void *p)
{
    // Treperenje diode
    while(1)
    {
        pinSetValue(PORT_B, 5, HIGH);
        OSSleep(500);
        pinSetValue(PORT_B, 5, LOW);
        OSSleep(500);
    }
}

int16_t main()
{
    // Inicijalizacija pina koji upravlja LED diodom i
        serijske komunikacije
    pinInit(PORT_B, 5, OUTPUT);
    usartInit(115200);
```

```
    OSInit(3); // Inicijalizacija operativnog sistema sa tri
        zadatka

    // Inicijalizacija muteksa i uslovnih promenljivih
    OSCreateMutex(&mutex);
    OSCreateConditional(&cv_full);
    OSCreateConditional(&cv_empty);

    // Inicijalzziacija zadataka
    OSCreateTask(0, task1, 0);
    OSCreateTask(1, task2, 0);
    OSCreateTask(2, task3, 0);

    OSRun(); // Pokretanje operativnog sistema

    while(1);
}

// Definicija funkcije za dodavanje elementa u bafer
void add(int8_t c)
{
    buffer[head_ptr] = c;
    head_ptr = (head_ptr + 1) % BUFFER_SIZE;
    count++;
}

// Definicija funkcije za uklanjanje elementa iz bafera
int8_t remove()
{
    int8_t tmp = buffer[back_ptr];
    back_ptr = (back_ptr + 1) % BUFFER_SIZE;
    count--;

    return tmp;
}
```

Listing 22.14: Uslovne promenljive u ArdOS

Pokretanjem aplikacije, čiji je kod prikazan na listingu 22.14, dobijaju se rezultati dati na listingu 22.15.

```
< Inicijalna pauza u uklanjanju duzine 3s
Added 0 -
Added 1  |
Added 2  | Dodavanje novih elemenata svakih 200ms
Added 3 -
< Bafer popunjen
Removed 0 - Uklanjanje aktiviran
Added 4    |
Removed 1  |
Removed 2  |
```

```
Added 5    | Naizmenico dodavanje i uklanjanje
Removed 3  |
Removed 4  |
Added 6    |
Removed 5  |
Removed 6 -
< Pauza u uklanjanju duzine 2s
Added 7  -
Added 8   |
Added 9   | Dodavanje novih elemenata svakih 200ms
Added 10 -
< Bafer popunjen
Removed 7
Added 11
< Pauza u dodavanju duzine 2s
Removed 8  -
Removed 9   |
Removed 10  | Uklanjanje elemenata svakih 100ms
Removed 11 -
< Bafer prazan
Added 0    -
Removed 0   | Naizmenico dodavanje i uklanjanje
```

Listing 22.15: Ispis primera za rad sa uslovnim promenljivama

Primena FIFO redova u ArdOS

Primer 22.4.5. Implementirati aplikaciju koja omogućava unos parametara sesije treperenja u narednom formatu.

`<num of pulses>X<pulse period>`

Parametar `<num of pulses>` predstavlja broj treptaja koje je potrebno realizovati, a `<pulse period>` periodu kojom je potrebno izvršiti treperenje. Omogućiti dodavanje novih sesija i pre isteka već dodatih. Maksimalan broj treptaja u sesiji ograničiti na 9, a maksimalnu dužinu periode treperenja na 2s.

Rešenje:

Kako je potrebno realizovati istovremeno izvršavanje dve funkcionalnosti, biće neophodno iskoristiti dva zadatka. Prvi zadatak će biti namenjen realizaciji učitavanja parametara sesija treperenja preko serijskog porta. Funkcija koja će biti povezana sa ovim zadatkom je *task1*. Drugi zadatak će biti realizovan u okviru funkcije *task2*, koja će biti zadužena za implementaciju pojedinačnih sesija treptanja. Podaci o sesijama treptaja će biti prosleđeni od strane prvog zadatka ka drugom pomoću FIFO strukture. Kako je potrebno proslediti dva podatka, jedan koji predstavlja broj, i drugi koji predstavlja period treperenja, red treba da prenosi podatke koji nose dva značenja. Prema tome, potrebno je voditi računa o tome kojim redosledom se podaci postavljaju i uklanjaju iz reda. Na primer, ukoliko je, u okviru prvog zadatka, prvo postavljen broj perioda, pa tek nakon toga dužina

periode, tada se isti redosled mora primeniti i prilikom uklanjanja iz reda. Ovo znači da je prvi podatak koji se ukloni broj periode, a drugi dužina periode. Alternativni način realizacije podrazumeva upotrebu dva reda: jednog za broj treptaja i drugog za dužinu periode treperenja. Rešenje zadatka je prikazano na listingu 22.16.

```
#include "kernel.h"
#include "queue.h"
#include "pin.h"
#include "usart.h"

// Kreiranje objekata za red
#define QLEN 20
int16_t qbuff[QLEN];
OSQueue queue;

void task1(void *p)
{
    int16_t num_of_pulses;
    int16_t pulse_period;
    int8_t tmp[128];

    while(1)
    {
        // Ucitavanje poruke
        do
        {
            usartPutString("Start␣pulsing␣by␣typing␣\"<
                num_of_pulses>X<pulse_period>\"!\r\n");
            while(usartAvailable() == 0)
                OSSleep(10);
            OSSleep(100);

            num_of_pulses = usartParseInt();
            pulse_period = usartParseInt();

        } while((num_of_pulses <= 0) || (num_of_pulses > 9) ||
             (pulse_period <= 0) || (pulse_period > 2000));

        // Dodavanje parametara treptanja u red
        OSEnqueue(num_of_pulses, &queue);
        OSEnqueue(pulse_period, &queue);

        // Ispis poruke o uspesnom dodavanju treptanja
        sprintf(tmp, "Added␣%d␣pulses␣with␣period␣%dms!\r\n",
            num_of_pulses, pulse_period);
        usartPutString(tmp);
    }
}
```

```
void task2(void *p)
{
    int16_t num_of_pulses;
    int16_t pulse_period;
    uint8_t i;

    while(1)
    {
        // Uklanjanje parametara treptanja iz reda
        num_of_pulses = OSDequeue(&queue);
        pulse_period = OSDequeue(&queue);

        // Treperenje diode
        for(i = 0; i < num_of_pulses; i++)
        {
            pinSetValue(PORT_B, 5, HIGH);
            OSSleep(pulse_period/2);
            pinSetValue(PORT_B, 5, LOW);
            OSSleep(pulse_period/2);
        }
    }
}

int16_t main()
{
    // Inicijalizacija pina koji upravlja LED diodom i
        serijske komunikacije
    pinInit(PORT_B, 5, OUTPUT);
    usartInit(115200);

    OSInit(2); // Inicijalizacija operativnog sistema sa dva
        zadatka

    // Inicijalizacija reda
    OSCreateQueue(qbuff, QLEN, &queue);

    // Inicijalizacija zadataka
    OSCreateTask(0, task1, 0);
    OSCreateTask(1, task2, 0);

    OSRun(); // Pokretanje operativnog sistema

    while(1);
}
```

Listing 22.16: FIFO redovi u ArdOS

Primena prioritetnih redova u ArdOS

Primer 22.4.6. Implementirati aplikaciju koja omogućava unos parametara sesije treperenja u narednom formatu.

```
<type><num of pulses>X<pulse period>
```

Parametar `<num of pulses>` predstavlja broj treptaja koji je potrebno realizovati, a `<pulse period>` periodu kojom je potrebno vršiti treperenje. Parametar `<type>` predstavlja prioritet izvršavanja sesije i može imati vrednost 'P' ili 'S'. Oznaka 'P' simbolizuje da je sesija prioritetna, a 'S' da je standardna. Takođe, potrebno je omogućiti dodavanje novih sesija i pre isteka već dodatih. Maksimalan broj treptaja u sesiji ograničiti na 9, a maksimalnu dužinu periode treperenja na 2s.

Rešenje:

Kako je potrebno realizovati istovremeno izvršavanje dve funkcionalnosti, biće neophodno iskoristiti dva zadatka. Prvi zadatak će biti namenjen realizaciji učitavanja parametara sesija treptanja putem serijskog porta. Funkcija koja će biti povezana sa ovim zadatkom je *task1*. Drugi zadatak će biti realizovan u okviru funkcije *task2*, koja će biti zadužena za implementaciju pojedinačnih sesija treptanja. Podaci o sesijama treptaja će biti prosleđeni od strane prvog zadatka ka drugom pomoću prioritetnih redova. Kako je potrebno proslediti dva podatka po prioritetu, u okviru ovog primera će biti upotrebljena dva reda za prenos pojedinačnih podataka. Ukoliko je sesija označena kao prioritetna, podatak će biti dodat u odgovarajući red sa prioritetom 0, dok, ukoliko je sesija standardna, prioritet prilikom dodavanja će biti 1. Rešenje zadatka je prikazano na listingu 22.17.

```
#include "kernel.h"
#include "queue.h"
#include "pin.h"
#include "usart.h"

// Kreiranje objekata za redove
#define QLEN 8
TPrioNode qbuff_num_of_pulses[QLEN];
TPrioNode qbuff_pulse_period[QLEN];
OSQueue queue_num_of_pulses;
OSQueue queue_pulse_period;

void task1(void *p)
{
    int16_t num_of_pulses;
    int16_t pulse_period;
    int8_t tmp[128];
    int8_t priority;

    while(1)
```

```
    {
        // Ucitavanje poruke
        do
        {
            usartPutString("Start␣pulsing␣by␣typing␣\"<type><
                num_of_pulses>X<pulse_period>\"!\r\n");
            while(usartAvailable() == 0)
                OSSleep(10);
            OSSleep(100);

            priority = usartGetChar();
            num_of_pulses = usartParseInt();
            pulse_period = usartParseInt();

        } while((num_of_pulses <= 0) || (num_of_pulses > 9) ||
             (pulse_period <= 0) || (pulse_period > 2000) || ((
            priority != 'S') && (priority != 'P')));

        // Dodavanje parametara treptanja u redove
        if(priority == 'P')
        {
            OSPrioEnqueue(num_of_pulses, 0, &
                queue_num_of_pulses);
            OSPrioEnqueue(pulse_period, 0, &queue_pulse_period
                );
            // Ispis poruke o prioritetnom dodavanju treptanja
            sprintf(tmp, "Added␣priority␣%d␣pulses␣with␣period
                ␣%dms!\r\n", num_of_pulses, pulse_period);
            usartPutString(tmp);
        }
        else
        {
            OSPrioEnqueue(num_of_pulses, 1, &
                queue_num_of_pulses);
            OSPrioEnqueue(pulse_period, 1, &queue_pulse_period
                );
            // Ispis poruke o standardnom dodavanju treptanja
            sprintf(tmp, "Added␣standard␣%d␣pulses␣with␣period
                ␣%dms!\r\n", num_of_pulses, pulse_period);
            usartPutString(tmp);
        }
    }
}

void task2(void *p)
{
    int16_t num_of_pulses;
    int16_t pulse_period;
    uint8_t i;
```

```
    while(1)
    {
        // Uklanjanje parametara treptanja iz redova
        num_of_pulses = OSDequeue(&queue_num_of_pulses);
        pulse_period = OSDequeue(&queue_pulse_period);

        // Treperenje diode
        for(i = 0; i < num_of_pulses; i++)
        {
            pinSetValue(PORT_B, 5, HIGH);
            OSSleep(pulse_period/2);
            pinSetValue(PORT_B, 5, LOW);
            OSSleep(pulse_period/2);
        }
    }
}

int16_t main()
{
    // Inicijalizacija pina koji upravlja LED diodom i
        serijske komunikacije
    pinInit(PORT_B, 5, OUTPUT);
    usartInit(115200);

    OSInit(2); // Inicijalizacija operativnog sistema sa dva
        zadatka

    // Inicijalizacija reda
    OSCreatePrioQueue(qbuff_num_of_pulses, QLEN, &
        queue_num_of_pulses);
    OSCreatePrioQueue(qbuff_pulse_period, QLEN, &
        queue_pulse_period);

    // Inicijalzziacija zadataka
    OSCreateTask(0, task1, 0);
    OSCreateTask(1, task2, 0);

    OSRun(); // Pokretanje operativnog sistema

    while(1);
}
```

Listing 22.17: Prioritetni redovi u ArdOS

22.5 Zadaci za vežbu

Zadatak 22.5.1. Napisati program koji implementira komunikaciju između dva zadatka upotrebom *steka* (eng. *stack*), odnosno niza podataka koji se ponaša po principu LIFO (eng. *Last Input First Output*). Za upotrebu niza implementirati funkcije za dodavanje i uklanjanje elemenata. Pristup steku ograničiti na način da, u jednom trenutku, samo jedan zadatak može da izvrši dodavanje ili uklanjanje elementa. Konačno, testirati implementaciju steka tako što će u jednom zadatku biti implementirano neprestano dodavanje novih elemenata, a u drugom njihovo uklanjanje i prikaz putem serijskog porta.

Zadatak 22.5.2. Napisati program koji omogućava slanje tri uzastopne poruke svakih 200ms preko serijskog porta od strane šest različitih zadataka.

Zadatak 22.5.3. Napisati program koji omogućava ispisivanje proizvoljno generisanih vrednosti i vrednosti dobijenih učitavanjem sa nekog od analognih ulaza mikrokontrolera. Vrednosti generisati i učitavati periodično u okviru posebnih zadataka. Takođe, kreirati poseban zadatak koji će očitane vrednosti ispisivati putem serijskog terminala.

Zadatak 22.5.4. Modifikovati Zadatak 22.5.3 na način da je slanje vrednosti učitane sa analognog ulaza višeg prioriteta.

Zadatak 22.5.5. Napisati program koji generiše proizvoljan broj u opsegu od 0 do 1000. Ukoliko je dobijena vrednost deljiva sa 57, *watchdog* tajmer, prethodno konfigurisan tako da kreira prekid usled isteka vremenskog intervala od jedne sekunde, će biti resetovan. U slučaju da broj nije deljiv, novi broj će biti generisan nakon 50ms. Istek brojanja tajmera će prouzrokovati deset treptaja diode frekvencijom od 5Hz propraćenih slanjem poruka putem serijskog porta.

Zadatak 22.5.6. Napisati program koji implementira sledeće funkcionalnosti na PLS7 ekspanzionoj ploči. Omogućiti ispisivanje karaktera A, b, C, D na prvom, drugom, trećem i četvrtom sedmo-segmentnom displeju ukoliko su pritisnuti tasteri LEFT, DOWN, RIGHT i UP redom, dok je, u suprotnom, potrebno ispisati simbol – na odgovarajućem displeju. Za implementaciju ovog dela funkcionalnosti *ne treba koristiti* prekid usled promene stanja pina. Istovremeno, omogućiti sledeću sekvencu paljenja dioda, gde se prelazi između dva stanja dešavaju periodično, na svakih pola sekunde.

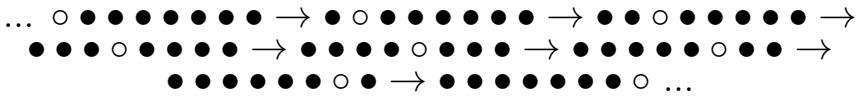
… ○ ● ○ ● ○ ● ○ ● → ● ○ ● ○ ● ○ ● ● → ○ ● ○ ● ○ ● ○ ● → …

○ – dioda je uključena ● – dioda je isključena

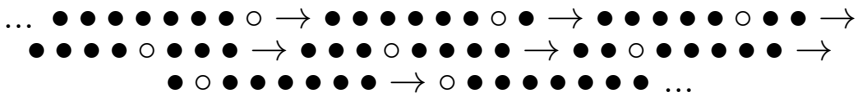
Zadatak 22.5.7. Modifikovati Zadatak 22.5.6 tako da se za detekciju pritiska tastera upotrebljava prekid usled promene stanja pina. Realizovati signalizaciju od strane prekidne rutine odgovarajućem zadatku koji treba da odredi koji je taster pritisnut i na osnovu toga promeni ispis na sedmo-segmentnim displejima.

Zadatak 22.5.8. Napisati program koji implementira sledeće funkcionalnosti na PLS7 ekspanzionoj ploči. Omogućiti naizmenično smenjivanje cifara od 0 do 9 na desnom sedmo-segmentnom displeju. Pored ovoga, implementirati u dva različita zadatka istovremeno paljenje dioda sledećim sekvencama, gde se prelazi između dva stanja dešavaju periodično, na svakih 100ms.

Prva sekvenca:



Druga sekvenca:



○ – dioda je uključena ● – dioda je isključena

Zadatak 22.5.9. Modifikovati Zadatak 22.5.8 tako da u jednom trenutku može biti aktivna samo jedna sekvenca.

Zadatak 22.5.10. Napisati program koji implementira sledeće funkcionalnosti na PLS7 ekspanizonoj ploči. Pritiskom tastera LEFT potrebno je učitati vrednost podešenu pomoću prekidača. Ovu vrednost je neophodno prikazati na sedmo-segmentnom displeju u decimalnom i na diodama u binarnom formatu. Prikazivanje na sedmo-segmentnom displeju i diodama traje minimalno 2s, nakon čega se prikazuje naredna vrednost, ukoliko je učitana. Prikazivanje vrednosti na displeju i diodama realizovati u posebnim zadacima. Omogućiti učitavanje novih vrednosti u svakom trenutku.

Zadatak 22.5.11. Modifikovati Zadatak 22.5.10 tako da u slučaju pritiska tastera RIGHT dolazi do prioritetnog prikazivanja učitane vrednosti sa prekidača.

Zadatak 22.5.12. Napisati program koji implementira sledeću funkcionalnosti na PLS7 ekspanzionoj ploči. Kreirati FIFO bafer *mod_fifo* koji treba da skladišti enumerisani tip podataka *mod_t* koji može da ima vrednosti `LOW_POWER`, `NORMAL_POWER`, `HIGH_POWER` ili `DANGEROUS_POWER`. Postaviti *watchdog* tajmer tako da broji 10s, nakon čega se izvršava sistemski reset. U okviru različitih zadataka potrebno je implementirati funkcionalnosti date u nastavku.

1. Očitavati vrednost koja se nalazi na analognom ulazu 4, frekvencijom od 1Hz. Vrednost proslediti zadatku 5 u sledećem formatu:
   - `4b` – zaglavlje poruke koje ima vrednost 0101;
   - `2b` – biti imaju vrednost 0;

- `10b` – vrednost učitana sa analognog ulaza 4

gde `4b`, `2b` i `10b` označavaju redom 4, 2 i 10 bita za reprezentaciju određenih vrednosti, što u zbiru daje 16 bita za ovaj format poruke.

2. Očitavati vrednost koja se nalazi na analognom ulazu 5 frekvencijom od 20Hz. Vrednost proslediti zadatku 5 u sledećem formatu:
   - `4b` – zaglavlje poruke koje ima vrednost 1001;
   - `2b` – biti imaju vrednost 0;
   - `10b` – vrednost učitana sa analognog ulaza 5.
3. Učitati vrednost prekidača frekvencijom 2Hz. Vrednost proslediti zadatku 5 u sledećem formatu:
   - `4b` – zaglavlje poruke koje ima vrednost 1101;
   - `2b` – biti imaju vrednost 0;
   - `10b` – vrednost prekidača.

   U zavisnosti od opsega u kojem se vrednost nalazi, potrebno je izvršiti sledeće:
   - 0 - 44 – dodati vrednost `LOW_POWER` u *mod_fifo*, resetovati *watchdog* tajmer i aktivirati signal *s_mod*;
   - 45 - 179 – dodati vrednost `NORMAL_POWER` u *mod_fifo*, resetovati *watchdog* tajmer i aktivirati signal *s_mod*;
   - 180 - 224 – dodati vrednost `HIGH_POWER` u *mod_fifo*, resetovati *watchdog* tajmer i aktivirati signal *s_mod*;
   - 225+ – dodati vrednost `DANGEROUS_POWER` u *mod_fifo* i aktivirati signal *s_mod*; nakon toga prioritetno proslediti zadatku 5 poruku o pojavi greške u sistemu, u sledećem formatu:
     - `4b` – zaglavlje poruke koje ima vrednost 1110;
     - `12b` – biti imaju vrednost 1.
4. Ukoliko *mod_fifo* nije prazan, učitati vrednost, nakon čega je potrebno, u zavisnosti od moda rada, izvršiti sledeće:
   - diode LED7, LED6, LED5 i LED4 podesiti na sledeći način, nakon čega je potrebno čekati na pojavu signala *s_mod* kako bi se ponovo učitala vrednost iz bafera;

     ○ ● ● ●

     ○ – dioda je uključena ● – dioda je isključena
   - diode LED7, LED6, LED5 i LED4 podesiti na sledeći način, nakon čega je potrebno čekati na pojavu signala *s_mod* kako bi se ponovo učitala vrednost iz bafera;

○ ○ ● ●

○ – dioda je uključena ● – dioda je isključena

- diodama LED7, LED6, LED5 i LED4 implementirati sledeću sekvencu frekvencijom od 5Hz, nakon čega je potrebno učitati novu vrednost iz *mod_fifo*, ukoliko je dostupna;

○ ○ ○ ● → ● ● ● ●

○ – dioda je uključena ● – dioda je isključena

- diodama LED7, LED6, LED5 i LED4 implementirati sledeću sekvencu frekvencijom od 10Hz nakon čega je potrebno učitati novu vrednost iz *mod_fifo* ukoliko je dostupna;

○ ○ ○ ○ → ● ● ● ●

○ – dioda je uključena ● – dioda je isključena

5. Prosleđivati poruke koje prima od ostalih zadataka na serijski port. Pri tome, potrebno je obratiti pažnju da su poruke 16-bitne, a da se putem serijskog porta podaci šalju bajt po bajt.