Spis treści

Rozdział 1

Wprowadzenie

Program **SRTA** wraz z programami **incom1** i **outcom1** służy do komunikacji ze sterownikiem silnika, który wykorzystywany jest w teleskopie astronomicznym w osi kąta godzinnego. Programy **nadaj**, **odbor** i **serwis-SRTA** służą do komunikacji z programem **SRTA** w czynnościach serwisowych.

Na rysunku 1 pokazany jest schemat ukazujący hierarchie między programami **SRTA**, **incom1** i **outcom1** oraz mechanizmy komunikacji między nimi.

Wszystkie katalogi posiadają wspólną ścieżkę gdzie znajduje się program **<ścieżka do programu>**. W tym miejscu znajdują się następujące katalogi:

bin – katalog zawiera pliki wykonywalne: **SRTA**, **incom1**, **outcom1**, **nadaj-SRTA**, **odbior** i **serwis-SRTA**.

etc – katalog zawierający pliki konfiguracyjne: **SRTA.conf** i **pozycje.conf**,

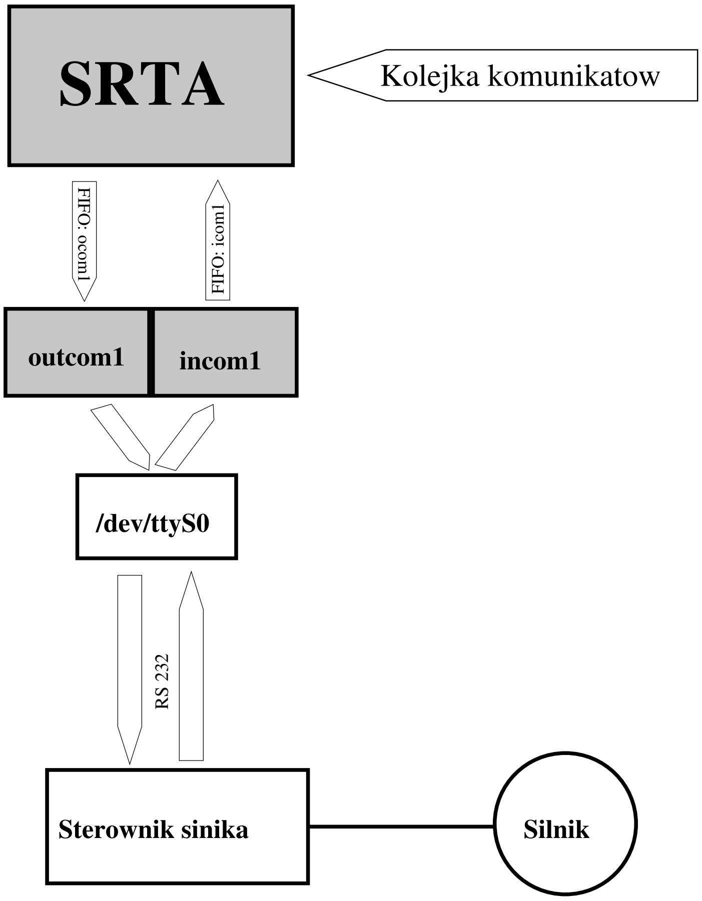
fifo – katalog zawiera kolejki FIFO: **icom1** i **ocom1**,

log – katalog zawiera logi programu: **SRTA.log**,

tmp – katalog zawiera pliki tymczasowe takie jak **pozycjaSRTA.tmp**, **baza-SRTA.tmp** i **braklacznosci.tmp**,

doc – katalog zawierający ten dokument.

W dalszej części omówione zostaną kody źródłowe poszczególnych programów oraz przedstawiony zostanie program znajdujący się w sterowniku silnika.



Rysunek 1.1: Schemat komunikacji programów ze sterownikiem

Rozdział 2

Pliki w katalogu <ścieżka do programu>/etc

2.1 SRTA.conf

Plik konfiguracyjny SRTA.conf :

```
5             /*Czas_oczekiwania_na_odpowiedz_sterownika_[s]_*/
10            /*Czas_po_jakim_program_sprawdza_lacznosc_ze_sterownikiem_[s]_*/
16385         /*Ilosc_incramentow_na_obrot_silnika*/
7200          /*Ilosc_sekund_luku_na_obrot*/
36            /*Nr_lini_zawierajacej_ilosc_incrementow_przy_poprawkach_z_autoguidera*/
32            /*Poczatek_podprogramu_do_poprawek_z_autoguidera*/
9             /*Poczatek_podprogramu_do_ruchu_dobowego_(sledzenie)*/
0             /*Linia_bezczynoosci_napedow*/
28            /*Nr_lini_zawierajacej_ilosc_incrementow_przy_najezdzie_na_pozycje*/
24            /*Poczatek_podprogramu_do_najazdu_na_pozycje*/
1             /*Zapis_logow_1-wlaczone_0-wylaczone*/
500           /*Przyspieszenie_i_opoznienie_silnikow_zadane_w_BIOSIE*/
500           /*Maksymalna_predkosc_silnikow_zadana_w_BIOSIE*/
56            /*Poczatek_podprogramu_do_autoguider_spowolniony_ruch_dobowy*/
41            /*Poczatek_podprogramu_do_autoguider_przyszpieszony_ruch_dobowy*/
```

2.2 poprawki.conf

Plik zawiera pozycje α i δ teleskopu, gdy znajduje on sie na bazach w obu osiach.

```
0   0  1.0    /*kat_godzinny_[godz_min_sek]*/
23 23 22.1    /*deklinacja___[st._min_sek] */
```

Rozdział 3

Pliki w katalogu <ścieżka do programu>/tmp

3.1 bazaSRTA.tmp

Plik zawiera pozycje bazy

```
0       279.614115490376             19066.1875       278.614115490376
```

zapamiętaną w czterech parametrach

1. znacznik bazowania (0 - bazowanie aktualne),
2. pozycja bazy w sekundach łuku,
3. pozycja bazy w inkramentach silnika,
4. poprawka do pozycji (zmienna **POPRAWKAPOZ** w programie **SRTA**).

3.2 pozycjaSRTA.tmp

Plik zawiera aktualną pozycje silnika (teleskopu)

```
      9015791.25297529              9779982       20517186.0666667
```

zapamiętaną w trzech parametrach

1. pozycja ze wszystkimi poprawkami,
2. pozycja bez poprawek,
3. pozycja ze wszystkimi poprawkami w inkramentach.

3.3 braklacznosci.tmp

Plik zawiera wartość 0, gdy łączność ze sterownikiem jest prawidłowa lub 1 gdy nie jest prawidłowa.

Rozdział 4

Program SRTA

4.1 Nagłówki

Program SRTA korzysta z plików nagłówkowych:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
#include <unistd.h>
~#include <signal.h>
#include <time.h>
#include <stdlib.h>
#include <math.h>
```

Struktury zawarte w plikach nagłówkowych **ipc.h msg.h** wykorzystane są do komunikacji między procesami (kolejka komunikatów), **termios.h** do portu szeregowego, **signal** do alarmów a **time.h** do obslugi czasu w programie.

4.1.1 Definicje stałych

W programie zdefiniowano kilka pomocnych stałych. Prędkość transmisji portu szeregowego zgodnie ze specyfikacją techniczną parametrów transmisji sterowników silników

```
#define BAUDRATE B19200
```

Nazwa urządzenia w Unixie odpowiadającemu portowi COM1

```
#define MODEMDEVICE0 "/dev/ttyS0"
```

Wartości dla stanu FALSE, TRUE i ZERO

```
#define FALSE 0
#define TRUE 1
#define ZERO 0
```

Wartości odpowiadające bajtom potwierdzających transmisje i stany logiczne portów w komunikacji między programem a sterownikami

```
#define ESC '\x1B'
#define ACK '\x06'
#define NAK '\x15'
#define TOUT '\x16'
#define CAN '\x18'
```

Nazwy i wartości komunikatów ESC, ACK, NAK, TOUT, CAN są zgodne ze specyfikacją techniczną protokołu komunikacyjnego silników.

4.1.2 Deklaracje funkcji pomocniczych

Program wykorzystuje szereg funkcji pomocniczych:

```
void setflag(int sig);
int zablokowaniesilnikow(int fd1, int fd2);
int odblokowaniesilnikow(int fd1, int fd2);
int pozycja(int fd1, int fd2);
int logowaniedosilnikow(int fd1,int fd2);
int ruchdobowy(int fd1,int fd2,int par);
int kontrolalacznosci(int fd1, int fd2);
void zgloszeniestanu(int mqId, int ROZKAZ, int STAN);
int fatal( char *s);
void logi(char *tekst);
int najazdnapoz(int fd1,int fd2);
int poprawkapoz(int fd1,int fd2);
void punktbazowy(int fd1,int fd2);
```

Funkcje te zostaną opisane w następnych podrozdziałach.

4.1.3 Deklaracja strumieni wejscia/wyjscia

```
FILE *output, *input;
```

4.1.4 Struktury dla kolejki komunikatów

Program wykorzystuje komunikacje między procesami opartą o koleke komunikatów. „Podrasowanie” zmiennej

```
volatile int STOP=FALSE;
```

oraz struktura samej kolejki komunikatów

```
struct msqbuff
{
     long mtype;
     char mtext[300];
} mbuf;
```

mtype jest urzywana do klasyfikowania komunikatów. Program odbiera komunikaty, które posiadają przydzielony programowi wartość zawartej w zmiennej **PID**..
mtext zawiera tekst samego komunikatu.

4.1.5 Zmienne globalne

Program wykorzystuje szereg zmiennych globalnych.

```
int STAN,STAN1, LACZNOSC,fd;
int ROZKAZ;
int ALARMTIME=2;
int ALARMKONTROLA=3;
int alarm_flag=FALSE;
pid_t pid,pid1,pid2,pid3,pid4,pid5;
key_t klucz;
int mqId;
char komunikat[80];
int KTO=1;
int PID=10;
char BCC;
double PARAMETR1,PARAMETR2,POPRAWKAPOZ,PAR2,PAR1;
double INCRA,SEKLUKU,SEKLUKU,GODZ,MIN,SEK,POCZATEK,ZEROINCR,DGODZ,DMIN,DSEK;
int LINIAPA,LINIARD,LINIAPOZ,LINIASTOP,LINIANNPOZ,LINIANNP,LINIACZAS,LOGI,ZNACZNIK;
double ACCELERACJA,VMAX,BIAS,apoz,ipozbezpop,aipoz;
int LINIARDPLUS,LINIARDMINUS;
```

4.2 Główny program

Główny program obsługuje kolejke komunikatów i komunikuje się ze sterownikami silników wykorzystując programy zewnętrzne (incom1 i ocom1) oraz procedury pomocnicze.

4.2.1 Otwarcie i deklaracja zmiennych

```
int main()
{
  int fd1,fd2;
  char buf[255],str[255],bzz;
  int err;
```

gdzie **fd1** i **fd2** sĄ wskaźnikami do kolejek FIFO.

4.2.2 Deklaracja struktury parametrów portu COM1 i alarmu

```
  struct termios oldtio;
  struct termios newtio;
```

Zmienne **oldtio newtio** i **oldtio1** wykorzystano do konfiguracji portu COM1 oraz zapamiętania i przywrócenia poprzedniej konfiguracji po zakończeniu programu. Następnie zdeklarowano strukturę do wywołania funkcji alarmowych.

```
  static struct sigaction act;
```

Zapisanie do pliku zawierającego logi programu (patrz podprogram **logi**).

```
  logi("#######################################################################");
  logi("######################### start programu ############################");
  logi("#######################################################################");
```

4.2.3 Wczytanie pliku konfiguracyjnego SRTA.conf

Plik konfiguracyjny znajduje sie w katalogu <scieżka do programu>/etc i nazywa się SRTA.conf .Plik ten opisany jest w następnym rozdziale.

```
logi("Wczytanie pliku konfiguracyjnego SRTA.conf");
if((input=fopen("../etc/SRTA.conf","r"))==NULL)
    {
    logi("Nie moge otworzyc pliku konfiguracyjnego SRTA.conf");
    exit(1);
    }
fscanf(input,"%d %s\n",&ALARMTIME,&bzz);
fscanf(input,"%d %s\n",&ALARMKONTROLA,&bzz);
fscanf(input,"%lf %s\n",&INCRA,&bzz);
fscanf(input,"%lf %s\n",&SEKLUKU,&bzz);
fscanf(input,"%d %s\n",&LINIAPOZ,&bzz);
fscanf(input,"%d %s\n",&LINIAPA,&bzz);
fscanf(input,"%d %s\n",&LINIARD,&bzz);
fscanf(input,"%d %s\n",&LINIASTOP,&bzz);
fscanf(input,"%d %s\n",&LINIANNPOZ,&bzz);
fscanf(input,"%d %s\n",&LINIANNP,&bzz);
fscanf(input,"%d %s\n",&LOGI,&bzz);/* 0-wylaczone, 1-wlaczone*/
fscanf(input,"%lf %s\n",&ACCELERACJA,&bzz);
fscanf(input,"%lf %s\n",&VMAX,&bzz);
fscanf(input,"%d %s\n",&LINIARDMINUS,&bzz);
fscanf(input,"%d %s\n",&LINIARDPLUS,&bzz);
fclose(input);
```

Z pliku konfiguracyjnego pobieramy informacje na temat:

- **ALARMTIME** – czas oczekiwania na odpowiedź sterownika w sekundach,
- **ALARMKONTROLA** – czas po jakim sprawdza sie łaczność ze sterownikiem silników,
- **INCRA** – ilość inkramentów (kroków silnika) na 1 obrót osi silnika,
- **SEKLUKU** – ilość sekund łuku jaką zrobi oś teleskopu przy jednym obrocie silnika (czyli przełożenie przekładni slimakowej),
- **LINIAPOZ** – Numer linii programu znajdującego się w sterowniku silnika zawierający poprawki do pozycji podawane np. przez autoguider.
- **LINIAPA** – Numer linii, w której znajduje się początego programu (znajdującego się w sterowniku silnika) do najazdu na pozycje zawartą w **LINIAPOZ**. Po wykonaniu najazdu program znajdujący się w sterowniku silnika wywołuje linię **LINIARD**,
- **LINIARD** – Numer linii w której znajduje się początek programu (znajdującego się w sterowniku silnika) odpowiedzialnego za ruch dobowy teleskopu.

- **LINIASTOP** – Numer linii, w której znajduje się począątego programu (znajdującego się w sterowniku silnika) odpowiedzialnego za zatrzymanie i postój silnika,
- **LINIANNPOZ** – Numer linii programu (znajdującego się w sterowniku silnika) zawierający poprawki do pozycji na którą ma najechać silnik.
- **LINIANNP** – Numer linii, w której znajduje się począątego programu (znajdującego się w sterowniku silnika) do najazdu na pozycje zawartą w **LINIANNPOZ**. Po wykonaniu najazdu program znajdujący się w sterowniku silnika wykonuje linię **LINIASTOP**,
- **LOGI** – włączenie (LOGI=1) lub wyłączenie (LOGI=0) szczegółowego zapisu logów do pliku <ścieżka do programu>/log/SRTA.log,
- **ACCELERACJA** – maksymalne przyspieszenie i opóźnienie silników wpisane w BIOS sterownika silnika,
- **VMAX** – maksymalna prędkość silników wpisana w BIOS sterownika silnika,
- **LINIARDMINUS** – Numer linii w której znajduje się początek programu (znajdującego się w sterowniku silnika) do ruchu dobowego z połową nominalnej prędkości.
- **LINIARDPLUS** – Numer linii w której znajduje się począątego programu (znajdującego się w sterowniku silnika) do ruchu dobowego z dwukrotną nominalną prędkością.

4.2.4 Początkowa poprawka i przesunięcie pozycji

Początkowo poprawka do pozycji ustawiona jest na 0 w dalszej części programu będzie pobrana z pliku <ścieżka do programu>/etc/poprawki.conf. **BIAS** jest w programie wprowadzony w celu ominięcia problemu przejścia przez ZERO na ujemne wartości. Zakładamy, że teleskop nie obróci się więcej niż 5 razy wokół osi.

```
POPRAWKAPOZ=0;
BIAS=6*360*60*60;
```

4.2.5 Ustalenie pozycji silnika

W czasie pracy programu zapisywana jest pozycja do pliku <ścieżka do programu>/tmp/pozycjaSRTA.tmp .

W przypadku gdy sterowniki silników pracują bez przerwy, kilkakrotne uruchamianie programu SRTA nie powoduje utraty informacji o położeniu.

```
if((input=fopen("../tmp/pozycjaSRTA.tmp","r"))==NULL)
    {
    logi("Nie moge otworzyc pliku pozycjaSRTA.tmp");
    exit(1);
    }
    fscanf(input,"%lf %lf %lf \n",&apoz,&ipozbezpop,&aipoz);
    fclose(input);
```

4.2.6 Wyzerowanie wartości parametrów STAN, LACZNOSC i PARAMETR2

```
STAN=0;
LACZNOSC=0;
PARAMETR2=0;
```

4.2.7 Nawiązywanie komunikacji ze sterownikiem silnika

Tworzenie kolejki icom1 i ocom1

```
logi("Tworzenie kolejej FIFO");
mkfifo("../fifo/icom1",0666);
mkfifo("../fifo/ocom1",0666);
```

Otwarcie urzadzenia ttyS0

```
logi("Otwarcie portu szeregowego /dev/ttyS0");
fd = open(MODEMDEVICE0, O_RDWR | O_NOCTTY );
if (fd <0) {perror(MODEMDEVICE0); exit(-1); }
```

Opcja **O_RDWR** oznacza, że możemy pisać i czytać port, a opcja **O_NOCTTY** oznacza, że otwórz ale bez kontrolującego terminala.

Ustawienie parametrow portu ttyS0

```
        logi("Ustawienie parametrow  portu szeregowego /dev/ttyS0");
```

Zapisanie dotychczasowej konfiguracji portu i wyzerowanie **newtio**

```
tcgetattr(fd,&oldtio);
bzero(&newtio, sizeof(&newtio));
```

Ustawienie szybkości transmisji na wejściu i wyjściu z portu

```
cfsetispeed(&newtio, BAUDRATE);
cfsetospeed(&newtio, BAUDRATE);
```

Ustawienie parametrów portu

```
  newtio.c_cflag |=( CLOCAL | CREAD);
  newtio.c_cflag |= PARENB;
  newtio.c_cflag &= ~PARODD;
  newtio.c_cflag &= ~CSIZE;
  newtio.c_cflag |= CS8;
```

czyli

- **CLOCAL** – kontrola transmisji jak w modemie,
- **CREAD** – czytanie znaków dopuszczone,
- **PARENB** – kontrola parzystości,
- **PARODD** – kontrola parzystości parzysta,
- **CSIZE** – rozpoznanie następnego parametru określającego rozmiar znaku,
- **CS8** – rozmiar znaku 8 bitów,

oraz

```
newtio.c_iflag = IGNPAR;
newtio.c_lflag = 0;
newtio.c_oflag = 0;
newtio.c_cc[VTIME]    = 0;
newtio.c_cc[VMIN]     = 1;
```

co oznacza, że

- **IGNPAR** – pomijanie znaków z błędem parzystości,
- **newtio.c_lflag = 0** – brak formatowania nadchodzących znaków,
- **newtio.c_oflag = 0** – brak formatowania wychodzących znaków,
- **newtio.c_cc[VTIME] = 0** – czas nie odgrywa roli,
- **newtio.c_cc[VMIN] = 1** – odczyt 1 znaku,

Następnie odbywa się przypisanie tych własności portu.

```
tcflush(fd, TCIFLUSH);
tcsetattr(fd,TCSANOW,&newtio);
```

Uruchamianie procesow potomnych incom1 i outcom1

```
        logi("Uruchomienie programu incom1");

switch(pid=fork())
{
case -1:
logi("[SSR] Nie moge uzyskac pid do incom1");
fatal("");
case 0:
execlp("./incom1",(char *)0);
logi("[SSR] Nie znalazlem lub nie moge uruchomic incom1");
fatal("");
break;
}
```

```
    logi("Uruchomienie programu outcom1");
switch(pid1=fork())
{
case -1:
logi("ERROR: Nie moge uzyskac pid do outcom1\n");
fatal("");
break;
case 0:
execlp("./outcom1",(char *)0);
logi("ERROR: Nie znalazlem lub nie moge uruchomic outcom1\n");
fatal("");
break;
}
```

Programy **incom1** i **outcom1** opisane będą w następnych rozdziałach.

4.2.8 Ustawienie alarmu

Program uruchamia funkcje **setflag** gdy dostanie sygnal **SIGALARM** wywołany po upływie czasu zadanego w funkcji **ALARM(czas)**

```
act.sa_handler = setflag;
sigaction(SIGALRM, &act, NULL);
```

4.2.9 Otwarcie kolejki komunikatów

```
logi("Otwarcie kolejki komunikatow");
klucz = ftok(".",'Q');
if((mqId=msgget((key_t) klucz, 0600 |IPC_CREAT)) == -1)
{
     fatal("[SSR] Nie moge utworzyc kolejki komunikatow\n");
     return -1;
}
```

Otwarcie kolejki komunikatów wymaga podania punktu, który będzie stanowił odniesienie wszystkich programów do konkretnej kolejki. Zakładamy, że takim punktem jest katalog bieżący ("."), co w konsekwencji wymaga uruchamiania wszystkich programów w jednym katalogu **<ścieżka do programów>/bin** . W czasie tworzenia kolejki dostajemy klucz **mqId**, który jest identyfikatorem kolejki w programie. Urzycie opcji **0600 | IPC_CREAT** pozwala utworzyć kolejkę jeśli jeszcze ona nie istnieje, z uprawnieniami **0600** lub otworzyć już istniejącą.

4.2.10 Otwarcie kolejek fifo

```
logi("Otwarcie kolejek FIFO: icom1,ocom1");
fd1=open("../fifo/icom1",O_RDWR);
fd2=open("../fifo/ocom1",O_RDWR);
```

Otwierane są dwie wcześniej stworzone kolejki z prawami **O_RDWR** co oznacza, że można pisać do kolejki i czytać z kolejki.

4.2.11 Sprawdzenie łączności ze sterownikiem i zakończenie gdy jej brak

```
        logi("Sprawdzenie lacznosci ze sterownikem");
LACZNOSC=kontrolalacznosci(fd1,fd2);
if(LACZNOSC==1)
    {
    zablokowaniesilnikow(fd1,fd2);
    odblokowaniesilnikow(fd1,fd2);
    logowaniedosilnikow(fd1,fd2);
    /*Ustalenie poprawek*/
    LACZNOSC=pozycja(fd1,fd2);
    POPRAWKAPOZ=apoz-PARAMETR2;
```

Do sprawdzenia łączności ze sterownikami wykorzystano funkcje pomocniczą **kontrolalacznosci**, którą opisano w dalszych podrozdziałach. Jeżeli łączność jest poprawna (**LACZNOSC==1**) to za pomocą funkcji pomocniczych **zablokowaniesilnkikow**, **odblokowaniesilnikow** i **logowaniesilnkikow** przygotowujemy sterownik i silniki do pracy. Sekwencja ta jest wymagana, ponieważ jeden sterownik po uruchomieniu wykazuje błąd, który zostaje usunięty przez tą procedurę. Następnie czytana jest pozycja silnika i obliczana poprawka do pozycji.

Jeżeli jest brak łączności wykonywana jest część programu znajdująca się na końcu programu głównego:

```
}
else
{
/* Zakonczenie procesow i zamkniecie kolejki fifo */
kill(pid1,SIGKILL);
kill(pid,SIGKILL);
close(fd1);
close(fd2);
/* Przestawienie portu com1 na poprzednie parametry */
tcsetattr(fd,TCSANOW,&oldtio);
logi("!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
logi("!!! Przerwanie startu:     brak lacznosci ze sterownikiem silnikow !!!");
logi("!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
}
return 0;
}
```

4.3 Pętla główna programu

```
    logi("Oczekiwanie na polecenia z kolejki komunikatow");
    for(;;)
    {
KTO=1;PARAMETR1=0;PARAMETR2=0;
alarm_flag=FALSE;
alarm(ALARMKONTROLA); /*alarm za n sekundy */
bzero(mbuf.mtext,sizeof(mbuf.mtext));
err=msgrcv(mqId, &mbuf, sizeof(mbuf.mtext), PID,0);
```

W pętli nieskończonej czytamy kolejkę komunikatów. W pierwszej kolejności ustawiamy zmienne oraz flagę alarmu **alarm_flag**. Następnie ustawiamy alarm na czas **ALARMKONTROLA** i czyścimy bufor **mbuf.mtext**. Czytamy kolejke komunikatów z opcjami **PID** i **0** co oznacza , że będziemy odbierać tylko komunikaty o znaczniku **PID**, natomiast **0** że będziemy w tym miejscu czekać na komunikat aż go otrzymamy lub dostaniemy sygnał **SIGALARM** (wtedy wykona się procedura **setflag**). Gdy przybędzie komunikat lub pojawi się przerwanie **SIGALARM** to zerujemy czas alarmu

```
alarm(0);
```

Następnie sprawdzamy czy nadszedł komunikat czy czekanie przerwał sygnał **SIGALARM**. Jeśli nastąpiło przerwanie oczekiwania to sprawdzamy połączenie ze sterownikiem silnika

```
if(alarm_flag == TRUE)
{
LACZNOSC=kontrolalacznosci(fd1,fd2);/
```

Kiedy nie mamy połączenia ze sterownikami silników to przerywamy program

```
if(LACZNOSC!=1)
{
kill(pid1,SIGKILL);
kill(pid,SIGKILL);
close(fd1);
close(fd2);
/* Przestawienie portu com1 na poprzednie parametry */
tcsetattr(fd,TCSANOW,&oldtio);
logi("!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
logi("!!!  Przerwanie programu: brak lacznosci ze sterownikiem silnikow !!!");
logi("!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
return 2;
}
}
```

Jeśli otrzymaliśmy komunikat to sprawdzamy czy to nie jest czasem błąd czytania kolejki

```
else
{
if(err==-1)
{
logi("Blad odczytu kolejki komunikatow");
perror("blad odczytu");
return 2;
}
```

Jeśli to nie jest błąd kolejki, to rozczytujemy otrzymany komunikat

```
else
{
strcpy(buf,mbuf.mtext);
sscanf(buf,"%d %d",&KTO,&ROZKAZ);
```

W dalszej części w zależności od otrzymanego komunikatu wykonujemy odpowiednie procedury.

4.4 Rozkazy z kolejki komunikatów

4.4.1 100 – Zablokowanie i odblokowanie silnika

Wraz z rozkazem przesyłana jest wartość **1** (zablokuj silnik) lub **0** (odblokuj silnik).

```
if(ROZKAZ==100)
    {
    sscanf(buf,"%d %d %lf",&KTO,&ROZKAZ,&PARAMETR1);
    if(PARAMETR1==1)
{
sprintf(str,"(100 od %d) Otrzymanie rozkazu zablokowania silnikow",KTO);
logi(str);
/* Zablokowanie silnikow */
LACZNOSC=zablokowaniesilnikow(fd1,fd2);
if(LACZNOSC==1)
    {
    /* Zgloszenie wykonania rozkazu */
    zgloszeniestanu(mqId,100,1);
    ROZKAZ=0;
    }
else
    {
    /* Zgloszenie nieprawidlaowosci */
    zgloszeniestanu(mqId,100,0);
    }
    ROZKAZ=0;
}
    if(PARAMETR1==0)
{
sprintf(str,"(100 od %d) Otrzymanie rozkazu odblokowania silnikow",KTO);
logi(str);
/* Odblokowanie silnikow */
LACZNOSC=odblokowaniesilnikow(fd1,fd2);
if(LACZNOSC==1)
    {
    /* Zgloszenie wykonania rozkazu */
    zgloszeniestanu(mqId,100,1);
    ROZKAZ=0;
    }
else
    {
    /* Zgloszenie nieprawidlaowosci */
    zgloszeniestanu(mqId,100,0);
    }
    ROZKAZ=0;
}
    }
```

Po otrzymaniu rozkazu wywoływana są funkcje pomocnicze **zablokowaniesilnikow**, **odblokowaniesilnikow** i **zgloszeniestanu**, które są opisane w następnych podrozdziałach.

4.4.2 200 – Odczytanie pozycji silnika

```
if(ROZKAZ==200)
```

```
{
sprintf(str,"(200 od %d) Otrzymanie rozkazu odczytania pozycji silnika.",KTO);
if(LOGI==1) logi(str);

/* Sprawdzenie lacznosci z sterownikiem */
LACZNOSC=pozycja(fd1,fd2);
if(LACZNOSC==1)
    {
    /* Zgloszenie gotowosci do pracy */
    zgloszeniestanu(mqId,200,1);
    }
else
    {
    /* Zgloszenie niegotowosci */
    zgloszeniestanu(mqId,200,0);
    }
ROZKAZ=0;
}
```

Po otrzymaniu rozkazu wywoływana są funkcje pomocnicze **pozycja** i **zgloszeniestanu**, które są opisane w następnych podrozdziałach.

4.4.3 300 – Poprawka do pozycji w czasie ruchu dobowego

Funkcja ta pozwala poprawić pozycje teleskopu o ilość sekud łuku zawartej w zmiennej **PARAMETR1** w czasie ruchu dobowego z nominalną prędkością. Po dokonaniu poprawki silnik porusza się ruchem dobowym.

```
if(ROZKAZ==300)
{
sscanf(buf,"%d %d %lf",&KTO,&ROZKAZ,&PARAMETR1);
PARAMETR1=INCRA*PARAMETR1/SEKLUKU;
sprintf(str,"(300 od %d) Otrzymanie poprawki od autoguidera (%10.10G s).",
        KTO,PARAMETR1*SEKLUKU/INCRA);
if(LOGI==1) logi(str);

/* Sprawdzenie lacznosci z sterownikiem */
LACZNOSC=poprawkapoz(fd1,fd2);
if(LACZNOSC==1)
    {
    /* Zgloszenie gotowosci do pracy */
    zgloszeniestanu(mqId,300,1);
    }
else
    {
    /* Zgloszenie niegotowosci */
    zgloszeniestanu(mqId,300,0);
    }
ROZKAZ=0;
}
```

Po otrzymaniu rozkazu wywoływana są funkcje pomocnicze **poprawkapoz** i **zgloszeniestanu**, które są opisane w następnych podrozdziałach.

4.4.4 400 – Właczenie i wyłączenie ruchu dobowego

Wraz z rozkazem przesyłana jest wartość **1** (włącz ruch dobowy) lub **0** (wyłącz ruch dobowy).

```
if(ROZKAZ==400)
    {
    sscanf(buf,"%d %d %lf",&KTO,&ROZKAZ,&PARAMETR1);

    if(PARAMETR1==0)
{
sprintf(str,"(400 od %d) Otrzymanie rozkazu zatrzymania ruchu dobowego.",KTO);
logi(str);
/* Zablokowanie silnikow */
LACZNOSC=ruchdobowy(fd1,fd2,0);
if(LACZNOSC==1)
    {
    /* Zgloszenie wykonania rozkazu */
    zgloszeniestanu(mqId,400,1);
    ROZKAZ=0;
    }
else
    {
    /* Zgloszenie nieprawidlaowosci */
    zgloszeniestanu(mqId,400,0);
    }
    ROZKAZ=0;
}
    if(PARAMETR1==1)
{
sprintf(str,"(400 od %d) Otrzymanie rozkazu uruchomienia ruchu dobowego.",KTO);
logi(str);
/* Odblokowanie silnikow */
LACZNOSC=ruchdobowy(fd1,fd2,1);
if(LACZNOSC==1)
    {
    /* Zgloszenie wykonania rozkazu */
    zgloszeniestanu(mqId,400,1);
    ROZKAZ=0;
    }
else
    {
    /* Zgloszenie nieprawidlaowosci */
    zgloszeniestanu(mqId,400,0);
    }
    ROZKAZ=0;
}
    }
```

Po otrzymaniu rozkazu wywoływana są funkcje pomocnicze **ruchdobowy** i **zgloszeniestanu**, które są opisane w następnych podrozdziałach.

4.4.5 430 – Włączenie ruchu dobowego z dwukrotną nominalną prędkością

Funkcja wykorzystywana przez autoguider żadający przyspieszenia ruchu dobowego.

```
if(ROZKAZ==430)
    {
    sscanf(buf,"%d %d %lf",&KTO,&ROZKAZ,&PARAMETR1);

sprintf(str,"(430 od %d) Otrzymanie rozkazu przyspieszenia ruchu dobowego.",KTO);
LACZNOSC=ruchdobowy(fd1,fd2,2);
if(LACZNOSC==1)
    {
    /* Zgloszenie wykonania rozkazu */
    zgloszeniestanu(mqId,430,2);
    ROZKAZ=0;
    }
else
    {
    /* Zgloszenie nieprawidlaowosci */
    zgloszeniestanu(mqId,430,0);
    }
    ROZKAZ=0;

    }
```

Po otrzymaniu rozkazu wywoływana są funkcje pomocnicze **ruchdobowy** i **zgloszeniestanu**, które są opisane w następnych podrozdziałach.

4.4.6 460 – Włączenie ruchu dobowego z połową nominalnej prędkości

Funkcja wykorzystywana przez autoguider żadający spowolnienia ruchu dobowego.

```
if(ROZKAZ==460)
    {
    sscanf(buf,"%d %d %lf",&KTO,&ROZKAZ,&PARAMETR1);

sprintf(str,"(460 od %d) Otrzymanie rozkazu zwolnienia ruchu dobowego.",KTO);
LACZNOSC=ruchdobowy(fd1,fd2,3);
if(LACZNOSC==1)
    {
    /* Zgloszenie wykonania rozkazu */
    zgloszeniestanu(mqId,460,3);
    ROZKAZ=0;
    }
else
    {
    /* Zgloszenie nieprawidlaowosci */
    zgloszeniestanu(mqId,460,0);
    }
    ROZKAZ=0;

    }
```

Po otrzymaniu rozkazu wywoływana są funkcje pomocnicze **ruchdobowy** i **zgloszeniestanu**, które są opisane w następnych podrozdziałach.

4.4.7 500 – Najazd na pozycje przy zatrzymanym silniku

Funkcja najeżdza na pozycje oddaloną o wartość parametru **PARAMETR1** wyrażoną w sekundach łuku. Funkcja ta w przeciwieństwie do funkcji **300** nie uwzględnia ruchu dobowego obiektu. Po jej wykonaniu silnik jest zatrzymany.

```
if(ROZKAZ==500)
{
sscanf(buf,"%d %d %lf",&KTO,&ROZKAZ,&PARAMETR1);
PARAMETR1=INCRA*PARAMETR1/SEKLUKU;
sprintf(str,"(500 od %d) Najazd na pozycje oddalona o (%10.10G s).",
     KTO,PARAMETR1*SEKLUKU/INCRA);
if(LOGI==1) logi(str);

/* Sprawdzenie lacznosci z sterownikiem */
LACZNOSC=najazdnapoz(fd1,fd2);
if(LACZNOSC==1)
    {
    /* Zgloszenie gotowosci do pracy */
    zgloszeniestanu(mqId,500,1);
    }
else
    {
    /* Zgloszenie niegotowosci */
    zgloszeniestanu(mqId,500,0);
    }
ROZKAZ=0;
}
```

Po otrzymaniu rozkazu wywoływana są funkcje pomocnicze **najazdnapoz** i **zgloszeniestanu**, które są opisane w następnych podrozdziałach.

4.4.8 600 – Wprowadzenie poprawek do pozycji silnika znajdującego się na bazie

Funkcja korzysta z pliku **<ścieżka do programu>/etc/poprawki.conf**.

```
if(ROZKAZ==600)
{
sscanf(buf,"%d %d %lf",&KTO,&ROZKAZ,&PARAMETR1);
sprintf(str,"(600 od %d) Rozkaz wprowadzenia poprawki do pozycji na bazie.",KTO);
logi(str);
```

Zapamiętanie licznika położenia z silnika

```
LACZNOSC=pozycja(fd1,fd2);
if(LACZNOSC==1)
    {
```

Odczytanie poprawek w pozycji z pliku **poprawki.conf** i obliczenie nowej wartości do zmiennej **POPRAWKAPOZ** .

```
logi("Wczytanie pliku z poprawkami poprawki.conf");
if((input=fopen("../etc/poprawki.conf","r"))==NULL)
    {
    logi("Nie moge otworzyc pliku poprawki.conf");
```

```
    }

fscanf(input,"%lf %lf %lf %s\n",&GODZ,&MIN,&SEK,&bzz);
fscanf(input,"%lf %lf %lf %s\n",&DGODZ,&DMIN,&DSEK,&bzz);
fclose(input);
POPRAWKAPOZ=-PARAMETR2+((GODZ+24)*60*60+MIN*60+SEK)*15+POPRAWKAPOZ;
logi("Zapisanie pozycji bazy w pliku bazaSRTA.tmp");
```

Zapamietanie pozycji polozenia silnika gdzie dokonano bazowania do pliku **<ścieżka do programu>/tmp/bazaSRTA.tmp**.

```
LACZNOSC=pozycja(fd1,fd2);
if(LACZNOSC==1)
{

    if((input=fopen("../tmp/bazaSRTA.tmp","w"))==NULL)
    {
    logi("Nie moge utworzyc pliku bazaSRTA.tmp");
    }

    fprintf(input,"%d %22.15G %22.15G %22.15G\n",ZNACZNIK,
                                            PARAMETR2,PARAMETR2*INCRA/SEKLUKU,POPRAWKAPOZ);
    fclose(input);
}
```

Ustawienie stanu **0** w pliku **<ścieżka do programu>/tmp/braklacznosci.tmp**. Plik ten jest wykorzystywany do ustalenia stanu sterownika w innych programach.

```
logi("Zapisanie danych do pliku braklacznosci.tmp");
if((input=fopen("../tmp/braklacznosci.tmp","w"))==NULL)
    {
    logi("Nie moge utworzyc pliku braklacznosci.tmp");
    }

fprintf(input,"0\n");
fclose(input);

/* Zgloszenie gotowosci do pracy */
zgloszeniestanu(mqId,600,1);

    }
else
    {
    /* Zgloszenie niegotowosci */
    zgloszeniestanu(mqId,600,0);
    }
ROZKAZ=0;
}
```

Użyte są funkcje pomocnicze **pozycja** i **zgloszeniestanu**, które są opisane w następnych podrozdziałach.

4.4.9 700 – Logowanie do sterownika silnika

Rozkaz wykorzystywany jest w przypadku, gdy wyłączony został sterownik silnika (np.awaria zasilania). Nie zalogowanie sie do sterownika silnika powoduje brak jego reakcji na wysyłane rozkazy.

```
if(ROZKAZ==700)
{
sprintf(str,"(700 od %d) Otrzymanie rozkazu zalogowania sie do silnika",KTO);
if(LOGI==1) logi(str);

/* Sprawdzenie lacznosci z sterownikiem */
LACZNOSC=logowaniedosilnikow(fd1,fd2);
if(LACZNOSC==1)
    {
    /* Zgloszenie gotowosci do pracy */
    zgloszeniestanu(mqId,700,1);
    }
else
    {
    /* Zgloszenie niegotowosci */
    zgloszeniestanu(mqId,700,0);
    }
ROZKAZ=0;
}
```

Po otrzymaniu rozkazu wywoływana są funkcje pomocnicze **logowaniedosilnikow** i **zgloszeniestanu**, które są opisane w następnych podrozdziałach.

4.4.10 800 – Sprawdzenie łączności ze sterownikiem silnika

```
if(ROZKAZ==800)
{
sprintf(str,"(800 od %d) Otrzymanie rozkazu sprawdzenia gotowosci pracy",KTO);
if(LOGI==1) logi(str);

/* Sprawdzenie lacznosci z sterownikiem */
LACZNOSC=kontrolalacznosci(fd1,fd2);
if(LACZNOSC==1)
    {
    /* Zgloszenie gotowosci do pracy */
    zgloszeniestanu(mqId,800,1);
    }
else
    {
    /* Zgloszenie niegotowosci */
    zgloszeniestanu(mqId,800,0);
    }
ROZKAZ=0;
}
```

Po otrzymaniu rozkazu wywoływana są funkcje pomocnicze **kontrolalacznosci** i **zgloszeniestanu**, które opisane zostaną w następnych podrozdziałach.

4.4.11 900 – Zakończenie pracy

Otrzymanie rozkazu **900** powoduje zakończenie pracy programu SRTA

```
if(ROZKAZ==900)
{
```

```
sprintf(str,"(900 od %d) Otrzymanie rozkazu zakonczenia pracy",KTO);
logi(str);
    /* Zakonczenie procesow i zamkniecie kolejki fifo */
kill(pid1,SIGKILL);
kill(pid,SIGKILL);
close(fd1);
close(fd2);
/* Przestawienie portu com1 na poprzednie parametry */
tcsetattr(fd,TCSANOW,&oldtio);
ROZKAZ=0;
zgloszeniestanu(mqId,900,1);
return 1;
}
```

Program wysyła potwierdzenie zakończenia pracy do programu od którego otrzymał rozkaz 900.

4.4.12 Zgłoszenie błędnego rozkazu

W przypadku gdy przybędzie rozkaz inny niż opisane powyżej wywołana zostanie funkcja pomocnicza **zgloszeniestanu** z parametrem **-1**.

```
if(ROZKAZ!=0)
    {
    zgloszeniestanu(mqId,-1,0);
    }
}
    }
    }
```

4.5 Funkcje pomocnicze

4.5.1 int fatal(czar ∗s)

Funkcja **fatal** powoduje wypisanie zwróconego przez jądro systemu błędu, kończąc wykonywanie programu **SRTA**.

```
int fatal(char *s)
{
perror(s);
exit(1);
}
```

4.5.2 int kontrolalacznosci(int fd1, int fd2)

Funkcje **kontrolalacznosci** wywołuje się z dwoma parametrami **fd1** i **fd2**, które są wskaźnikami do kolejek FIFO.

```
int kontrolalacznosci(int fd1,int fd2)
{
int i,res,ss,krok;
char buf[255],buf1[255];
```

Wyczyszczenie buforów

```
for(i=0;i<255;i=i+1)
{

buf[i]=0;
buf1[i]=0;
}
```

Przygotowanie rozkazu, który zostanie wysłany do sterownika silnika

```
buf[0]=ESC;
buf[1]='\x01';/* Numer silnika */
buf[2]='\x06';/* Numer command =6*/
buf[3]='\x1C';/* Suma kontrolna */
```

Wysłanie 4 bajtów rozkazu do silnika

```
write(fd2,buf,4); /* wysylam 4 bajty */
```

Ustawienie alarmu

```
alarm_flag=FALSE;
alarm(ALARMTIME);
```

Ilość odczytywanych bajtów, które zwraca sterownik po otrzymaniu rozkazu

```
krok=14;
```

Odczytywanie z kolejki FIFO bajt po bajcie

```
for(i=0;i<krok;i=i+1)
{
res=read(fd1,&buf1,1); /* czytam pojedynczy */
}
```

Wyłączenie alarmu

```
alarm(0);
```

Sprawdzenie czy zgadza się suma kontrolna rozkazu i czy nie nastąpiło przerwanie sygnałem **SIGALARM**

```
if(buf1[0]=='\x1a' && alarm_flag == FALSE)
```

Jeśli suma kontrolna zgadza się i brak sygnału **SIGALARM** to

```
{
ss=1;
}
```

jeśli suma się niezgadza lub przybył sygnał **SIGALARM** to

```
else
{
ss=0;
logi("Zapisanie danych do pliku braklacznosci.tmp");
if((input=fopen("../tmp/braklacznosci.tmp","w"))==NULL)
{
logi("Nie moge utworzyc pliku braklacznosci.tmp");
}

fprintf(input,"1\n");
fclose(input);

alarm_flag=FALSE;
}
```

Zwrot wartości **ss** po zakończeniu podprogramu.

```
return ss;
}
```

4.5.3 int pozycja(int fd1, int fd2)

Funkcje **pozycja** wywołuje się z dwoma parametrami **fd1** i **fd2**, które są wskaźnikami do kolejek FIFO.

```
int pozycja(int fd1,int fd2)
{
int i,res,ss,krok;
long int dane[255],liczba;
char buf[255],buf1[255];
```

Zerowanie buforów

```
for(i=0;i<255;i=i+1)
{
buf[i]=0;
buf1[i]=0;

}
```

Wprowadzenie do bufora **buf** rozkazu, który zostanie wysłany do sterownika silnika

```
buf[0]=ESC;
buf[1]='\x01';/* Numer silnika */
buf[2]='\x07';/* Numer komendy =7 */
buf[3]='\x1D';/* Suma kontrolna */
```

Wysłanie 4 bajtów rozkaz do sterownika silnika

```
write(fd2,buf,4);/* Wysylam 4 bajty */
```

Ustawienie alarmu

```
alarm_flag=FALSE;
alarm(ALARMTIME);
```

Ilość odczytywanych bajtów, które zwraca sterownik po otrzymaniu rozkazu

```
krok=22;
```

Czytamy je bajt po bajcie, wprowadzamy do tablicy **dane** i obliczamy sumę kontrolną

```
for(i=0;i<krok;i=i+1)
{
res=read(fd1,&buf1,1);/* Czytam po jednym bajcie */
liczba=buf1[0];
if(liczba<0) liczba=liczba+256;
dane[i+1]=liczba;
/* Obliczam sume kontrolna */
if(i==0) BCC=buf1[0];
if(i!=krok-1 && i>0) BCC=BCC^buf1[0];
}
```

Sprawdzamy czy zgadza się nasza obliczona suma kontrolna z sumą przesłaną przez sterownik silnika oraz czy przybył sygnał **SIGALARM**

```
if( buf1[0]==BCC && alarm_flag == FALSE)
{
```

Jeśli sumy kontrolne się zgadzają i nie było sygnału **SIGALARM** to obliczamy pozycje w inkramentach

```
/* Wylaczenie alarmu */
alarm(0);
/* obliczam ilosc incrementow */
PARAMETR2=(dane[14]+dane[15]*256+dane[16]*256*256+dane[17]*256*256*256);
```

Następnie przeliczamy pozycje na sekundy łuku z uwzględnieniem poprawki

```
PAR2=PARAMETR2;
/* Przeliczenie na sekundy luku */
PARAMETR2=PARAMETR2/INCRA*SEKLUKU+POPRAWKAPOZ;
```

Ustawiam zmienną **ss** na wartość **1** co oznacza, że łączność była prawidłowa

```
ss=1;
```

Dodaje **BIAS** do pozycji silnika

```
PARAMETR2=PARAMETR2+BIAS;
```

i zapisuję nowe wartości pozycji do pliku **<ścieżka do programu>/tmp/pozycjaSRTA.tmp**

```
if((input=fopen("../tmp/pozycjaSRTA.tmp","w"))==NULL)
{
logi("Nie moge utworzyc pliku pozycjaSRTA.tmp");
}

fprintf(input,"%22.15G %22.15G %22.15G\n",PARAMETR2,PAR2,PARAMETR2*INCRA/SEKLUKU);
fclose(input);
}
```

Jeżeli sumy kontrolne niezgadzały się lub przybył sygnał **SIGALARM** to

```
else
{
ss=0;
logi("Zapisanie danych do pliku braklacznosci.tmp");
if((input=fopen("../tmp/braklacznosci.tmp","w"))==NULL)
{
logi("Nie moge utworzyc pliku braklacznosci.tmp");
}

fprintf(input,"1\n");
fclose(input);

alarm_flag=FALSE;

}
```

Funkcja zwraca warość **ss** po zakończeniu.

```
return ss;
}
```

4.5.4 void logi(char *komunikat)

Funkcja logi wywołana jest z jednym parametrem **komunikat**, który jest łańcuchem znaków.

```
void logi(char *komunikat)
{
time_t t;
struct tm *tp;
```

Odczytanie bieżącego czasu

```
time(&t);
```

Przejscie do struktury **tm** czasu

```
tp=localtime(&t);
```

i zapisanie do pliku **< ścieżka do programu>/log/SRTA.log**

```
output=fopen("../log/SRTA.log","a");
fprintf(output,"%d/%d/%d %d:%d:%d [SRTA]: %s\n",(tp->tm_year)+1900,
(tp->tm_mon)+1,(tp->tm_mday),tp->tm_hour,tp->tm_min,tp->tm_sec,komunikat);
fclose(output);
}
```

4.5.5 int zablokowaniesilnikow(int fd1, int fd2)

Funkcje **zablokowaniesilnikow** wywołuje się z dwoma parametrami **fd1** i **fd2**, które są wskaźnikami do kolejek FIFO.

```
int zablokowaniesilnikow(int fd1,int fd2)
{
int i,res,ss;
char buf[255],buf1[255];
```

Zerowanie buforów

```
for(i=0;i<255;i=i+1)
{
buf1[i]=0;
buf[i]=0;
}
```

Wprowadzenie do bufora **buf** rozkazu, który zostanie wysłany do sterownika silnika

```
buf[0]=ESC;
buf[1]='\x01';
buf[2]='\x00';
buf[3]='\x1A';
```

Wysłanie 4 bajtów rozkaz do sterownika silnika

```
write(fd2,buf,4);
```

Ustawienie alarmu

```
alarm_flag=FALSE;
alarm(ALARMTIME);
```

Odczytanie 1 bajtu potwierdzającego wykonanie rozkazu

```
res=read(fd1,&buf1,1);
```

Wyłączenie alarmu

```
alarm(0);
```

Sprawdzamy, czy sterownik przesłał bajt o wartości **ACK** i czy przybył sygnał **SIGALARM**

```
if( buf1[0]==ACK && alarm_flag == FALSE)
```

Jeśli tym bajtem jest **ACK** i nie było sygnału **SIGALARM** to

```
{
ss=1;
}
```

jeśli był inny bajt lub przybył sygnał **SIGALARM** to

```
else
{
ss=0;
logi("Zapisanie danych do pliku braklacznosci.tmp");
if((input=fopen("../tmp/braklacznosci.tmp","w"))==NULL)
{
logi("Nie moge utworzyc pliku braklacznosci.tmp");
}

fprintf(input,"1\n");
fclose(input);

alarm_flag=FALSE;
}
```

Funkcja zwraca wartość **ss** po zakończeniu.

```
return ss;
}
```

4.5.6 int logowaniedosilnikow(int fd1, int fd2)

Funkcje **logowaniedosilnikow** wywołuje się z dwoma parametrami **fd1** i **fd2**, które są wskaźnikami do kolejek FIFO.

```
int logowaniedosilnikow(int fd1,int fd2)
{
int i,res,ss;
char buf[255],buf1[255];
```

Zerowanie buforów

```
for(i=0;i<255;i=i+1)
{
buf1[i]=0;
buf[i]=0;
}
```

Wprowadzenie do bufora **buf** rozkazu, który zostanie wysłany do sterownika silnika

```
buf[0]=ESC;
buf[1]='\x01';
buf[2]='\x03';
buf[3]='\x19';
```

Wysłanie 4 bajtów rozkaz do sterownika silnika

```
write(fd2,buf,4);
```

Ustawienie alarmu

```
alarm_flag=FALSE;
alarm(ALARMTIME);
```

Odczytanie 1 bajtu potwierdzającego wykonanie rozkazu

```
res=read(fd1,&buf1,1);
```

Wyłączenie alarmu

```
alarm(0);
```

Sprawdzamy, czy sterownik przesłał bajt o wartości **ACK** i czy przybył sygnał **SIGALARM**

```
if( buf1[0]==ACK && alarm_flag == FALSE)
```

Jeśli tym bajtem jest **ACK** i nie było sygnału **SIGALARM** to

```
{
ss=1;
}
```

jeśli był inny bajt lub przybył sygnał **SIGALARM** to

```
else
{
ss=0;
logi("Zapisanie danych do pliku braklacznosci.tmp");
if((input=fopen("../tmp/braklacznosci.tmp","w"))==NULL)
{
logi("Nie moge utworzyc pliku braklacznosci.tmp");
}

fprintf(input,"1\n");
fclose(input);

alarm_flag=FALSE;
}
```

Zwrot wartości **ss** po zakończeniu.

```
return ss;
}
```

4.5.7 int poprawkapoz(int fd1, int fd2)

Funkcje **poprawkapoz** wywołuje się z dwoma parametrami **fd1** i **fd2**, które są wskaźnikami do kolejek FIFO. Program najpierw wysyła do sterownika silników nową warości inkramentów a następnie uruchamia program znajdujący się w sterowniku silnika.

```
int poprawkapoz(int fd1,int fd2)
{
int i,res,ss,pomoc,licznik;
double a,v,vv,t,t1,t2,s2,ssd,dd,vvd,odd;
char buf[255],buf1[255];
```

Wyzerwoanie buforów

```
for(i=0;i<255;i=i+1)
{
buf1[i]=0;
buf[i]=0;
}
```

Obliczenie poprawki, która uwzględnia ruch dobowy z nominalną prędkością. Wyrażamy przyspieszenie i prędkość maksymalną w sekundach łuku na sekundy czasu. Wartości **ACCELERACJA** i **VMAX** muszą być zgodne z wartościami zadanymi w programie znajdującym się w sterowniku silnika.

```
a=ACCELERACJA*SEKLUKU/60.; /* sekundy luku/sek*/
v=VMAX*SEKLUKU/60.; /* sekundy luku/sek*/
```

Otrzymany **PARAMETR1** z rozkazu **300** przeliczamy na sekundy łuku

```
PARAMETR1=PARAMETR1*SEKLUKU/INCRA;
```

Jeżeli **PARAMETR1** jest dodatni to

```
if(PARAMETR1>=0.)
    {
```

Obliczamy prędkość jaką by uzyskał silnik gdyby cały czas przyspieszał

```
    vv=sqrt(2.*PARAMETR1*a);
    dd=PARAMETR1;
    odd=10e30;
    vvd=1.*SEKLUKU;
```

Jeżeli obliczona prędkość **vv** jest mniejsza od **v** (predkości maksymalnej **VMAX**) to obliczamy metodą iteracji odległość o jaką ma się przestawić silnik (uwzględniając ruch dobowy).

```
    if(vv<=v)
{
while(sqrt((dd-odd)*(dd-odd))>1.)
    {
    t=sqrt(2*dd/a);
    ssd=vvd*t;
    odd=dd;
    dd=PARAMETR1+ssd;
    }

}
```

Jeżeli obliczona prędkość **vv** jest większa od **v** (predkości maksymalnej **VMAX**) to obliczamy metodą iteracji odległość o jaką ma się przestawić silnik (uwzględniając ruch dobowy) z uwzględniemiem maksymalnej prędkości **VMAX**.

```
    else
{
while(sqrt((dd-odd)*(dd-odd))>1.)
    {

    t1=v/a;
    s2=a*t1*t1/2;
    t2=(dd-2*s2)/v;
    t=2*t1+t2;
    ssd=vvd*t;
    odd=dd;
    dd=PARAMETR1+ssd;
    }
}
    }
```

Jeżeli natomiast parametr **PARAMETR1** jest mniejszy od zera to zmieniamy jego znak

```
else
    {
    PARAMETR1=-PARAMETR1;
```

Obliczamy prędkość jaką by uzyskał silnik gdyby cały czas przyspieszał

```
    vv=sqrt(2*PARAMETR1*a);
    dd=PARAMETR1;
    odd=10e30;
    vvd=1*SEKLUKU;
```

Jeżeli obliczona prędkość **vv** jest mniejsza od **v** (predkości maksymalnej **VMAX**) to obliczamy metodą iteracji odległość o jaką ma się przestawić silnik (uwzględniając ruch dobowy).

```
    if(vv<=v)
{
while(sqrt((dd-odd)*(dd-odd))>1.)
    {
    t=sqrt(2*dd/a);
    ssd=vvd*t;
    odd=dd;
    dd=PARAMETR1-ssd;
    }


}
```

Jeżeli obliczona prędkość **vv** jest większa od **v** (predkości maksymalnej **VMAX**) to obliczamy metodą iteracji odległość o jaką ma się przestawić silnik (uwzględniając ruch dobowy) z uwzględniemiem maksymalnej prędkości **VMAX**.

```
    else
{
while(sqrt((dd-odd)*(dd-odd))>1.)
    {
    t1=v/a;
```

```
    s2=a*t1*t1/2;
    t2=(dd-2*s2)/v;
    t=2*t1+t2;
    ssd=vvd*t;
    odd=dd;
    dd=PARAMETR1-ssd;
    }

}
```

Przywracamy właściwy znak parametrowi **PARAMETR1**

```
    PARAMETR1=-PARAMETR1;
    }
```

Przeliczamy wartość parametru na inkramenty

```
PARAMETR1=PARAMETR1/SEKLUKU*INCRA;
```

Wpisujemy rozkaz w bufor **buf**, który zostanie wysłany do sterownika silników

```
buf[0]=ESC;
buf[1]='\x01'; /* nr silnika*/
buf[2]='\x4c'; /* command 76b */
buf[3]='\x01';  /* write-option=1 */
```

Wprowadzenie numeru linii zawartej w zmiennej **LINIAPOZ** do dwóch bajtów

```
    licznik=0;
    while((LINIAPOZ-licznik*256)>255) licznik=licznik+1;
    buf[4]= (LINIAPOZ-licznik*256);  /* linia BIAS mlodszy */
    buf[5]= (licznik);  /* linia BIAS starszy */
```

Dalsza część rozkazu

```
buf[6]='\x20';  /* BIAS command */
```

Wpisanie ilości inkamentów zawartych w zmiennej **PARAMETR1** do czterech bajtów, gdy **PARAMETR1** jest dodatni

```
    if(PARAMETR1>=0)
    {
    licznik=0;
    while((PARAMETR1-licznik*256*256*256)>256*256*255) licznik=licznik+1;
    pomoc= (PARAMETR1-licznik*256*256*256);  /* linia BIAS najstarsz */
    buf[10]=licznik;
    licznik=0;
    while((pomoc-licznik*256*256)>256*255) licznik=licznik+1;
    pomoc= (pomoc-licznik*256*256);  /* linia BIAS starszy */
    buf[9]=licznik;
    licznik=0;
    while((pomoc-licznik*256)>255) licznik=licznik+1;
    pomoc= (pomoc-licznik*256);  /* linia BIAS mlodszy */
    buf[8]=licznik;
    buf[7]=pomoc;  /* linia BIAS najmlodszy */
   /* printf("%d\n",pomoc);*/
    }
```

lub gdy jest ujemny

```
    else
    {
    PARAMETR1=-PARAMETR1;
    licznik=0;
    while((PARAMETR1-licznik*256*256*256)>256*256*255) licznik=licznik+1;
    pomoc= (PARAMETR1-licznik*256*256*256);  /* linia BIAS najstarsz */
    buf[10]=255-licznik;
    licznik=0;
    while((pomoc-licznik*256*256)>256*255) licznik=licznik+1;
    pomoc= (pomoc-licznik*256*256);  /* linia BIAS starszy */
    buf[9]=255-licznik;
    licznik=0;
    while((pomoc-licznik*256)>255) licznik=licznik+1;
    pomoc= (pomoc-licznik*256);  /* linia BIAS mlodszy */
    buf[8]=255-licznik;
    buf[7]=255-pomoc;  /* linia BIAS najmlodszy */
    }
```

Uzupełnienie zerami trzech następnych bajtów rozkazu

```
    buf[11]='\x0';
    buf[12]='\x0';
    buf[13]='\x0';
```

Obliczenie sumy kontrolnej i umieszczenie jej na końcu rozkazu

```
BCC=ESC;
for (i=1;i<14;i=i+1)
    BCC=BCC^buf[i];
buf[14]=BCC;
```

Wysłanie 15 bajtów do sterownika silnika

```
write(fd2,buf,15);
```

Ustawienie alarmu

```
alarm_flag=FALSE;
alarm(ALARMTIME);
```

Odczytanie 1 bajtu potwierdzenia z sterownika silnika

```
res=read(fd1,&buf1,1);
```

Wyłączenie alarmu

```
alarm(0);
```

Sprawdzenie czy odczytany bajt to **ACK** i czy nie było sygnału **SIGALARM**

```
if(buf1[0]==ACK && alarm_flag == FALSE)
    {
```

Jeśli był to znak **ACK** i nie było sygnału **SIGALARM** to możemy uruchomić program w sterowniku silnika. Zerujemy bufory

```
for(i=0;i<255;i=i+1)
{
buf1[i]=0;
buf[i]=0;
}
```

Wprowadzamy pierwsze bajty rozkazu do bufora **buf**

```
buf[0]=ESC;
buf[1]='\x01'; /*numer osi */
buf[2]='\x0D';  /* command  */
```

Wprowadzamy numer linii zawartej w zmiennej **LINIAPA** do dwóch kolejnych bajtów rozkazu

```
    licznik=0;
    while((LINIAPA-licznik*256)>255) licznik=licznik+1;
    buf[3]= (LINIAPA-licznik*256);  /* linia BIAS mlodszy */
    buf[4]= (licznik);  /* linia BIAS starszy */
```

Obliczamy sumę kontrolną i wprowadzamy do ostatniego bajtu rozkazu

```
BCC=ESC;
for (i=1;i<5;i=i+1)
    BCC=BCC^buf[i];
buf[5]=BCC;
```

Wysyłamy 6 bajtów rozkazu do sterownika silnika

```
write(fd2,buf,6);
```

Ustawiamy alrarm

```
alarm_flag=FALSE;
alarm(ALARMTIME);
```

Odczytanie 1 bajtu potwierdzenia z sterownika silnika

```
res=read(fd1,&buf1,1);
```

Wyłączenie alarmu

```
alarm(0);
```

Sprawdzenie czy odczytany bajt to **ACK** i czy nie było sygnału **SIGALARM**

```
if( buf1[0]==ACK && alarm_flag == FALSE)
```

Jeśli jest to **ACK** i nie było sygnału **SIGALARM** to

```
    {
    ss=1;

    }
```

Jeśli był bład (brak **ACK** lub był sygnał **SIGALARM**) przy uruchomieniu programu to wykonujemy

```
else
    {
    ss=0;
     logi("Zapisanie danych do pliku braklacznosci.tmp");
if((input=fopen("../tmp/braklacznosci.tmp","w"))==NULL)
{
logi("Nie moge utworzyc pliku braklacznosci.tmp");
}

fprintf(input,"1\n");
fclose(input);

    alarm_flag=FALSE;

    }
    }
```

Jeśli był błąd (brak **ACK** lub był sygnał **SIGALARM**) przy przesyłaniu pozycji to wykonujemy

```
else
    {
    ss=0;
     logi("Zapisanie danych do pliku braklacznosci.tmp");
if((input=fopen("../tmp/braklacznosci.tmp","w"))==NULL)
{
logi("Nie moge utworzyc pliku braklacznosci.tmp");
}

fprintf(input,"1\n");
fclose(input);

    alarm_flag=FALSE;

    }
```

Funkcja zwraca wartość **ss** po zakończeniu.

```
return ss;
}
```

4.5.8 int najazdnapoz(int fd1, int fd2)

Funkcje **najazdnapoz** wywołuje się z dwoma parametrami **fd1** i **fd2**, które są wskaźnikami do kolejek FIFO. Program najpierw wysyła do sterownika silnika nową warości inkramentów a następnie uruchamia program znajdujący się w sterowniku silnika.

```
int najazdnapoz(int fd1,int fd2)
{
int i,res,ss,pomoc,licznik;
char buf[255],buf1[255];
```

Wyzerowanie buforów

```
for(i=0;i<255;i=i+1)
{
buf1[i]=0;
buf[i]=0;
}
```

Wprowadzenie pierwszych bajtów rozkazu który zostanie wysłany do sterownika silnika

```
buf[0]=ESC;
buf[1]='\x01'; /* nr silnika*/
buf[2]='\x4c'; /* command 76b */
buf[3]='\x01';  /* write-option=1 */
```

Wprowadzenie numeru linii zawartej w zmiennej **LINIANNPOZ** do dwóch kolejnych bajtów rozkazu

```
    licznik=0;
    while((LINIANNPOZ-licznik*256)>255) licznik=licznik+1;
    buf[4]= (LINIANNPOZ-licznik*256);  /* linia BIAS mlodszy */
    buf[5]= (licznik);  /* linia BIAS starszy */
```

Następny bajt rozkazu

```
buf[6]='\x20';  /* BIAS command */
```

Wprowadzenie wartości inkramentów zawartej w zmiennej **PARAMETR1** do czterech bajtów rozkazu, gdy **PARAMETR1** jest dodatni

```
    if(PARAMETR1>=0)
    {
    licznik=0;
    while((PARAMETR1-licznik*256*256*256)>256*256*255) licznik=licznik+1;
    pomoc= (PARAMETR1-licznik*256*256*256);  /* linia BIAS najstarsz */
    buf[10]=licznik;
    licznik=0;
    while((pomoc-licznik*256*256)>256*255) licznik=licznik+1;
    pomoc= (pomoc-licznik*256*256);  /* linia BIAS starszy */
    buf[9]=licznik;
    licznik=0;
    while((pomoc-licznik*256)>255) licznik=licznik+1;
    pomoc= (pomoc-licznik*256);  /* linia BIAS mlodszy */
    buf[8]=licznik;
    buf[7]=pomoc;  /* linia BIAS najmlodszy */
    }
```

lub gdy jest ujemny

```
    else
    {
    PARAMETR1=-PARAMETR1;
    licznik=0;
    while((PARAMETR1-licznik*256*256*256)>256*256*255) licznik=licznik+1;
    pomoc= (PARAMETR1-licznik*256*256*256);  /* linia BIAS najstarsz */
    buf[10]=255-licznik;
    licznik=0;
    while((pomoc-licznik*256*256)>256*255) licznik=licznik+1;
    pomoc= (pomoc-licznik*256*256);  /* linia BIAS starszy */
    buf[9]=255-licznik;
    licznik=0;
    while((pomoc-licznik*256)>255) licznik=licznik+1;
    pomoc= (pomoc-licznik*256);  /* linia BIAS mlodszy */
    buf[8]=255-licznik;
    buf[7]=255-pomoc;  /* linia BIAS najmlodszy */
    }
```

Uzupełnienie pozostałych bajtów zerami

```
    buf[11]='\x0';
    buf[12]='\x0';
    buf[13]='\x0';
```

Obliczenie sumy kontrolnej i wprowadzenie jej do ostatniego bajtu rozkazu

```
BCC=ESC;
for (i=1;i<14;i=i+1)
    BCC=BCC^buf[i];
buf[14]=BCC;
```

Wyslanie 15 bajtów rozkazu do sterownika silnika

```
write(fd2,buf,15);
```

Ustawienie alarmu

```
alarm_flag=FALSE;
alarm(ALARMTIME);
```

Odczytanie 1 bajtu potwierdzenia z sterownika silnika

```
res=read(fd1,&buf1,1);
```

Wyłączenie alarmu

```
alarm(0);
```

Sprawdzenie czy tym znakiem jest **ACK** i czy był sygnał **SIGALARM**

```
if(buf1[0]==ACK && alarm_flag == FALSE)
    {
```

Jeśli to był znak **ACK** i nie było sygnału **SIGALARM** to uruchamiamy program znajdujący się w sterowniku silnika. Zerujemy bufory

```
for(i=0;i<255;i=i+1)
{
buf1[i]=0;
buf[i]=0;
}
```

Wprowadzamy pierwsze bajty rozkazu do bufora **buf**

```
buf[0]=ESC;
buf[1]='\x01'; /*numer osi */
buf[2]='\x0D';  /* command  */
```

Wprowadzamy numer linii zawarty w zmiennej **LINIANNP** do dwóch kolejnych bajtów rozkazu

```
    licznik=0;
    while((LINIANNP-licznik*256)>255) licznik=licznik+1;
    buf[3]= (LINIANNP-licznik*256);  /* linia BIAS mlodszy */
    buf[4]= (licznik);  /* linia BIAS starszy */
```

Obliczamy sumę kontrolną i wprowadzamy do ostatniego bajtu rozkazu

```
BCC=ESC;
for (i=1;i<5;i=i+1)
    BCC=BCC^buf[i];
buf[5]=BCC;
```

Wysyłamy 6 bajtów do sterownika silnika

```
write(fd2,buf,6);
```

Ustawiam alarm

```
alarm_flag=FALSE;
alarm(ALARMTIME);
```

Odczytanie 1 bajtu potwierdzenia z sterownika silnika

```
res=read(fd1,&buf1,1);
```

Wyłączamy alarm

```
alarm(0);
```

Sprawdzenie czy tym znakiem jest **ACK** i czy był sygnał **SIGALARM**

```
if(buf1[0]==ACK && alarm_flag == FALSE)
    {
```

Jeśli to był znak **ACK** i nie było sygnału **SIGALARM** to

```
    ss=1;
    }
```

Jeśli był błąd (brak **ACK** lub był sygnał **SIGALARM**) przy uruchamianiu programu znajdującego się w sterowniku silnika to

```
else
    {
    ss=0;
     logi("Zapisanie danych do pliku braklacznosci.tmp");
if((input=fopen("../tmp/braklacznosci.tmp","w"))==NULL)
{
logi("Nie moge utworzyc pliku braklacznosci.tmp");
}

fprintf(input,"1\n");
fclose(input);

    alarm_flag=FALSE;
    }
    }
```

Jeśli był błąd (brak **ACK** lub był sygnał **SIGALARM**) przy przesyłaniu pozycji to wykonujemy

```
else
    {
    ss=0;
     logi("Zapisanie danych do pliku braklacznosci.tmp");
if((input=fopen("../tmp/braklacznosci.tmp","w"))==NULL)
{
logi("Nie moge utworzyc pliku braklacznosci.tmp");
}

fprintf(input,"1\n");
fclose(input);

    alarm_flag=FALSE;
    }
```

Funkcja zwraca wartość **ss** po zakończeniu.

```
return ss;
}
```

4.5.9 int odblokowaniesilnikow(int fd1, int fd2)

Funkcje **odblokowaniesilnikow** wywołuje się z dwoma parametrami **fd1** i **fd2**, które są wskaźnikami do kolejek FIFO.

```
int odblokowaniesilnikow(int fd1,int fd2)
{
int i,res,ss;
char buf[255],buf1[255];
```

Wyzerowanie buforów

```
for(i=0;i<255;i=i+1)
{
buf1[i]=0;
buf[i]=0;
}
```

Wprowadzenie rozkazu do bufora **buf**

```
buf[0]=ESC;
buf[1]='\x01';
buf[2]='\x01';
buf[3]='\x1B';
```

Wysłanie 4 bajtów do sterownika silnika

```
write(fd2,buf,4);
```

Ustawienie alarmu

```
alarm_flag=FALSE;
alarm(ALARMTIME);
```

Odczytanie 1 bajtu potwierdzenia z sterownika silnika

```
res=read(fd1,&buf1,1);
```

Wyłaczenie alarmu

```
alarm(0);
```

Sprawdzenie czy tym znakiem jest **ACK** i czy był sygnał **SIGALARM**

```
if(buf1[0]==ACK && alarm_flag == FALSE)
{
```

Jeśli to był znak **ACK** i nie było sygnału **SIGALARM** to

```
ss=1;
}
```

Jeśli był błąd (brak **ACK** lub był sygnał **SIGALARM**) to

```
else
{
ss=0;
logi("Zapisanie danych do pliku braklacznosci.tmp");
if((input=fopen("../tmp/braklacznosci.tmp","w"))==NULL)
{
logi("Nie moge utworzyc pliku braklacznosci.tmp");
}

fprintf(input,"1\n");
fclose(input);

alarm_flag=FALSE;
}
```

Funkcja zwraca wartość **ss** po zakończeniu.

```
return ss;
}
```

4.5.10 int ruchdobowy(int fd1, int fd2, int par)

Funkcje **ruchdobowy** wywołuje się z trzema parametrami **fd1**, **fd2** i **par**. Pierwsze dwa są wskaźnikami do kolejek FIFO, trzeci natomiast określa z jaką prędkością będzie poruszał się silnik :

0 – zatrzymanie silnika,

1 – uruchomienie ruchu dobowego z nominalną prędkością,

2 – uruchomienie ruchu dobowego z podwójną prędokściȩą,

3 – uruchomienie ruchu dobowego z połową nominalnej prędkości.

```
int ruchdobowy(int fd1,int fd2,int par)
{
int i,res,ss,licznik;
char buf[255],buf1[255];
```

Zerowanie buforów

```
for(i=0;i<255;i=i+1)
{
buf1[i]=0;
buf[i]=0;
}
```

Wprowadzenie pierwszych bajtów rozkazu do bufora **buf**

```
buf[0]=ESC;
buf[1]='\x01'; /*numer osi */
buf[2]='\x0D';  /* command  */
```

Gdy **par** ma wartość **0** to wpisujemu numer linii zawartej w parametrze **LINIASTOP** do dwóch kolejnych bajtów rozkazu

```
if(par==0)
    {
    licznik=0;
    while((LINIASTOP-licznik*256)>255) licznik=licznik+1;
    buf[3]= (LINIASTOP-licznik*256);  /* linia BIAS mlodszy */
    buf[4]= (licznik);  /* linia BIAS starszy */
    }
```

Gdy **par** ma wartość **1** to wpisujemu numer linii zawartej w parametrze **LINIARD** do dwóch kolejnych bajtów rozkazu

```
if(par==1)
    {
    licznik=0;
    while((LINIARD-licznik*256)>255) licznik=licznik+1;
    buf[3]= (LINIARD-licznik*256);  /* linia BIAS mlodszy */
    buf[4]= (licznik);  /* linia BIAS starszy */
    }
```

Gdy **par** ma wartość **2** to wpisujemu numer linii zawartej w parametrze **LINIARDPLUS** do dwóch kolejnych bajtów rozkazu

```
if(par==2)
    {
    licznik=0;
    while((LINIARDPLUS-licznik*256)>255) licznik=licznik+1;
    buf[3]= (LINIARDPLUS-licznik*256);  /* linia BIAS mlodszy */
    buf[4]= (licznik);  /* linia BIAS starszy */
    }
```

Gdy **par** ma wartość **3** to wpisujemu numer linii zawartej w parametrze **LINIARDMINUS** do dwóch kolejnych bajtów rozkazu

```
if(par==3)
    {
    licznik=0;
    while((LINIARDMINUS-licznik*256)>255) licznik=licznik+1;
    buf[3]= (LINIARDMINUS-licznik*256);  /* linia BIAS mlodszy */
    buf[4]= (licznik);  /* linia BIAS starszy */
    }
```

Obliczamy sume kontrolną i umieszczamy ją w ostatnim bajcie rozkazu

```
BCC=ESC;
for (i=1;i<5;i=i+1)
    BCC=BCC^buf[i];
buf[5]=BCC;
```

Wysyłamy 6 bajtów rozkazu do sterownika silnika

```
write(fd2,buf,6);
```

Ustawiamy alarm

```
alarm_flag=FALSE;
alarm(ALARMTIME);
```

Odczytanie 1 bajtu potwierdzenia z sterownika silnika

```
res=read(fd1,&buf1,1);
```

Wyłączenie alarmu

```
alarm(0);
```

Sprawdzenie czy tym znakiem jest **ACK** i czy był sygnał **SIGALARM**

```
if(buf1[0]==ACK && alarm_flag == FALSE)
{
```

Jeśli to był znak **ACK** i nie było sygnału **SIGALARM** to

```
ss=1;
}
```

Jeśli był błąd (brak **ACK** lub był sygnał **SIGALARM**) to

```
else
{
ss=0;
logi("Zapisanie danych do pliku braklacznosci.tmp");
if((input=fopen("../tmp/braklacznosci.tmp","w"))==NULL)
{
logi("Nie moge utworzyc pliku braklacznosci.tmp");
}

fprintf(input,"1\n");
fclose(input);

alarm_flag=FALSE;
}
```

Funkcja zwraca wartość zmiennej **ss** po zakończeniu.

```
return ss;
}
```

4.5.11 void setflag(int sig)

Funkcja **setflag** jest wykonywana gdy do programu zostanie wysłany sygnał **SIGALARM**. Funkcja ustawia wartość zmiennej **alarm_flag** na wartość **TRUE** i ponownie inicjuje alarm.

```
void setflag(int sig)
{
    alarm_flag = TRUE;
    alarm(ALARMTIME);
}
```

4.5.12 void zgloszeniestanu(int mqId, int ROZKAZ, int STAN)

Funkcja **zgloszeniestanu** wywoływana jest z trzema parametrami **mqId**, **ROZKAZ** i **STAN**. Pierwszy parametr jest wskaźnikiem do kolejki komunikatów. Drugi zawiera rozkaz jaki otrzymał program **SRTA** a odpowiedź jest formatowana w zależności od konkretnego komunikatu. Trzeci parametr przyjmuje dwa stany **0** lub **1** w zależności czy odpowiedz ma zgłosić wykonanie rozkazu czy błąd łączności. Funkcja również obsługuje logi programu.

```
void zgloszeniestanu(int mqId, int ROZKAZ, int STAN)
{
  char buf[255],str[255];
```

Część programu odpowiedzialna za obsługę logów programowych

```
if(ROZKAZ==999 && STAN==1) logi("Zgloszenie gotowosci do procesu nadzorujacego ");
if(ROZKAZ==999 && STAN==0) logi("!!! Zgloszenie awarii do procesu nadzorujacego !!!");
if(ROZKAZ==900 && STAN==1) logi("(900) Zakonczenie programu na zadanie");
if(ROZKAZ==900 && STAN==1) logi("################################################
                                ######################");
if(ROZKAZ==800 && STAN==1) if(LOGI==1) logi("(800) Zgloszenie gotowosci na zadanie
                                pytajacego ");
if(ROZKAZ==800 && STAN==0) logi("(800) !!! Zglosenie braku gotowosci na zadanie
```

```
                                pytajacego !!!");
if(ROZKAZ==700 && STAN==1) logi("(700) Zalogowanie sie do BIOSA silnika ");
if(ROZKAZ==700 && STAN==0) logi("(700) !!! Zglosenie nie wykonania rozkazu
                                logowania do BIOSU !!!");
if(ROZKAZ==100 && STAN==1 && PARAMETR1==1) logi("(100) Silniki zablokowane ");
if(ROZKAZ==100 && STAN==1 && PARAMETR1==0) logi("(100) Silniki odblokowane ");
if(ROZKAZ==100 && STAN==0 && PARAMETR1==1) logi("(100) !!! Proba zablokowania
                                silnikow zakonczona niepowodzeniem !!!");
if(ROZKAZ==100 && STAN==0 && PARAMETR1==0) logi("(100) !!! Proba odblokowania
                                silnikow zakonczona niepowodzeniem !!!");
if(ROZKAZ==200 && STAN==1) if(LOGI==1) logi("(200) Zgloszenie pozycji do pytajacego ");
if(ROZKAZ==200 && STAN==0) logi("(200) !!! Zglosenie braku gotowosci na zadanie
                                pytajacego !!!");
if(ROZKAZ==300 && STAN==1) logi("(300) Poprawienie pozycji
                                (i ruch dobowy) ");
if(ROZKAZ==300 && STAN==0) logi("(300) !!! Zgloszenie braku gotowosci na
                                zadanie pytajacego !!!");
if(ROZKAZ==400 && STAN==1 && PARAMETR1==1) logi("(400) Uruchomienie ruchu dobowego ");
if(ROZKAZ==400 && STAN==1 && PARAMETR1==0) logi("(400) Zatrzymanie ruchu dobowego ");
if(ROZKAZ==400 && STAN==0) logi("(400) !!! Zgloszenie braku gotowosci na zadanie
                                pytajacego !!!");
if(ROZKAZ==430 && STAN==1) logi("(430) Przyspieszenie ruchu  dobowego
                                przez autoguider ");
if(ROZKAZ==430 && STAN==0) logi("(430) !!! Zgloszenie braku gotowosci na zadanie
                                pytajacego !!!");
if(ROZKAZ==460 && STAN==1) logi("(460) Spowolnienie ruchu  dobowego
                                przez autoguider ");
if(ROZKAZ==460 && STAN==0) logi("(460) !!! Zgloszenie braku gotowosci na zadanie
                                pytajacego !!!");
if(ROZKAZ==500 && STAN==1) logi("(500) Poprawienie pozycji (silnik stop) ");
if(ROZKAZ==500 && STAN==0) logi("(500) !!! Zgloszenie braku gotowosci na zadanie
                                pytajacego !!!");
if(ROZKAZ==600 && STAN==1) logi("(600) Bazowanie i wprowadzenie poprawki ");
if(ROZKAZ==600 && STAN==0) logi("(300) !!! Zgloszenie braku gotowosci na zadanie
                                pytajacego !!!");


if(ROZKAZ==-1  && STAN==0)
    {
    sprintf(str,"!!!! Otrzymano bledny kod rozkazu od %d !!!!",KTO);
    logi(str);
    }
```

Część odpowiedzialna za wysłanie odpowiedzi przez kolejkę komunikatów do programu, który wysłał do SRTA rozkaz.

```
bzero(buf,sizeof(buf));
mbuf.mtype=KTO;  /* Wysylamy do glownego*/
sprintf(buf,"%d %d %d %22.15G",PID,ROZKAZ,STAN,PARAMETR2);
strcpy(mbuf.mtext,buf);
msgsnd(mqId, &mbuf, strlen(buf),0);
}
```

Rozdział 5

Komendy wysyłane do sterownika silnika

Wartości stanów logicznych:

ACK = x06,

NAK = x15,

TOUT = x16,

CAN = x18.

5.1 Komenda x00 – Zablokowanie silnika

Wysyłamy do sterownika cztery bajty:

ESC – rozpoczęcie komendy,

x01 – numer osi (silnika),

x00 – komenda zablokowania silnika,

x1A – suma kontrolna.

Otrzymujemy ze sterownika:

ACK – potwierdzenie otrzymania i wykonania komendy,

lub

NAK – nieprawidlowa suma kontrolna,

lub

TOUT – niekompletna transmisja.

5.2 Komenda x01 – Odblokowanie silnika

Wysyłamy do sterownika cztery bajty:

ESC – rozpoczęcie komendy,

x01 – numer osi (silnika),

x01 – komenda odblokowania silnika,

x1B – suma kontrolna.

Otrzymujemy ze sterownika:

ACK – potwierdzenie otrzymania i wykonania komendy,

lub

NAK – nieprawidlowa suma kontrolna,

lub

TOUT – niekompletna transmisja.

5.3 Komenda x03 – Logowanie do silnika

Wysyłamy do sterownika cztery bajty:

ESC – rozpoczęcie komendy,

x01 – numer osi (silnika),

x03 – komenda logowania do silnika,

x19 – suma kontrolna.

Otrzymujemy ze sterownika:

ACK – potwierdzenie otrzymania i wykonania komendy,

lub

NAK – nieprawidlowa suma kontrolna,

lub

TOUT – niekompletna transmisja.

lub

CAN – potwierdzenie transmisji ale juz jesteśmy zalogowanie do silnika.

5.4 Komenda x06 – Czytanie wersji oprogramowania sterownika

Wysyłamy do sterownika cztery bajty:

ESC – rozpoczęcie komendy,

x01 – numer osi (silnika),

x06 – komenda o podanie wersji oprogramowania sterownika,

x1C – suma kontrolna.

Otrzymujemy ze sterownika:

ACK – potwierdzenie otrzymania komendy,

x36 – znak **6**,

x33 – znak **3**,

x31 – znak **1**,

x20 – znak spacji,

x56 – znak **V**,

x20 – znak spacji,

x35 – znak **5**,

x2E – znak **.**,

x31 – znak **1**,

x30 – znak **0**,

x61 – znak **a**,

x20 – znak spacji,

BCC – suma kontrolna.

lub

NAK – nieprawidlowa suma kontrolna,

lub

TOUT – niekompletna transmisja.

5.5 Komenda x07 – Informacje o silniku

Wysyłamy do sterownika cztery bajty:

ESC – rozpoczęcie komendy,

x01 – numer osi (silnika),

x07 – komenda o stan silnika,

x1D – suma kontrolna.

Otrzymujemy ze sterownika:

ACK – potwierdzenie otrzymania komendy,

bajt1 – bajt zawierający flagi błędów,

bajt2 – bajt zawierający flagi błędów,

bajt3 – bajt zawierający flagi stanu,

bajt4 – bajt zawierający flagi stanu,

bajt5 – bajt zawierający flagi stanu,

bajt6 – bajt zawierający flagi stanu,

bajt7 – bajt zawierający flagi stanu,

bajt8 – bajt zawierający flagi stanu,

bajt9 – młodszy bajt zawierający actualną prędkość,

bajt10 – starszy bajt zawierający actualną prędkość,

bajt11 – bajt zawierający flagi stanu wejscia/wyjscia,

bajt12 – zarezerwowane

bajt13 – najmłodszy bajt aktualnej pozycji,

bajt14 – młodszy bajt aktualnej pozycji,

bajt15 – starszy bajt aktualnej pozycji,

bajt16 – najstarszy bajt aktualnej pozycji,

bajt17 – zarezerwowane,

bajt18 – zarezerwowane,

bajt19 – zarezerwowane,

bajt20 – zarezerwowane,

BCC – suma kontrolna.

lub

NAK – nieprawidlowa suma kontrolna,

lub

TOUT – niekompletna transmisja.

5.6 Komenda x0D – Wywołanie programu w BIASie od zadanej linii

Wysyłamy do sterownika cztery bajty:

ESC – rozpoczęcie komendy,

x01 – numer osi (silnika),

x0D – komenda wywołania programu w BIASIE,

bajt1 – młodszy bajt zawierający numer linii,

bajt2 – starszy bajt zawierający numer linii,

BCC – suma kontrolna.

Otrzymujemy ze sterownika:

ACK – potwierdzenie otrzymania i wykonania komendy,

lub

NAK – nieprawidlowa suma kontrolna,

lub

TOUT – niekompletna transmisja.

lub

CAN – potwierdzenie transmisji ale numer linii poza zakresem (0-1499).

5.7 Komenda x4C – Wpisanie linii programu do BIASu

Wysyłamy do sterownika cztery bajty:

ESC – rozpoczęcie komendy,

x01 – numer osi (silnika),

x4C – komenda wpisania linii programu do BIASU do silnika,

x01 – tryb zapisu,

bajt1 – młodszy bajt zawierający numer linii,

bajt2 – starszy bajt zawierający numer linii,

x20 – komenda BIASa odpowiadająca za ilość inkramentów o jaką ma się obrócić silnik,

bajt4 – najmłodszy bajt ilości inkramentów,

bajt5 – młodszy bajt ilości inkramentów,

bajt6 – starszy bajt ilości inkramentów,

bajt7 – najstarszy bajt ilości inkramentów,

x0 – niewykorzystane bajty przy tej komendzie,

x0 – niewykorzystane bajty przy tej komendzie,

x0 – niewykorzystane bajty przy tej komendzie,

BCC – suma kontrolna.

Otrzymujemy ze sterownika:

ACK – potwierdzenie otrzymania i wykonania komendy,

lub

NAK – nieprawidlowa suma kontrolna,

lub

TOUT – niekompletna transmisja.

lub

CAN – potwierdzenie transmisji ale coś nie pasuje.

Rozdział 6

Program umieszczony w sterowniku silnika

W sterowniku silniku znajduje się program **TELESKOP.ABS**. Za pomocą programu **EASYRIDER ver 5.13A** program możemy edytować i przesyłać do sterownika.

6.1 Program TELESKOP-SRTA.ASB

Pierwsza część programu odpowiada za zatrzymanie i postój silnika.

```
 PROG_START:
0     end of program,   mode =  1
```

Zdefiniowane są przyspieszenia i opóźnienia ruchu silnika

```
1     acceleration   = 500    rpm/s
2     deceleration   = 500    rpm/s
```

Część programu odpowiedzialna za przejaz o zadaną ilość incrementów a następnie skok do następnej części.

```
 START_POS:
3     speed           = 500   rpm
4     position =  163840      INCR
5     start axis
6     move incremental position
7     jump     ruchdob
8     end of program,   mode =  1
```

Ta część programu odpowiada za ruch dobowy silnika

```
 ruchdob:
9     NOP
10    acceleration   = 500    rpm/s
11    deceleration   = 500    rpm/s
12    speed           = 500   rpm
```

```
13    position =  5           INCR
14    start axis
15    move incremental position
16    wait time    150    ms
17    speed           = 100   rpm
18    position =  5           INCR
19    start axis
20    move incremental position
21    wait time    150    ms
22    jump     ruchdob
23    end of program,   mode =  1
```

Część programu odpowiedzialna za najazd na zadaną pozycje i zatrzymanie silnika

```
 napoz-stop:
24    NOP
25    acceleration   = 500    rpm/s
26    deceleration   = 500    rpm/s
27    speed           = 500   rpm
28    position =  -24748      INCR
29    start axis
30    move incremental position
31    end of program,   mode =  1
```

Część programu odpowiedzialna za najazd na zadaną pozycje a następnie silnik wykonuje ruch dobowy.

```
 napoz-ruch:
32    NOP
33    acceleration   = 500    rpm/s
34    deceleration   = 500    rpm/s
35    speed           = 500   rpm
36    position =  -24748      INCR
37    start axis
38    move incremental position
39    jump     ruchdob
40    end of program,   mode =  1
```

Część programu odpowiadająca za ruch silnika z podwójną prędkością ruchu dobowego

```
 ruch+dob:
41    NOP
42    acceleration   = 500    rpm/s
43    deceleration   = 500    rpm/s
44    speed           = 500   rpm
45    position =  5           INCR
46    start axis
47    move incremental position
48    wait time    74     ms
49    speed           = 100   rpm
50    position =  5           INCR
51    start axis
52    move incremental position
53    wait time    74     ms
54    jump     ruch+dob
55    end of program,   mode =  1
```

Część programu odpowiadająca za ruch silnika z połowó prędkości ruchu dobowego

```
 ruch-dob:
56    NOP
57    acceleration   = 500    rpm/s
58    deceleration   = 500    rpm/s
59    speed           = 500   rpm
60    position =  5           INCR
61    start axis
62    move incremental position
63    wait time    300    ms
64    speed           = 100   rpm
65    position =  5           INCR
66    start axis
67    move incremental position
68    wait time    300    ms
69    jump     ruch-dob
70    end of program,   mode =  1
```

6.2 Profil silnika

Profil przesyłany do silników. Wydruk pochodzi z programu **EASYRIDER ver 5.13A**

```
  program start  = PROG_START
deactive reaction
  BIAS-program   = start after reactiv. at program start     0
  PLC-program    = switch off execution at deactivation      0
program definitions
  name = 2. BIAS application example feed
  datum:  01.07.1997
  version = 1.0
unit (drive)
  cog count output drive        = 100
  cog count input drive         = 100
  high resolver resolution      = Y
  distance per output turn      = 16384 INCR
unit (X 40)
  cog count input drive         = 100
  cog count output drive        = 100
  encoder resolution            = 4096
  distance per output turn      = 4096  INCR
digital inputs
  input   X10.2  = 3  BIAS-input 2
  input   X10.4  = 3  BIAS-input 4
  input   X10.11 = 0  start input  (BIAS)
  input   X10.14 = 0  BIAS-input 14
  input   X10.15 = 0  BIAS-input 15
  input   X10.24 = 1  referencesensor
  input   X10.25 = 3  BIAS-input 25
digital outputs
  output  X10.12 = 0  position reached
  output  X10.13 = 1  initialized
  output  X10.20 = 3  trail monitoring
  output  X10.23 = 0  active OK
631 digital inputs
  input   X10.8  = 0  BIAS-input 8
  input   X10.9  = 0  BIAS-input 9
  input   X10.10 = 0  BIAS-input 10
```

```
631 digital outputs
  output  X10.5  = 0  drive ready
  output  X10.6  = 0  position reached
analoge inputs
  speed control   1 V set value = -400 rpm
X40 connector
  X40 mode                      = output
  encoder resolution            = 0
  count direction               = -

    profile number 0

    synchronous profile= 0
    increm. correction = N
    master dist.p.cycle= 4096         INCR
    slave dist. p.cycle= 4096         INCR
    supporting points  = 128

    profile number 1

    synchronous profile= 0
    increm. correction = N
    master dist.p.cycle= 4096         INCR
    slave dist. p.cycle= 4096         INCR
    supporting points  = 128

    profile number 2

    synchronous profile= 0
    increm. correction = N
    master dist.p.cycle= 4096         INCR
    slave dist. p.cycle= 4096         INCR
    supporting points  = 128

    profile number 3

    synchronous profile= 0
    increm. correction = N
    master dist.p.cycle= 4096         INCR
    slave dist. p.cycle= 4096         INCR
    supporting points  = 128

    profile number 4

    synchronous profile= 0
    increm. correction = N
    master dist.p.cycle= 4096         INCR
    slave dist. p.cycle= 4096         INCR
    supporting points  = 128

    profile number 5

    synchronous profile= 0
    increm. correction = N
    master dist.p.cycle= 4096         INCR
    slave dist. p.cycle= 4096         INCR
    supporting points  = 128

    profile number 6

    synchronous profile= 0
    increm. correction = N
```

```
master dist.p.cycle= 4096         INCR
slave dist. p.cycle= 4096         INCR
supporting points  = 128

profile number 7

synchronous profile= 0
increm. correction = N
master dist.p.cycle= 4096         INCR
slave dist. p.cycle= 4096         INCR
supporting points  = 128

profile number 8

synchronous profile= 0
increm. correction = N
master dist.p.cycle= 4096         INCR
slave dist. p.cycle= 4096         INCR
supporting points  = 128

profile number 9

synchronous profile= 0
increm. correction = N
master dist.p.cycle= 4096         INCR
slave dist. p.cycle= 4096         INCR
supporting points  = 128

profile number 10

synchronous profile= 0
increm. correction = N
master dist.p.cycle= 4096         INCR
slave dist. p.cycle= 4096         INCR
supporting points  = 128

profile number 11

synchronous profile= 0
increm. correction = N
master dist.p.cycle= 4096         INCR
slave dist. p.cycle= 4096         INCR
supporting points  = 128

profile number 12

synchronous profile= 0
increm. correction = N
master dist.p.cycle= 4096         INCR
slave dist. p.cycle= 4096         INCR
supporting points  = 128

profile number 13

synchronous profile= 0
increm. correction = N
master dist.p.cycle= 4096         INCR
slave dist. p.cycle= 4096         INCR
supporting points  = 128

profile number 14
```

```
synchronous profile= 0
increm. correction = N
master dist.p.cycle= 4096         INCR
slave dist. p.cycle= 4096         INCR
supporting points  = 128

profile number 15

synchronous profile= 0
increm. correction = N
master dist.p.cycle= 4096         INCR
slave dist. p.cycle= 4096         INCR
supporting points  = 128
```

Rozdział 7

Program incom1

Program **incom1** łączy kolejkę **FIFO** z urządzeniem **MODEMDIVICE**. Odbiera dane z portu **/dev/ttyS0** i przesyła je do kolejki FIFO która nazywa się **ścieżka do programu>/fifo/icom1**. Gdy nie ma danych na porcie program się zatrzymuje i oczekuje na następne dane.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>

#define MODEMDEVICE "/dev/ttyS0"
#define _POSIX_SOURCE 1 /* POSIX compliant source */
#define FALSE 0
#define TRUE 1

volatile int STOP=FALSE;
```

Program główny

```
main()
{

  int fifoo,fd;
  char buf[255];
```

Otwarcie kolejki FIFO tylko do zapisu

```
 fifoo=open("../fifo/icom1",O_WRONLY);
```

Otwarcie porty **MODEMDEVICE** do zapisu i odczytu bez nadzoru terminalowego.

```
 fd = open(MODEMDEVICE, O_RDWR | O_NOCTTY );
```

Jeżeli nie mogę otworzyć to przerywam program

```
 if (fd <0) {perror(MODEMDEVICE); exit(-1); }
```

Pętla nieskończona

```
 while (STOP==FALSE)
 {
```

w której czytamy po jednym bajcie z urządzenia **MODEMDEVICE**

```
   read(fd,&buf,1);
```

i zapisywamy go do kolejki **FIFO**

```
   write(fifoo,buf,1);
 }
```

Zamknięcie kolejek i zakończenie pracy.

```
 close(fifoo);
 close(fd);
}
```

Rozdział 8

Program outcom1

Program **outcom1** łączy kolejkę **FIFO** z urządzeniem **MODEMDIVICE**. Odbiera dane z kolejki FIFO która nazywa się **<ścieżka do programu>/fifo/icom1** i przesyła je do portu **/dev/ttyS0**. Gdy nie ma danych w kolejce FIFO program się zatrzymuje i oczekuje na następne dane.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>

#define MODEMDEVICE "/dev/ttyS0"
#define _POSIX_SOURCE 1 /* POSIX compliant source */
#define FALSE 0
#define TRUE 1

volatile int STOP=FALSE;

main()
{

  int fifoo,fd;
  char buf[255],buf1[255];
```

Otwarcie kolejki FIFO tylko do czytania

```
 fifoo=open("../fifo/ocom1",O_RDONLY);
```

Otwarcie porty **MODEMDEVICE** do zapisu i odczytu bez nadzoru terminalowego.

```
 fd = open(MODEMDEVICE, O_RDWR | O_NOCTTY );
```

Jeżeli nie mogę otworzyć to przerwamy program

```
 if (fd <0) {perror(MODEMDEVICE); exit(-1); }
```

Pętla nieskończona

```
 while (STOP==FALSE) {
```

w której czytamy po jednym bajcie z koleki **FIFO**

```
   read(fifoo,&buf,1);
```

i zapisanie go do urządzenia **MODEMDEVICE**

```
   write(fd,buf,1);
 }
```

Zamknięcie kolejek i urządzeń

```
 close(fifoo);
 close(fd);
}
```

Rozdział 9

Program nadaj

Program **nadaj** jest prostym programem do komunikacji z programem **SRTA** poprzez kolejke komunikatów. Program jest bardzo wygodny do sprawdzania komunikacji na poziomie komunikatów. Na początku nagłówki i definiujemy zmienne

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int PID;
struct msqbuff
{
    long mtype;
    char mtext[300];
} mbuf;

main()
{
char buf[300];
key_t klucz;
int mqId,a,b,aa,bb,cc;
double dd,pop;
```

Otwieramy kolejkę komunikatów

```
klucz = ftok(".",'Q');
if((mqId=msgget((key_t) klucz, 0600 |IPC_CREAT)) == -1)
{
    perror("msgget_tworz");
    return -1;
}
```

Musimy podać swój unikatowy numer procesu np. 23

```
 printf("Podaj PID nadawcy:");
scanf("%d",&PID);
```

W pętli nieskończonej program czeka na podanie z klawiatury numeru rozkazu

```
    for(;;){
bzero(buf,sizeof(buf));
 mbuf.mtype=10;
```

```
printf("\n\n\n Wysylanie rozkazow do SRTA:\n");
printf(" 100 - zablokuj/odblokuj silniki\n");
printf(" 200 - odczytanie pozycji z silnika\n");
printf(" 300 - poprawki autoguidera w sek\n");
printf(" 400 - ruch w kacie godzinnym \n");
printf(" 430 - 2x ruch w kacie godzinnym \n");
printf(" 460 - 1/2 ruchu w kacie godzinnym \n");
printf(" 500 - poprawki do pozycji w sek \n");
printf(" 600 - zerowanie pozycji i poprawka \n");
printf(" 700 - logowanie do silnika \n");
printf(" 800 - sprawdz gotowosc,\n");
printf(" 900 - zakoncz program SRTA\n");
printf(" 0 - zakoncz wysylanie\n\n\n");

printf("\n Podaj nr rozkazu: ");
```

Odczytanie podanej liczby

```
b=0;
scanf("%d",&a);
```

Jeżeli była to liczba 100 to program prosi o dodatkowy parametr i wpisuje rozkaz do bufora

```
if(a==100)
{
printf(" Podaj parametr ( 1-zablokuj, 0-odblokuj ):");
scanf("%d",&b);
sprintf(buf," %d %d %d\n",PID,a,b);
}
```

Jeżeli była to liczba 400 to program prosi o dodatkowy parametr i wpisuje rozkaz do bufora

```
if(a==400)
{
printf(" Podaj parametr ( 1-wlacz, 0-wylacz ):");
scanf("%d",&b);
sprintf(buf," %d %d %d\n",PID,a,b);
printf("zrobione\n");
}
```

Jeżeli była to liczba 300 to program prosi o podanie poprawki w sekundach łuku a następnie wpisuje rozkaz do bufora

```
if(a==300)
{
printf(" Podaj poprawke (sek):");
scanf("%lf",&pop);
sprintf(buf," %d %d %lf\n",PID,a,pop);
}
```

tak samo kiedy wystąpi liczba 500

```
if(a==500)
{
printf(" Podaj poprawke (sek):");
scanf("%lf",&pop);
sprintf(buf," %d %d %lf\n",PID,a,pop);
}
```

Jeżeli nie była to liczba żadna z powyższych to rozkaz zapisywany jest do bufora

```
if(a!=300 && a!=100 && a!=400 && a!=500)
sprintf(buf," %d %d %d\n",PID,a,b);
```

Jeżeli tą liczbą było zero to koniec programu

```
if( a==0) return 1;
```

Wysłanie komunikatu

```
strcpy(mbuf.mtext,buf);
msgsnd(mqId, &mbuf, strlen(buf),0);
```

Odbiór potwierdzenia

```
bzero(mbuf.mtext,sizeof(mbuf.mtext));
if(msgrcv(mqId, &mbuf, sizeof(mbuf.mtext), PID,0)==-1)
    {
    perror("blad odczytu");
    return 2;
    }
```

Jeśli prawidłowo odebraliśmy komunikat to czytamy jego parametry

```
else
{
strcpy(buf,mbuf.mtext);
aa=0;bb=0;cc=0;dd=0;
sscanf(buf,"%d %d %d %lf\n", &aa, &bb, &cc,&dd);
printf("----> Odpowiedz:\n");
```

i wyświetlamy na ekranie (aa=10 oznacza że są czytane komunikaty tylko z programu SRTA). Gdy jest wszystko dobrze

```
if(aa==10 && bb==a && cc==1)
printf("Odebralem od %d potwierdzenie wykonania rozkazu %d zwrocony parametr (%22.15G)\n",aa,bb,dd);
```

Gdy zgłoszony został błąd wykonania rozkazu

```
if(aa==10 && bb==a && cc==0)
printf("Odebralem od %d blad wykonania rozkazu %d\n",aa,bb);
```

Gdy program SRTA nie rozpoznał wysłanego rozkazu

```
if(aa==10 && bb==-1 && cc==0)
printf("Odebralem od %d blad: Nieznany rozkazu %d\n",aa,a);
}
}
}
```

Rozdział 10

Program odbior

Program **odbior** jest pomocny przy czyszceniu kolejki komunikatów oraz przy kontroli odsyłanych przez program SRTA komunikatów.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

struct msqbuff
{
    long mtype;
    char mtext[300];
} mbuf;

main()
{
char buf[300];
long type;

key_t klucz;
int mqId;
int licz=0;
```

Otwarcie kolejki komunikatów

```
klucz = ftok(".",'Q');
if((mqId=msgget((key_t) klucz, 0600 |IPC_CREAT)) == -1)
{
    perror("msgget_tworz");
    return -1;
}
```

Program wymaga podania jaki typ komunikatu ma być odebrany (0 - wszystkie , 10 - z programu SRTA)

```
printf("\n Podaj typ komunikatu jaki chcesz odebrac: ");
scanf("%ld", &type);
```

W nieskończonej pętli odczytywane są komunikaty z kolejki. Program się zatrzymuje i czeka aż nadejdzie następny komunikat.

```
    for(;;)
    {
licz=licz+1;
bzero(mbuf.mtext,sizeof(mbuf.mtext));
if(msgrcv(mqId, &mbuf, sizeof(mbuf.mtext), type,0)==-1)
    {
    perror("blad odczytu");
    return 2;
    }
else
{
printf("Odebralem %d komnikat typu %d: %s\n",licz,mbuf.mtype, mbuf.mtext);
}
}
}
```

Rozdział 11

Program serwis-SRTA

Program **serwis-SRTA** jest okienkową aplikacją pozwalającą komunikować się z silnikiem za pomocą prostych funkcji. Uruchomiony w terminalu pozwala efektywnie kierować osią kąta godzinnego teleskopu obsłudze technicznej.

11.1 Nagłówki

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <curses.h>
#include <time.h>
#include <signal.h>
#include <sys/io.h>
```

11.1.1 Definicje stałych

```
#define KROK_SK1      4
#define KIERUNEK_SK1  8
#define KROK_SK2      16
#define KIERUNEK_SK2  32
#define LUZ_SK1       64
#define LUZ_SK2       128
#define ADRES 0x278
WINDOW *okienko,*okienko1,*okienko2,*okienko3;
WINDOW *nowe;
FILE *input, *input1;

int POZST=0,POZM=0,POZS=0,ID;
int APOZST=0,APOZM=0,APOZS=0;
int RPOZST=1,RPOZM=0,RPOZS=0;
int APOZSTNEW=0,APOZMNEW=0,APOZSNEW=0;
int znacznik=0,sttop;
int MIN1=0,MAX1=10000,POZYCJA1=1234,OGNISKO1=5000,KIERUNEK1=1;
int AKTYWNE=0,POZ=0;
int KOL[20],KOLSTAN[20],ii,PIERWSZE=1;
int MIN2=0,MAX2=10000,POZYCJA2=2345,OGNISKO2=2000,KIERUNEK2=1;
```

```
int KROK=0,oldCOLS,oldLINES,KOLPOZ;
```

11.1.2 Deklaracje funkcji pomocniczych

```
int wyslij(char buf[300], int mqID);
void menu1(void);
void menu2(void);
void menu3(void);
void menu4(void);
void start(void);
void setflag(int sig);
```

11.1.3 Definicja identyfikatora w kolejce

```
int PID=33;
struct msqbuff
{
    long mtype;
    char mtext[300];
} mbuf;
```

11.2 Główny program

```
main()
{
char get,s[20];
int i;
pid_t pid;

static struct sigaction act;
```

11.2.1 Otwarcie kolejki komunikatów

```
char buf[300];
key_t klucz;
int mqId,a,b,aa,bb,cc;
double dd,pop;
klucz = ftok(".",'Q');
if((mqId=msgget((key_t) klucz, 0600 |IPC_CREAT)) == -1)
{
    perror("msgget_tworz");
    return -1;
}
ID=mqId;
KOLPOZ=2;
```

11.2.2 Czyszczenie kolejki komunikatów

```
    for(i=0;i<100;i=i+1)
    {
```

```
bzero(mbuf.mtext,sizeof(mbuf.mtext));
if(msgrcv(mqId, &mbuf, sizeof(mbuf.mtext),0,IPC_NOWAIT)==-1)
    {
    i=1000;
    }
     }
```

11.2.3 Uruchomienie programu SRTA

```
    switch(pid=fork())
    {
    case -1:
    printf("Nie moge otworzyc procesu\n");
    return 1;
    break;
    case 0:
    execlp("./SRTA",(char *)0);
    printf("Nie odnaleziono programu SRTA\n");
    return 1;
    break;
    }
```

11.2.4 Ustawienia alarmu

```
act.sa_handler = setflag;
sigaction(SIGALRM, &act, NULL);
```

11.2.5 Zablokowanie silnika

Przy starcie programu jak i na końcu blokujemy silnik. Korzystamy z funkcji pomocniczej **wyslij**, która opisana jest w dalszych podrozdziałach.

```
 bzero(buf,sizeof(buf));
 mbuf.mtype=10;
 sprintf(buf," %d 100 1\n",PID);
 wyslij(buf,mqId);
```

11.2.6 Inicjacja ekranu (ncurses)

```
for(ii=0;ii<20;ii=ii+1)
    {
KOLSTAN[ii]=2;
    }
    KOLSTAN[0]=6;

start();
noecho();

    for(ii=0;ii<20;ii=ii+1)
    {
KOL[ii]=KOLSTAN[ii];
    }
    KOL[AKTYWNE]=11;
```

```
menu1();
menu2();
menu3();
menu4();
refresh();
nodelay(nowe,FALSE);
sttop=0;
```

11.3 Pętla główna

W pętli głównej czytamy znaki nadchodzące z klawiatury. Jeżeli naciśnięty zostanie klawisz **z** to program się zakończy.

```
while(get!=122)
{
bzero(buf,sizeof(buf));
mbuf.mtype=10;
alarm(1);
get=getch();
alarm(0);
switch(get)
{
```

11.3.1 Naciśnięcie klawisza TAB lub strzałek prawo/lewo

```
    case(9):case(27):
    znacznik=1;
    if(get==27)
{
get=getch();
if(get==91)
    {
    get=getch();

    if(get==68) AKTYWNE=AKTYWNE-1;
    if(get==67) AKTYWNE=AKTYWNE+1;
    }
}
    if(get==9) AKTYWNE=AKTYWNE+1;
    if(AKTYWNE>16) AKTYWNE=0;
    if(AKTYWNE<0) AKTYWNE=16;
    if(KOLSTAN[0]==6)
{
if(AKTYWNE==2) AKTYWNE=15;
if(AKTYWNE==14) AKTYWNE=1;
}
/* Powi"azanie tabeli KOL z aktywnymi polami
    KOLZABLOKOWANY --- KOL[0]
    KOLODBLOKOWANY --- KOL[1]
    KOLSZYBKI      --- KOL[2]
    KOLNORMALNY    --- KOL[3]
    KOLWOLNY       --- KOL[4]
    KOLSTOP        --- KOL[5]
    KOLRPOZST       --- KOL[6]
    KOLRPOZM        --- KOL[7]
    KOLRPOZS        --- KOL[8]
    KOLDOPRZODU    --- KOL[9]
```

```
    KOLDOTYL       --- KOL[10]
    KOLNAJAZDNAPOZ --  KOL[11]
    KOLPOZST       --- KOL[12]
    KOLPOZM        --- KOL[13]
    KOLPOZS        --- KOL[14]
    KOLZAPAMIENTAJ --- KOL[15]
    KOLLOGOWANIE   --- KOL[16]
*/

    for(ii=0;ii<20;ii=ii+1)
    {
KOL[ii]=KOLSTAN[ii];
    }
    KOL[AKTYWNE]=11;
    menu4();
    break;
```

11.3.2 Naciśnięcie klawisza ENTER

Każdorazowe naciśnięcie klawisza **ENTER** na aktywnym polu powoduje wysłanie rozkazu do programu **SRTA**.

```
    case(10):
    switch(AKTYWNE)
    {
```

Jeżeli aktywne jest pole blokowanie silnika

```
    case(0):
    KOLSTAN[0]=6;
    KOLSTAN[1]=2;
    KOLSTAN[5]=2;
    KOLSTAN[2]=2;
    KOLSTAN[3]=2;
    KOLSTAN[4]=2;
    for(ii=0;ii<20;ii=ii+1)
    {
KOL[ii]=KOLSTAN[ii];
    }
    KOL[AKTYWNE]=12;
    menu4();
    sprintf(buf," %d 100 1\n",PID);
    wyslij(buf,mqId);
    for(ii=0;ii<20;ii=ii+1)
    {
KOL[ii]=KOLSTAN[ii];
    }
    KOL[AKTYWNE]=11;
    menu4();
    break;
```

Jeżeli aktywne jest pole odblokowanie silnika

```
    case(1):
    KOLSTAN[1]=6;
    KOLSTAN[0]=2;
    KOLSTAN[5]=6;
    for(ii=0;ii<20;ii=ii+1)
    {
KOL[ii]=KOLSTAN[ii];
```

```
    }
    KOL[AKTYWNE]=12;
    menu4();
    sprintf(buf," %d 100 0\n",PID);
    wyslij(buf,mqId);
    for(ii=0;ii<20;ii=ii+1)
    {
KOL[ii]=KOLSTAN[ii];
    }
    KOL[AKTYWNE]=11;
    menu4();
    break;
```

Jeżeli aktywne jest pole ruch dobowy z podwójną prędokścią

```
    case(2):
    KOLSTAN[2]=6;
    KOLSTAN[3]=2;
    KOLSTAN[4]=2;
    KOLSTAN[5]=2;
    for(ii=0;ii<20;ii=ii+1)
    {
KOL[ii]=KOLSTAN[ii];
    }
    KOL[AKTYWNE]=12;

    menu4();
    sprintf(buf," %d 430 1\n",PID);
    wyslij(buf,mqId);
    for(ii=0;ii<20;ii=ii+1)
    {
KOL[ii]=KOLSTAN[ii];
    }
    KOL[AKTYWNE]=11;
    menu4();

    break;
```

Jeżeli aktywne jest pole ruch dobowy z nominalną prędkością

```
    case(3):
    KOLSTAN[2]=2;
    KOLSTAN[3]=6;
    KOLSTAN[4]=2;
    KOLSTAN[5]=2;
    for(ii=0;ii<20;ii=ii+1)
    {
KOL[ii]=KOLSTAN[ii];
    }
    KOL[AKTYWNE]=12;
    menu4();
    sprintf(buf," %d 400 1\n",PID);
    wyslij(buf,mqId);
    for(ii=0;ii<20;ii=ii+1)
    {
KOL[ii]=KOLSTAN[ii];
    }
    KOL[AKTYWNE]=11;
    menu4();
    break;
```

Jeżeli aktywne jest pole ruch dobowy z połową prędkości

```
    case(4):
    KOLSTAN[2]=2;
    KOLSTAN[3]=2;
    KOLSTAN[4]=6;
    KOLSTAN[5]=2;
    for(ii=0;ii<20;ii=ii+1)
    {
KOL[ii]=KOLSTAN[ii];
    }
    KOL[AKTYWNE]=12;
    menu4();
    sprintf(buf," %d 460 1\n",PID);
    wyslij(buf,mqId);
    for(ii=0;ii<20;ii=ii+1)
    {
KOL[ii]=KOLSTAN[ii];
    }
    KOL[AKTYWNE]=11;
    menu4();
    break;
```

Jeżeli aktywne jest pole zatrzymanie ruchu dobowego

```
    case(5):
    KOLSTAN[2]=2;
    KOLSTAN[3]=2;
    KOLSTAN[4]=2;
    KOLSTAN[5]=6;
    for(ii=0;ii<20;ii=ii+1)
    {
KOL[ii]=KOLSTAN[ii];
    }
    KOL[AKTYWNE]=12;
    menu4();
    sprintf(buf," %d 400 0\n",PID);
    wyslij(buf,mqId);
    for(ii=0;ii<20;ii=ii+1)
    {
KOL[ii]=KOLSTAN[ii];
    }
    KOL[AKTYWNE]=11;
    menu4();
    break;
```

Jeżeli aktywne jest pole jazdy do przodu

```
    case(9):
    for(ii=0;ii<20;ii=ii+1)
    {
KOL[ii]=KOLSTAN[ii];
    }
    KOL[AKTYWNE]=12;
    menu4();
    sprintf(buf," %d 500 %G\n",PID,(RPOZST*60.*60.+RPOZM*60.+RPOZS));
    wyslij(buf,mqId);
    KOLSTAN[2]=2;
    KOLSTAN[3]=2;
    KOLSTAN[4]=2;
    KOLSTAN[5]=6;
```

```
    for(ii=0;ii<20;ii=ii+1)
    {
KOL[ii]=KOLSTAN[ii];
    }
    KOL[AKTYWNE]=11;
    menu4();

    break;
```

Jeżeli aktywne jest pole jazdy do tył

```
    case(10):
    for(ii=0;ii<20;ii=ii+1)
    {
KOL[ii]=KOLSTAN[ii];
    }
    KOL[AKTYWNE]=12;
    menu4();
    sprintf(buf," %d 500 %G\n",PID,-1.*(RPOZST*60.*60.+RPOZM*60.+RPOZS));
    wyslij(buf,mqId);
    KOLSTAN[2]=2;
    KOLSTAN[3]=2;
    KOLSTAN[4]=2;
    KOLSTAN[5]=6;

    for(ii=0;ii<20;ii=ii+1)
    {
KOL[ii]=KOLSTAN[ii];
    }
    KOL[AKTYWNE]=11;
    menu4();

    break;
```

Jeżeli aktywne jest pole najazd na pozycje

```
    case(11):
    for(ii=0;ii<20;ii=ii+1)
    {
KOL[ii]=KOLSTAN[ii];
    }
    KOL[AKTYWNE]=12;
    menu4();
    sprintf(buf," %d 500 %G\n",PID,(POZST*60.*60.+POZM*60.+POZS)-(APOZST*60.*60.+APOZM*60.+APOZS));
    wyslij(buf,mqId);
    KOLSTAN[2]=2;
    KOLSTAN[3]=2;
    KOLSTAN[4]=2;
    KOLSTAN[5]=6;

    for(ii=0;ii<20;ii=ii+1)
    {
KOL[ii]=KOLSTAN[ii];
    }
    KOL[AKTYWNE]=11;
    menu4();

    break;
```

Jeżeli aktywne jest pole zapamiętaj pozycje

```
    case(15):
    for(ii=0;ii<20;ii=ii+1)
    {
KOL[ii]=KOLSTAN[ii];
    }
    KOL[AKTYWNE]=12;
    POZST=APOZST;
    POZM=APOZM;
    POZS=APOZS;
    menu4();
    KOL[AKTYWNE]=11;
    menu4();
    break;
```

Jeżeli aktywne jest pole logowania

```
    case(16):
    for(ii=0;ii<20;ii=ii+1)
    {
KOL[ii]=KOLSTAN[ii];
    }
    KOL[AKTYWNE]=12;
    menu4();
    sprintf(buf," %d 700 0\n",PID);
    wyslij(buf,mqId);
    for(ii=0;ii<20;ii=ii+1)
    {
KOL[ii]=KOLSTAN[ii];
    }
    KOL[AKTYWNE]=11;
    menu4();
    break;

    }
```

11.3.3 Naciśnięcie klawiszy cyfr, del i backspace

```
    /* Jezeli nacisniete zostana cyfry lub del lub backspace*/
    case(48):case(49):case(50):case(51):case(52):case(53):case(54):case(55):
    case(56):case(57):case(58):case(127):
```

Jeżeli aktywne jest pole ze stopniami funkcji jazdy do przodu/tył

```
    if(AKTYWNE==6)
{
if (znacznik==1) RPOZST=0;
if (get>47 && get <59)
    {
    RPOZST=10*RPOZST+(get-48);
    if(RPOZST>99999) RPOZST=(RPOZST-(get-48))/10;
    znacznik=0;
    }
if (get==127)
    {
    RPOZST=0;
    znacznik=0;
    }
    menu4();
    }
```

Jeżeli aktywne jest pole z minutami funkcji jazdy do przodu/tył

```
    if(AKTYWNE==7)
{
if (znacznik==1) RPOZM=0;
if (get>47 && get <59)
    {
    RPOZM=10*RPOZM+(get-48);
    if(RPOZM>99) RPOZM=(RPOZM-(get-48))/10;
    if(RPOZM>59) RPOZM=59;
    znacznik=0;
    }
if (get==127)
    {
    RPOZM=0;
    znacznik=0;
    }
    menu4();
    }
```

Jeżeli aktywne jest pole z sekundami funkcji jazdy do przodu/tył

```
    if(AKTYWNE==8)
{
if (znacznik==1) RPOZS=0;
if (get>47 && get <59)
    {
    RPOZS=10*RPOZS+(get-48);
    if(RPOZS>99) RPOZS=(RPOZS-(get-48))/10;
    if(RPOZS>59) RPOZS=59;
    znacznik=0;
    }
if (get==127)
    {
    RPOZS=0;
    znacznik=0;
    }
    menu4();
    }
```

Jeżeli aktywne jest pole ze stopniami pozycji

```
    if(AKTYWNE==12)
{
if (znacznik==1) POZST=0;
if (get>47 && get <59)
    {
    POZST=10*POZST+(get-48);
    if(POZST>99999) POZST=(POZST-(get-48))/10;
    znacznik=0;
    }
if (get==127)
    {
    POZST=0;
    znacznik=0;
    }
    menu4();
    }
```

Jeżeli aktywne jest pole z minutami pozycji

```
    if(AKTYWNE==13)
{
if (znacznik==1) POZM=0;
if (get>47 && get <59)
    {
    POZM=10*POZM+(get-48);
    if(POZM>99) POZM=(POZM-(get-48))/10;
    if(POZM>59) POZM=59;
    znacznik=0;
    }
if (get==127)
    {
    POZM=0;
    znacznik=0;
    }
    menu4();
    }
```

Jeżeli aktywne jest pole z sekundami pozycji

```
    if(AKTYWNE==14)
{
if (znacznik==1) POZS=0;
if (get>47 && get <59)
    {
    POZS=10*POZS+(get-48);
    if(POZS>99) POZS=(POZS-(get-48))/10;
    if(POZS>59) POZS=59;
    znacznik=0;
    }
if (get==127)
    {
    POZS=0;
    znacznik=0;
    }
    menu4();
    }
    break;

}
}
```

11.4 Zakończenie

11.4.1 Zablokowanie silników

```
 bzero(buf,sizeof(buf));
 mbuf.mtype=10;
 sprintf(buf," %d 100 1\n",PID);
 wyslij(buf,mqId);
```

11.4.2 Zamknięcie programu SRTA

```
 bzero(buf,sizeof(buf));
 mbuf.mtype=10;
 sprintf(buf," %d 900 1\n",PID);
```

```
 wyslij(buf,mqId);
```

11.4.3 Zamknięcie okienek (ncurses)

```
delwin(okienko3);
delwin(okienko2);
delwin(okienko1);
delwin(okienko);
refresh();
endwin();
system("clear");
}
```

Poz zakończeniu programu znajdujemy się w oknie terminala.

11.5 Funkcje pomocnicze

11.5.1 int wyslij(char buf[300], int mqId)

Funkcja wysyła i odbiera potwierdzenia otrzymanych rozkazów

```
int wyslij(char buf[300],int mqId)
{
  strcpy(mbuf.mtext,buf);
msgsnd(mqId, &mbuf, strlen(buf),0);
bzero(mbuf.mtext,sizeof(mbuf.mtext));
if(msgrcv(mqId, &mbuf, sizeof(mbuf.mtext), PID,0)==-1)
    {
    perror("blad odczytu");
    return 2;
    }
else
{
strcpy(buf,mbuf.mtext);

}
}
```

11.5.2 void menu1(void)

Funkcja inicjuje główne okienko i definiuje kolory

```
void menu1(void)
{
int ii;
/* menu glowne */
init_pair(1,COLOR_WHITE,COLOR_GREEN);
init_pair(2,COLOR_WHITE,COLOR_BLUE);
init_pair(3,COLOR_RED,COLOR_GREEN);
init_pair(4,COLOR_BLACK,COLOR_GREEN);
init_pair(5,COLOR_WHITE,COLOR_GREEN);
init_pair(6,COLOR_BLACK,COLOR_CYAN);
init_pair(7,COLOR_BLACK,COLOR_GREEN);
```

```
init_pair(8,COLOR_WHITE,COLOR_GREEN);
init_pair(9,COLOR_WHITE,COLOR_GREEN);
init_pair(10,COLOR_WHITE,COLOR_GREEN);
init_pair(11,COLOR_RED,COLOR_WHITE);
init_pair(12,COLOR_BLACK,COLOR_MAGENTA);
attron(COLOR_PAIR(2));
attrset(A_BLINK);
bkgd(COLOR_PAIR(2));
move(LINES/2,COLS/2-COLS/4);
printw(" Dopasowywanie okienka - prosze czekac");
border(ACS_VLINE,ACS_VLINE,ACS_HLINE,ACS_HLINE,ACS_ULCORNER,
       ACS_URCORNER,ACS_LLCORNER,ACS_LRCORNER);
refresh();
```

Otwieranie okienka z nagłówkiem

```
okienko3=newwin(3,COLS,0,0);
wattron(okienko3,COLOR_PAIR(2));
wattrset(okienko3,A_BLINK);
wbkgd(okienko3,COLOR_PAIR(2));
wborder(okienko3,ACS_VLINE,ACS_VLINE,ACS_HLINE,ACS_HLINE,ACS_ULCORNER,
                 ACS_URCORNER,ACS_LLCORNER,ACS_LRCORNER);
wmove(okienko3,1,COLS/2-21);
wprintw(okienko3,"Program serwisowy do silnika kata godzinnego");
wrefresh(okienko3);
```

Otwieranie okienka z odczytanymi parametrami

```
okienko1=newwin(6,COLS,3,0);
wattron(okienko1,COLOR_PAIR(2));
wattrset(okienko1,A_BLINK);
wbkgd(okienko1,COLOR_PAIR(2));
wborder(okienko1,ACS_VLINE,ACS_VLINE,ACS_HLINE,ACS_HLINE,ACS_ULCORNER,
                 ACS_URCORNER,ACS_LLCORNER,ACS_LRCORNER);

wattron(okienko1,COLOR_PAIR(3));
for(ii=1;ii<COLS-1;ii=ii+1)
{
wmove(okienko1,1,ii);
wprintw(okienko1," ");
}
wmove(okienko1,1,COLS/2-17);
wprintw(okienko1,"Odczyt parametrow i stan silnikow");
}
```

11.5.3 void menu2(void)

Funkcja wyświetla tekst w okienku z odczytanymi parametrami.

```
void menu2(void)
{
wattron(okienko1,COLOR_PAIR(2));
wmove(okienko1,2,1);
whline(okienko1,ACS_HLINE,COLS-2);

wmove(okienko1,3,COLS/4-12);
wprintw(okienko1,"             o   ,   ,,");
```

```
wattron(okienko1,COLOR_PAIR(KOLPOZ));
wmove(okienko1,4,COLS/4-12);
wprintw(okienko1,"POZYCJA:");

wattron(okienko1,COLOR_PAIR(2));
wmove(okienko1,4,COLS/4-3);
wprintw(okienko1,"%5.0d",APOZST);

wmove(okienko1,4,COLS/4+4);
wprintw(okienko1,"%2.2d",APOZM);

wmove(okienko1,4,COLS/4+8);
wprintw(okienko1,"%2.2d",APOZS);


wmove(okienko1,4,3*COLS/4-12);
wprintw(okienko1,"LACZNOSC: PRAWIDLOWA  ");

wrefresh(okienko1);
}
```

11.5.4 void menu3(void)

Funkcja otwiera okienko z panelem sterowania

```
void menu3(void)
{
/* Otwarcie okna dla 2 zwierciadla */
okienko2=newwin(LINES-6-2,COLS,6+2,0);
wattron(okienko2,COLOR_PAIR(2));
wattrset(okienko2,A_BLINK);
wbkgd(okienko2,COLOR_PAIR(2));
wborder(okienko2,ACS_VLINE,ACS_VLINE,ACS_HLINE,ACS_HLINE,ACS_ULCORNER,
                 ACS_URCORNER,ACS_LLCORNER,ACS_LRCORNER);
}
```

11.5.5 void menu4(void)

Funkcja wyświetla tekst w okienku panela sterowania

```
void menu4(void)
{
leaveok(okienko2,FALSE);
wattron(okienko2,COLOR_PAIR(12));
for(ii=1;ii<COLS-1;ii=ii+1)
{
wmove(okienko2,1,ii);
wprintw(okienko2," ");
}
wmove(okienko2,1,COLS/2-26);
wprintw(okienko2,"PANEL STEROWANIA           ( z - koniec programu)");

wattron(okienko2,COLOR_PAIR(2));
wmove(okienko2,2,1);
whline(okienko2,ACS_HLINE,COLS-2);

wattron(okienko2,COLOR_PAIR(2));
wmove(okienko2,5,COLS/2-26);
```

```
wprintw(okienko2,"SILNIK:");

wattron(okienko2,COLOR_PAIR(KOL[0]));
wmove(okienko2,5,COLS/2-8);
wprintw(okienko2," ZABLOKOWANY ");

wattron(okienko2,COLOR_PAIR(KOL[1]));
wmove(okienko2,5,COLS/2+14);
wprintw(okienko2," ODBLOKOWANY ");

wattron(okienko2,COLOR_PAIR(2));
wmove(okienko2,7,COLS/2-26);
wprintw(okienko2,"RUCH DOBOWY:");

wattron(okienko2,COLOR_PAIR(KOL[2]));
wmove(okienko2,7,COLS/2-8);
wprintw(okienko2," SZYBKI ");

wattron(okienko2,COLOR_PAIR(KOL[3]));
wmove(okienko2,7,COLS/2+1);
wprintw(okienko2," NORMALNY ");

wattron(okienko2,COLOR_PAIR(KOL[4]));
wmove(okienko2,7,COLS/2+13);
wprintw(okienko2," WOLNY ");

wattron(okienko2,COLOR_PAIR(KOL[5]));
wmove(okienko2,7,COLS/2+21);
wprintw(okienko2," STOP ");


wattron(okienko2,COLOR_PAIR(2));
wmove(okienko2,10,COLS/2-26);
wprintw(okienko2,"                   o   ,   ,,  ");

wattron(okienko2,COLOR_PAIR(2));
wmove(okienko2,11,COLS/2-26);
wprintw(okienko2,"PRZEJEDZ O:");

wattron(okienko2,COLOR_PAIR(KOL[6]));
wmove(okienko2,11,COLS/2-13);
wprintw(okienko2,"%5.0d",RPOZST);

wattron(okienko2,COLOR_PAIR(KOL[7]));
wmove(okienko2,11,COLS/2-6);
wprintw(okienko2,"%2.2d",RPOZM);

wattron(okienko2,COLOR_PAIR(KOL[8]));
wmove(okienko2,11,COLS/2-2);
wprintw(okienko2,"%2.2d",RPOZS);

wattron(okienko2,COLOR_PAIR(KOL[9]));
wmove(okienko2,11,COLS/2+6);
wprintw(okienko2," DO PRZODU ");

wattron(okienko2,COLOR_PAIR(KOL[10]));
wmove(okienko2,11,COLS/2+19);
wprintw(okienko2," DO TYL ");

wattron(okienko2,COLOR_PAIR(2));
wmove(okienko2,14,COLS/2-26);
wprintw(okienko2,"                          o   ,   ,,  ");
```

```
wattron(okienko2,COLOR_PAIR(KOL[11]));
wmove(okienko2,15,COLS/2-27);
wprintw(okienko2," NAJAZD NA POZYCJE: ");

wattron(okienko2,COLOR_PAIR(KOL[12]));
wmove(okienko2,15,COLS/2-6);
wprintw(okienko2,"%5.0d",POZST);

wattron(okienko2,COLOR_PAIR(KOL[13]));
wmove(okienko2,15,COLS/2+1);
wprintw(okienko2,"%2.2d",POZM);

wattron(okienko2,COLOR_PAIR(KOL[14]));
wmove(okienko2,15,COLS/2+5);
wprintw(okienko2,"%2.2d",POZS);

wattron(okienko2,COLOR_PAIR(KOL[15]));
wmove(okienko2,18,COLS/2-27);
wprintw(okienko2," ZAPAMIENTAJ  POZYCJE ");

wattron(okienko2,COLOR_PAIR(KOL[16]));
wmove(okienko2,18,COLS/2+4);
wprintw(okienko2," LOGOWANIE DO SILNIKOW ");

wrefresh(okienko2);
menu2();
}
```

11.5.6 void setflag(int sig)

Funkcja wywoływana gdy pojawi się sygnał **SIGALARM**. Funkcja dopasowuje wyświetlanie do rozmiarów terminala oraz czyta pozycje ze sterownika silników.

```
void setflag(int sig)
{
int i,j;
int aa,bb,cc;
double dd,ddold;
char buf[300];
    alarm(0);
```

Sprawdzenie czy nie zmienił się rozmiar okienka

```
    if(oldCOLS!=COLS || oldLINES!=LINES)
    {
```

Jeśli rozmiar się zmienił to zamykamy i otwieramy ponownie okienka

```
    endwin();
    nowe=initscr();
start_color();
has_colors();
oldCOLS=COLS;oldLINES=LINES;
/* kolory */
init_pair(1,COLOR_WHITE,COLOR_GREEN);
init_pair(2,COLOR_WHITE,COLOR_BLUE);
init_pair(3,COLOR_RED,COLOR_GREEN);
init_pair(4,COLOR_BLACK,COLOR_GREEN);
```

```
init_pair(5,COLOR_WHITE,COLOR_GREEN);
init_pair(6,COLOR_RED,COLOR_GREEN);
init_pair(7,COLOR_BLACK,COLOR_GREEN);
init_pair(8,COLOR_WHITE,COLOR_GREEN);
init_pair(9,COLOR_WHITE,COLOR_GREEN);
init_pair(10,COLOR_WHITE,COLOR_GREEN);
init_pair(11,COLOR_RED,COLOR_WHITE);

attron(COLOR_PAIR(2));

for(i=1;i<LINES;i=i+1)
{
for(j=1;j<COLS;j=j+1)
{
move(i,j);
printw(" ");
}
}
move(LINES/2,COLS/2-8);
printw("Za male okienko");
refresh();
```

Jeżeli okienko jest wystarczające duże to (jeśli nie to na ekranie pozostaje napis „Za małe okienko”)

```
if(COLS>=71)
if(LINES>=29)
{
move(LINES/2,COLS/2-8);
printw("               ");

menu1();
menu2();
menu3();
menu4();

}
    }
```

Wysłanie zapytania do sterownika silnika o pozycje

```
sprintf(buf," %d 200 0\n",PID);
strcpy(mbuf.mtext,buf);
msgsnd(ID, &mbuf, strlen(buf),0);
```

Odbiór potwierdzenia

```
bzero(mbuf.mtext,sizeof(mbuf.mtext));
if(msgrcv(ID, &mbuf, sizeof(mbuf.mtext), PID,0)==-1)
    {
    perror("blad odczytu");
    }
else
{
strcpy(buf,mbuf.mtext);
aa=0;bb=0;cc=0;dd=0;
```

i rozczytanie komunikatu

```
sscanf(buf,"%d %d %d %lf\n", &aa, &bb, &cc, &dd);
APOZSTNEW=0;APOZMNEW=0;APOZSNEW=0;
```

W przypadku gdy zmiana pozycji była większa niż 5 sekun łuku zmieniamy kolor napisu POZYCJA

```
if((ddold-dd)*(ddold-dd)>5*5)
{
KOLPOZ=11;
}
else
{
KOLPOZ=2;
}
          ddold=dd;
```

Wyznaczamy nowe wartości pozycji w stopniach

```
while(dd>3600)
  {
  dd=dd-3600;
  APOZSTNEW=APOZSTNEW+1;
  }
while(dd>60)
  {
  dd=dd-60;
  APOZMNEW=APOZMNEW+1;
  }
while(dd>1)
{
dd=dd-1;
APOZSNEW=APOZSNEW+1;
}
```

Jeżeli pozycja się zmieniła to zapamiętujemy nową.Sprawdzamy czy to pierwsze czytanie pozycji z silnika po uruchomieniu. Jeśli tak to ustawiamy te same wartości zapamiętanej pozycji do najazdu.

```
if(APOZSTNEW==APOZST && APOZMNEW==APOZM && APOZSNEW==APOZS)
{

}
else
{
APOZST=APOZSTNEW;
APOZM=APOZMNEW;
APOZS=APOZSNEW;
}
if(PIERWSZE==1)
    {
    POZST=APOZST;
    POZM=APOZM;
    POZS=APOZS;
    PIERWSZE=0;
    }
menu2();
menu3();
}

}
```

11.5.7 void start(void)

Funkcja inicjuje ekran przy pomocy biblioteki ncurses

```
void start(void)
{
int i,j;
    nowe=initscr();
    alarm(0);
start_color();
has_colors();
curs_set(0);
oldCOLS=COLS;oldLINES=LINES;
init_pair(1,COLOR_WHITE,COLOR_GREEN);
init_pair(2,COLOR_WHITE,COLOR_BLUE);
init_pair(3,COLOR_RED,COLOR_GREEN);
init_pair(4,COLOR_BLACK,COLOR_GREEN);
init_pair(5,COLOR_WHITE,COLOR_GREEN);
init_pair(6,COLOR_RED,COLOR_GREEN);
init_pair(7,COLOR_BLACK,COLOR_GREEN);
init_pair(8,COLOR_WHITE,COLOR_GREEN);
init_pair(9,COLOR_WHITE,COLOR_GREEN);
init_pair(10,COLOR_WHITE,COLOR_GREEN);
init_pair(11,COLOR_RED,COLOR_WHITE);

attron(COLOR_PAIR(2));

for(i=1;i<LINES;i=i+1)
{
for(j=1;j<COLS;j=j+1)
{
move(i,j);
printw(" ");
}
}
move(LINES/2,COLS/2-8);
printw("Za male okienko");
refresh();
if(COLS>=71 && LINES>=29)
{
move(LINES/2,COLS/2-8);
printw("               ");
menu1();
menu2();
menu3();
menu4();

}
else
{
endwin();
exit();
}

}
```