Rozdział 1

Dyscyplina linii

Autor: Sebastian Zagrodzki
Data: 20. grudnia 2000

Ten rozdział opisuje mechanizm obsługi terminali oraz sposób działania dyscypliny linii razem z niektórymi implementacjami interfejsu dyscypliny.

1.1 Terminale

Terminale są urządzeniami znakowymi, których przeznaczeniem jest komunikacja między systemem a użytkownikiem, który pracuje bezpośrednio na danej maszynie (z wykorzystaniem monitora i klawiatury jako urządzeń we/wy), lub np. używa terminala tekstowego[1] podłączonego po łączu szeregowym.

Z terminalem, jak z każdym urządzeniem znakowym w systemie, związane są podstawowe operacje wysokopoziomowe [2] zdefiniowane w strukturze `file_operations`. Z tych funkcji korzystają programy użytkowe operujące na terminalach. Po przeciwnej stronie znajdują się niskopoziomowe [3] mechanizmy obsługi sprzętu takiego jak klawiatura lub port szeregowy.

Terminale od innych urządzeń wyróżniają dodatkowe struktury definiujące parametry pracy (prędkość transmisji, sposób kontroli przepływu danych, sposób wstępnej obróbki przesyłanych znaków, itp.) — jest to przede wszystkim struktura `tty_struct`, łącząca terminal z odpowiednim sterownikiem niskiego poziomu oraz z właściwą **dyscypliną linii**, dodatkowo zawierająca wskaźniki do innych struktur jak np. `termios`, opisujących tryb pracy terminala.

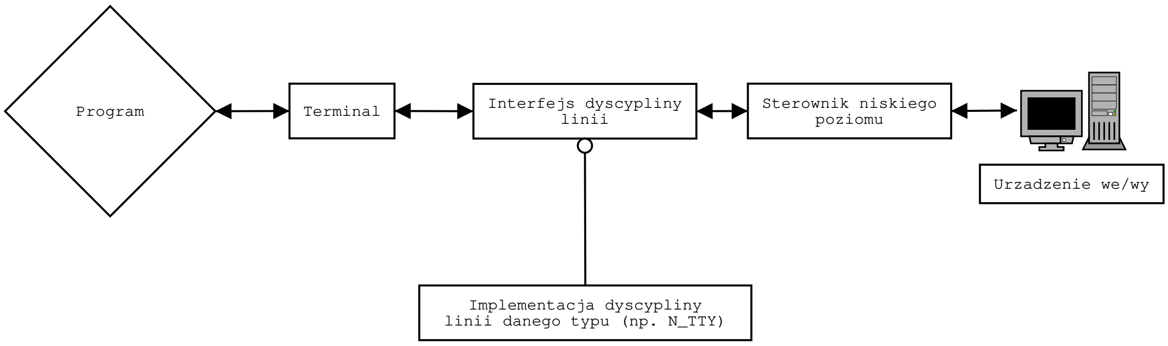
[1] Terminal tekstowy to urządzenie na pierwszy rzut oka podobne do zwykłego komputera jak np. PC, ale nie dysponujące własną mocą obliczeniową, potrafiące tylko komunikować się przez kabel szeregowy

[2] pisząc „wysokopoziomowe" mam na myśli operacje występujące w warstwie obsługi terminala będącej najbliżej procesu użytkownika

[3] czyli występujące w układzie warstwowym najbliżej fizycznego sprzętu, zajmujące się takimi rzeczami jak przerwania, porty itp.

1.2 Dyscyplina linii

Dyscyplinę linii można traktować jako filtr między fizycznym urządzeniem (i jego sterownikiem) a funkcjami zdefiniowanymi w `file_operations`. To ona jest odpowiedzialna np. za przesłanie sygnału `SIGINT` do procesu, który korzysta z terminala, w momencie kiedy użytkownik wciśnie kombinację klawiszy `Ctrl-C`.



Rysunek 1.1: Schemat komunikacji między poszczególnymi warstwami obsługi terminala

1.2.1 Interfejs dyscypliny

W Linuksie jest zdefiniowany ogólny interfejs dyscypliny, zdefiniowany przez strukturę `tty_ldisc`. Struktura ta zawiera wskaźniki do następnujących funkcji:

- `int (*open)(struct tty_struct *)`
  Ta funkcja jest wywoływana w momencie przypisania danej dyscypliny linii do terminala (funkcją `tty_set_ldisc`) — jest to inicjacja dyscypliny.

- `void (*close)(struct tty_struct *)`
  Funkcja wywoływana w momencie odłączenia dyscypliny od terminala w momencie, kiedy terminal jest zamykany lub gdy następuje zmiana używanej dyscypliny.

- `void (*flush_buffer)(struct tty_struct *tty)`
  Poleca dyscyplinie wyczyścić kolejkę znaków przeznaczonych do przesłania do procesu.

- `ssize_t (*chars_in_buffer)(struct tty_struct *tty)`
  Zwraca liczbę znaków w buforze dyscypliny.

- `ssize_t (*read)(struct tty_struct * tty, struct file * file, unsigned char * buf, size_t nr)`
  Ta funkcja jest wołana w momencie kiedy proces próbuje coś odczytać z terminala. Dyscyplina zwraca w tym momencie znaki, które zdążyła zgromadzić w kolejce do wysłania.

- `ssize_t (*write)(struct tty_struct * tty, struct file * file, const unsigned char * buf, size_t nr)`

Funkcja wywoływana w chwili kiedy proces próbuje coś zapisać (wykonać operację `write()` na terminalu). Dyscyplina dostarcza otrzymane znaki do fizycznego urządzenia przez niskopoziomowy sterownik.

- `int (*ioctl)(struct tty_struct * tty, struct file * file, unsigned int cmd, unsigned long arg)`
  `ioctl` jest wywoływane w momencie, kiedy użytkownik wykona polecenie `ioctl` na terminalu, ale nie zostanie ono obsłużone ani przez warstwę górną obsługi terminala, ani przez niskopoziomowy sterownik urządzenia (w tej kolejności).

- `void (*set_termios)(struct tty_struct *tty, struct termios * old)`
  Powiadamia dyscyplinę o zmianie trybu pracy definiowanego przez strukurę `termios`.

- `int (*poll)(struct tty_struct * tty, struct file * file, poll_table *wait)`
  Ta funkcja jest wywoływana przy wykonaniu przez proces operacji wybierania/odpytywania (ang. *select/poll*).

- `void (*receive_buf)(struct tty_struct *, const unsigned char *cp, char *fp, int count)`
  Funkcja wywoływana przez sterownik niskopoziomowy w momencie, kiedy ma on znaki do przekazania do procesu użytkownika. Dyscyplina kolejkuje te znaki, i po ich wstępnym przetworzeniu przekazuje dalej.

- `int (*receive_room)(struct tty_struct *)`
  Zwraca liczbę znaków, które dyscyplina może jeszcze otrzymać. W momencie kiedy urządzenie będzie wysyłać więcej znaków niż zwróci ta funkcja — mogą one zostać zgubione (nie zmieszczą się w kolejce)

- `void (*write_wakeup)(struct tty_struct *)`
  Sygnalizuje dyscyplinie, że może dostarczyć znaki które ma do przekazania do urządzenia. Jeśli nie ma żadnych, to funkcja nie ma żadnego efektu.

Ostatnie trzy funkcje z tego zestawu są wywoływane przez sterownik niskopoziomowy, który np. dostał właśnie od urządzenia znaki, lub skończył je wysyłać.

1.2.2 Implementacje dyscyplin w kernelu Linuksa

W plikach nagłówkowych kernela (konkretnie w pliku `/usr/include/asm/termios.h`) zostało zdefiniowane 13 różnych rodzajów dyscyplin.

```
#define N_TTY          0
#define N_SLIP         1
#define N_MOUSE        2
#define N_PPP          3
#define N_STRIP        4
#define N_AX25         5
#define N_X25          6       /* X.25 async */
#define N_6PACK        7
```

```
#define N_MASC          8       /* Reserved for Mobitex module <kaz@cafe.net> */
#define N_R3964         9       /* Reserved for Simatic R3964 module */
#define N_PROFIBUS_FDL  10      /* Reserved for Profibus <Dave@mvhi.com> */
#define N_IRDA          11      /* Linux IR - http://www.cs.uit.no/~dagb/irda/irda.html */
#define N_SMSBLOCK      12      /* SMS block mode - for talking to GSM data cards
                                   about SMS messages */
#define N_HDLC          13      /* synchronous HDLC */
```

Co ciekawe, jedynie dwa z tych typów są zaimplementowane: są to `N_TTY` oraz `N_HDLC`.

1.2.3 Implementacja dyscypliny `N_TTY`

Najważniejszym typem dyscypliny jest `N_TTY`. Jest ona używana domyślnie, dodatkowo w wielu miejscach w kodzie jądra są instrukcje, które działają bazując na tym, że `N_TTY` jest dyscypliną o najmniejszym numerze identyfkacyjnym(0).

Schemat implementacji niektórych funkcji z interfejsu dyscypliny można znaleźć w ramkach.

Algorytm 1 open()

Dane wejściowe: wskaźnik do `tty_struct`
Wynik: 0, `-EINVAL` lub `-ENOMEM`

jeśli wskaźnik jest `NULL` **to**
    wyrzuć błąd `-EINVAL`
koniec jeśli
jeśli ten terminal nie ma bufora do czytania (`tty->read_buf`) **to**
    zaalokuj stronę i ustaw wskaźnik, ew. zwróć błąd `-ENOMEM`
koniec jeśli
ustaw domyślne wartości
zwróć 0

Algorytm 2 close()

Dane wejściowe: wskaźnik do `tty_struct`
Wynik: 0

`flush_buffer()`
jeśli jest bufor do czytania (`tty->read_buf`) **to**
    zwolnij stronę i ustaw `NULL`
koniec jeśli
ustaw domyślne wartości
zwróć 0

1.3 Zadania

1. Zmienić obsługę dyscypliny `N_TTY` tak, aby po naciśnięciu kombinacji klawiszy `Ctrl-C` i wpisaniu dwucyfrowego numeru sygnału ten sygnał był wysyłany do procesu.

Algorytm 3 flush_buffer()

Dane wejściowe: wskaźnik do `tty_struct`

```
wyczyść wszystkie flagi
jeśli terminal ma drugi terminal podrzędny (jest pseudoterminalem) i pracuje w trybie pakietowym to
    dokończ wszystkie operacje odczytu/zapisu
koniec jeśli
```

Algorytm 4 read()

Dane wejściowe: wskaźnik do `tty_struct`, pliku do czytania, bufora do którego należy wstawić wczytane znaki; ilość znaków

```
sprawdź czy możesz działać (ang. job control — nie dotyczy konsoli systemowej)
jeśli nie ma ustawionego trybu kanonicznego (przesyłanie całych linii) to
    oczekuj (aktywnie) na możliwość wysłania znaku
koniec jeśli
jeśli otwieramy w trybie nieblokującym to
    spróbuj otworzyć urządzenie, jeśli się nie uda to wyjdź
w przeciwnym przypadku
    przejdź w stan oczekiwania (interruptible)
koniec jeśli
dopóki są znaki wykonuj
    jeśli terminal pracuje w trybie pakietowym i poprzedni znak został wysłany to
        zmień status
    w przeciwnym przypadku
        jeśli terminal zginął to
            wyjdź
        koniec jeśli
        jeśli terminal pracuje w trybie kanonicznym to
            jeśli jest dość miejsca w buforze do przesłania znaków to
                wyślij znak
            w przeciwnym przypadku
                oczekuj
            koniec jeśli
        w przeciwnym przypadku
            wyslij znak
        koniec jeśli
    koniec jeśli
    jeśli zwolniło się wystarczająco dużo miejsca to
        odetkaj (ang. unthrottle) urządzenie
    koniec jeśli
koniec dopóki
```

Algorytm 5 write()

Dane wejściowe: wskaźnik do `tty_struct`, pliku do czytania, bufora do którego należy wstawić wczytane znaki; ilość znaków

Wynik: ilość zapisanych znaków

```
job control - jak we write()
dopóki są znaki wykonuj
  jeśli to jest przetwarzanie końcowe (flaga O_OPOST) to
    wyślij wszystkie znaki
    wyskocz
  w przeciwnym przypadku
    wyslij jeden znak
  koniec jeśli
koniec dopóki
```

Algorytm 6 receive_buf()

Dane wejściowe: wskaźnik do `tty_struct`, bufora znaków, flag; ilość znaków

```
jeśli terminal pracuje w trybie surowym to
  prześlij wszystko co się da
w przeciwnym przypadku
  dopóki są znaki wykonuj
    dla każdego znaku sprawdź jego typ
    jeśli normalny znak to
      wyślij
    koniec jeśli
    jeśli znak BREAK to
      wyślij sygnał SIGINT (przez funkcję receive_break())
    koniec jeśli
    jeśli błąd parzystości, ramki lub nadpisanie bufora (na liniach szeregowych) to
      wywolaj procedurę obsługi błędu
    koniec jeśli
  koniec dopóki
koniec jeśli
```

1.4 Bibliografia

- źródła kernela Linuksa w wersji 2.2.18
- programy obsługujące terminale (getty, miniterm)
- materiały dot. SO z lat poprzednich