Sterowniki urządzeń w systemie operacyjnym RTEMS

Adam Piotrowski

2 kwietnia 2007

1 Informacje podstawowe

Jednym z podstawowych zadań systemu operacyjnego jest zapewnienie niezawodnej i wydajnej współpracy z urządzeniami zewnętrznymi. Wobec mnogości rozwiązań sprzętowych, system nie jest w stanie dostarczyć oprogramowania do komunikacji z wszystkimi możliwymi typami urządzeń zewnętrznych. W latach 80 w systemie operacyjnym MS DOS i jego klonach zadanie to spoczywało na autorach konkretnego oprogramowania - każdy program korzystający z karty dźwiękowej lub zaawansowanych opcji karty graficznej musiał mieć własne sterowniki obsługujące konkretne modele urządzeń. Sytuacja uległa zmianie w momencie pojawienia się systemów operacyjnych Windows (NT lub 95), w przypadku których odpowiedzialność za dostarczanie sterowników urządzeń spoczywa na projektantach danego urządzenia. System operacyjny zapewnia tylko odpowiedni interfejs programistyczny, umożliwiający prosty i zunifikowany dostęp do każdego typu urządzeń.

2 Sterowniki urządzeń

Sterowniki są bibliotekami umożliwiającymi komunikację z urządzeniami zewnętrznymi. W przeciwieństwie do standardowych aplikacji program sterownika nie jest ciągle wykonywany lecz jego poszczególne procedury są wywoływane w odpowiedzi na żądania systemu operacyjnego. Każdy sterownik musi zostać odpowiednio zainicjalizowany, tak aby system operacyjny mógł określić w jaki sposób przeprowadzić komunikację z danym urządzeniem. System operacyjny dostarcza odpowiedni interfejs programistyczny umożliwiający przypisanie konkretnych procedur dostarczanych przez sterownik do odpowiednich wywołań standardowych funkcji API.

3 Komunikacja z urządzeniami I/O w systemie RTEMS

W systemie RTEMS przypisanie funkcji API do poszczególnych procedur sterownika realizowane jest na etapie kompilacji projektu i polega na samodzielnym wypełnieniu struktury typu *rtems_driver_address_table* o nazwie *Device_drivers* oraz zdefiniowaniu stałej *CONFIGURE_HAS_OWN_DEVICE_DRIVER_TABLE*. Należy jednak pamiętać że taka deklaracja wymusza na programiście samodzielne dopisanie do wyżej wymienionej zmiennej identyfikatorów wszystkich sterowników które mają być dostępne w programie. Wszystkie definicje postaci *CONFIGURE_APPLICATION_NEEDS_CONSOLE_DRIVER* są unieważnione. Przykład inicjalizacji tablicy *Device_drivers* znajduje się na rysunku 2.

Typ *rtems_driver_address_table* to struktura wskaźników do funkcji dostarczanych przez sterownik w celu zapewnienia określonej funkcjonalności. Przypisania poszczególnych składników struktury do kolejnych funkcji API przedstawione jest w tabeli 1.

```
typedef rtems_device_driver ( *rtems_device_driver_entry )(
                rtems_device_major_number,
                rtems_device_minor_number,
                void *
        );

typedef struct {
  rtems_device_driver_entry initialization_entry; /* initialization procedure */
  rtems_device_driver_entry open_entry;        /* open request procedure */
  rtems_device_driver_entry close_entry;       /* close request procedure */
  rtems_device_driver_entry read_entry;        /* read request procedure */
  rtems_device_driver_entry write_entry;       /* write request procedure */
  rtems_device_driver_entry control_entry;     /* special functions procedure */
}   rtems_driver_address_table;
```

Rysunek 1: Deklaracja struktury *rtems_driver_address_table*

```
#define DRV_DRIVER_TABLE_ENTRY \
               {drv_init, drv_open, drv_close, drv_read, drv_write, drv_ioctrl}

#ifdef CONFIGURE_INIT
rtems_driver_address_table  Device_drivers[2] =
{
               CONSOLE_DRIVER_TABLE_ENTRY,
               DRV_DRIVER_TABLE_ENTRY
};
#endif
```

Rysunek 2: Inicjalizacja struktury *Device_drivers*

Funkcja API POSIX	*Funkcja API RTEMS*	*Wskaźnik do funkcji*
brak, brak	rtems_io_initialize	initialization_entry
open, fopen	rtems_io_open	open_entry
read, fread	rtems_io_read	read_entry
write, fwrite	rtems_io_write	write_entry
close, fclose	rtems_io_close	close_entry
ioctl, brak	rtems_io_control	control_entry

Tabela 1: Przypisania standardowych funkcji API

```
typedef struct {
    rtems_libio_t          *iop;
    uint32_t                flags;
    uint32_t                mode;
} rtems_libio_open_close_args_t;
```

Rysunek 3: Deklaracja struktury *rtems_libio_open_close_args_t*

3.1 Inicjalizacja urządzenia - initialization_entry

*rtems_device_driver drv_init(rtems_device_major_number major, rtems_device_minor_number minor, void * context);*

Pierwszy z definiowanych w strukturze *rtems_driver_address_table* wskaźników do funkcji dotyczy funkcji wywoływanej automatycznie przez system operacyjny na etapie startu. Umożliwia ona przeprowadzenie konfiguracji urządzenia oraz wszystkich zmiennych sterownika. Pierwszy i drugi parametr funkcji identyfikuje urządzenie do którego jest skierowane polecenie. Trzeci parametr funkcji nie jest wykorzystywany. Ważnym zadaniem tej funkcji jest przypisanie nazwy urządzenia do numerów identyfikujących urządzenie. Zadanie to realizowane jest za pomocą funkcji rtems_io _register_name.

3.2 Operacja otwierania urządzenia - open_entry

*rtems_device_driver drv_open(rtems_device_major_number major, rtems_device_minor_number minor, void * context);*

Wskaźnik do funkcji odpowiedzialnej za obsługę poleceń *open, fopen* oraz *rtems_io_open*. Funkcja odpowiedzialna jest za inicjalizację sterownika w momencie próby otworzenia dostępu do urządzenia, umożliwia między innymi kontrolę poprawności praw dostępu do urządzenia. Trzeci parametr funkcji jest wskaźnikiem do struktury *rtems_libio_open_close_args_t* której deklaracja jest przedstawiona na rysunku 3. Opis znaczenia poszczególnych pól struktury znajduje się w tabeli 2.

Nazwa pola	*Opis*
rtems_libio_t *iop;	Wskaźnik do struktury identyfikującej otworzone urządzenie
uint32_t flags;	Określa tryb otwierania urządzenia
uint32_t mode;	Określa prawa dostępu do otwieranego urządzenia

Tabela 2: Elementy struktury *rtems_libio_open_close_args_t*

```
typedef struct {
      rtems_libio_t          *iop;
      off_t                   offset;
      char                   *buffer;
      uint32_t                count;
      uint32_t                flags;
      uint32_t                bytes_moved;
} rtems_libio_rw_args_t;
```

Rysunek 4: Deklaracja struktury *rtems_libio_rw_args_t*

Nazwa pola	*Opis*
rtems_libio_t *iop;	Wskaźnik do struktury identyfikującej zapisywan/odczytywane urządzenie
off_t offset;	Początkowe przesunięcie odczytywanych/zapisywanych danych
char *buffer;	Wskaźnik do bufora zawierającego dane do zapisu/miejsce na odczytane dane
uint32_t count;	Rozmiar bufora
uint32_t flags;	
uint32_t bytes_moved;	Ilość odczytanych/zapisanych danych

Tabela 3: Elementy struktury *rtems_libio_rw_args_t*

3.3 Operacja zamykania urządzenia - close_entry

*rtems_device_driver drv_close(rtems_device_major_number major, rtems_device_minor_number minor, void * context);*

Wskaźnik do funkcji odpowiedzialnej za obsługę poleceń *close, fclose* oraz *rtems_io_close*. Funkcja odpowiedzialna jest za zamknięcie dostępu do urządzenia. Trzeci parametr funkcji jest wskaźnikiem do struktury *rtems_libio_open_close_args_t* której deklaracja jest przedstawiona na rysunku 3. Opis znaczenia poszczególnych pól struktury znajduje się w tabeli 2.

3.4 Operacja zapisu danych do urządzenia - write_entry

*rtems_device_driver drv_write(rtems_device_major_number major, rtems_device_minor_number minor, void * context);*

Wskaźnik do funkcji odpowiedzialnej za obsługę poleceń *rwrite, fwrite* oraz *rtems_io_write*. Funkcja odpowiedzialna jest za zapis danych przekazywanych przez użytkownika do urządzenia. Trzeci parametr funkcji jest wskaźnikiem do struktury *rtems_libio_rw_args_t* której deklaracja jest przedstawiona na rysunku 4. Opis znaczenia poszczególnych pól struktury znajduje się w tabeli 3.

3.5 Operacja odczytu danych z urządzenia - read_entry

*rtems_device_driver drv_read(rtems_device_major_number major, rtems_device_minor_number minor, void * context);*

Wskaźnik do funkcji odpowiedzialnej za obsługę poleceń *read, fread* oraz *rtems_io_read*. Funkcja odpowiedzialna jest za realizację operacji odczytu danych z urządzenia. Trzeci parametr funkcji jest wskaźnikiem do struktury *rtems_libio_rw_args_t* której deklaracja jest przedstawiona na rysunku 4. Opis znaczenia poszczególnych pól struktury znajduje się w tabeli 3.

```
typedef struct {
    rtems_libio_t *iop;
    uint32_t command;
    void *buffer;
    uint32_t ioctl_return;
} rtems_libio_ioctl_args_t;
```

Rysunek 5: Deklaracja struktury *rtems_libio_ioctl_args_t*

3.6 Rozkazy IOCTL - control_entry

*rtems_device_driver drv_ioctl(rtems_device_major_number major, rtems_device_minor_number minor, void * context);*

Wskaźnik do funkcji realizującej obsługę poleceń wykonywanych za pomocą rozkazów *ioctl* oraz *rtems_io_control*. Rozkazy te przeznaczone są do wysyłania poleceń sterujących definiowanych przez programistę. Trzeci parametr funkcji jest wskaźnikiem do struktury *rtems_libio_ioctl_args_t* której deklaracja jest przedstawiona na rysunku 5. Opis znaczenia poszczególnych pól struktury znajduje się w tabeli 4.

Identyfikator wykonywanego polecenia jest liczbą 32 bitową składającą się z czterech części określających poszczególne właściwości danego polecenia:

- Numer magiczny - ośmiobitowy numer wybierany przez programistę
- Kolejny numer ośmiobitowy
- Kierunek przesyłania danych - możliwe wartości to:
  - _IOC_NONE - brak przesyłania danych (makro _IO(magiczny_numer, numer))
  - _IOC_READ - odczyt danych z urządzenia (makro _IOR(magiczny_numer, numer, ilosc_danych))
  - _IOC_WRITE - zapis danych do urządzenia (makro _IOW(magiczny_numer, numer, ilosc_danych))
  - _IOC_WRITE | _IOC_READ - zapis i odczyt danych z/do urządzenia (makro _IOWR(magiczny_numer, numer, ilosc_danych))
- Ilość przesyłanych danych - pole opcjonalne

Przykład kodu definującego nowe polecenia pokazany jest poniżej.

```
#define DEV_IO_MAGIC                    0x07
#define IOCTRL_DEV_NO_DATA              _IO(DEV_IO_MAGIC,0x01)
#define IOCTRL_DEV_ONLY_WRITE           _IOW(DEV_IO_MAGIC, 0x02, sizeof(int))
#define IOCTRL_DEV_ONLY_READ            _IOR(DEV_IO_MAGIC,0x03, sizeof(int)*5)
#define IOCTRL_DEV_RW                   _IOWR(DEV_IO_MAGIC,0x04, sizeof(int)*10)
```

4 Podstawowe struktury danych

Definicje wszystkich struktur danych wykorzystywanych podczas komunikacji pomiędzy systemem operacyjnym a sterownikami urządzeń znajdują się w plikach:

- ${RTEMS_SRC}/cpukit/libcsupport/include/rtems/libio.h

Nazwa pola	*Opis*
rtems_libio_t *iop;	Wskaźnik do struktury identyfikującej urządzenie
uint32_t command;	Identyfikator polecenia do wykonania
void *buffer;	Wskaźnik do bufora zawierającego dane - opcjonalne
uint32_t ioctl_return;	Wartość zwracana z funkcji określająca status zakończenia operacji

Tabela 4: Elementy struktury *rtems_libio_ioctl_args_t*

- ${RTEMS_SRC}/cpukit/sapi/include/rtems/io.h
- ${RTEMS_SRC}/cpukit/libcsupport/src/*.*

gdzie ${RTEMS_SRC} określa katalog w którym znajdują się kody źródłowe systemu operacyjnego.

5 Komunikacja z sterownikami urządzeń w systemie RTEMS

Z punktu widzenia programisty aplikacji działającej pod kontrolą systemu RTEMS, komunikacja z urządzeniami I/O może być realizowana za pomocą natywnego interfejsu wejścia wyjścia dostarczonego z API systemu operacyjnego, lub też za pomocą funkcji zdefiniowanych w standardzie POSIX. Przykład programu wykorzystującego polecenia POSIX API do komunikacji z sterownikiem przedstawiony jest poniżej.

```
rtems_signed32 od;
FILE* old_stdout = stdout;
rtems_unsigned8    move = 12;
rtems_status_code  status;
od = open("/dev/lcd",O_WRONLY , 0);
FILE* fd = fopen("/dev/lcd", "a");

if (!fd) {
          printf("Cannot open device /dev/lcd\n\r"); fflush(old_stdout);
          rtems_task_suspend(RTEMS_SELF);
}
stdout = fd;
if ((status = ioctl(od, IOCTRL_LCD_CLR)) != RTEMS_SUCCESSFUL){
          fprintf(old_stdout, "%s\n", rtems_status_text(status));
          fflush(old_stdout);
          rtems_task_suspend(RTEMS_SELF);
}
if ((status = ioctl(od, IOCTRL_LCD_CURSOR_ON)) != RTEMS_SUCCESSFUL){
          fprintf(old_stdout, "%s\n", rtems_status_text(status));
          fflush(old_stdout);
          rtems_task_suspend(RTEMS_SELF);
}
printf("HELLO_1"); fflush(stdout);
if ((status = ioctl(od, IOCTRL_LCD_MOVE_CURSOR, &move)) != RTEMS_SUCCESSFUL){
          fprintf(old_stdout, "%s\n", rtems_status_text(status));
          fflush(old_stdout);
          rtems_task_suspend(RTEMS_SELF);
}
```

```
printf("HELLO_2"); fflush(stdout);
if ((status = ioctl(od, IOCTRL_LCD_CLR+7)) != RTEMS_SUCCESSFUL){
         fprintf(old_stdout, "%s\n", rtems_status_text(status));
         fflush(old_stdout);
}
fclose(fd);
close(od);
rtems_task_suspend(RTEMS_SELF);
```