Konrad Gotfryd

STER

Projekt II: Wyświetlacz LCD z panelem dotykowym na platformę Raspberry Pi.

1. Opis platformy.

W projekcie zastosowano platformę Raspberry Pi model B oraz wyświetlacz LCD TFT 2.8” z interfejsem SPI i linią DATA/CMD, dodatkowo wyposażony w dotykowy panel rezystancyjny.

2. Przygotowanie platformy do projektu.

Celem uruchomienia własnych sterowników należało:

- Skompilować jądro systemu Linux.
- Zmienić wpisy w drzewie urządzeń (device tree). W tym celu został zmieniony plik bcm2708-rpi-b.dts. Do dokumentacji został dołączony plik po zmianach. Plik ten został skompilowany (wywołanie make ARCH=arm bcm2708-rpi-b.dtb) i dodany do katalogu /boot/.
- Dodać reguły do pliku udev celem zmiany praw dostępu do pliku (plik załączony do dokumentacji).

3. Założenia projektowe.

- Obsługa interfejsu przez sterowniki jądra.
- Daemon w przestrzeni użytkownika dla realizacji algorytmów.

4. Wykonanie.

Ładowanie sterownika następuje automatycznie przez wpis do device tree nazw sterowników oraz przez program modprobe.

Komunikacja ze sterownikiem touchscreen_spi przebiega prawie identycznie jak z lcd_spi, tylko bez obsługi linii data/cmd.

Daemon komunikuje się ze sterownikiem lcd_spi przez ioctl, wysyłając najpierw strukturę zawierającą pola:

```
struct lcdd_transfer {
        uint32_t byte_cnt;
        const uint8_t *tx_buf;
        uint8_t *rx_buf;
}
```

byte_cnt – liczba bajtów do skopiowania z przestrzeni użytkownika.
tx_buf i rx_buf – wskaźniki na bufory z przestrzeni użytkownika.

Zgodnie z tym kopiowana jest określona liczba bajtów z tx_buf, a następnie

wysyłane są w tej samej kolejności bajty przez interfejs SPI. Od rodzaju komendy ioctl zależy stan linii data/cmd.

W przestrzeni użytkownika klient komunikuje się z daemonem poprzez gniazda sieciowe. Wraz z pojawieniem się nowego połączenia, w daemonie tworzy się nowe gniazdo, które obsługuje klienta, a akceptacja kolejnych połączeń jest wstrzymana, przez co po stronie użytkownika został rozwiązany problem dostępu danych i synchronizacji. Realizacja tego po stronie jądra mogłaby wyglądać następująco:

dopuszczalne byłoby otworzenie przez wiele procesów pliku urządzenia, a bufory przydzielane byłyby do struktury skojarzonej z deskryptorem pliku. Wtedy należy zastosować mutex obejmujący całą funkcję ioctl, ponieważ w przypadku programu wielowątkowego mogłoby dojść do pracy na tych samych buforach tx i rx po stronie kernela. Mutex ten tworzony byłby razem z buforami tx i rx przy otwieraniu pliku urządzenia. Nie ma konieczności wprowadzania globalnego semafora, ponieważ funkcja spi_async zawiera rygiel pętlowy, który kontroluję magistralę spi: (http://lxr.free-electrons.com/source/drivers/spi/spi.c#L2256).

Nie ma konieczności oczekiwania między między ustawieniem stanu pinu data/cmd a realizacją transferu spi, stąd udelay(1) w funkcji lcdd_message_send został zakomentowany.

Sterownik panelu dotykowego reaguje na dotyk poprzez wykrycie stanu na linii IRQ modułu panelu. Linia zmienia swój stan na negatywny przy dotyku. Powiadamianie modułu wyświetlacza odbywa się przez notification chain. Opcja ta jest wyłączana po 1 powiadomieniu, by niepotrzebnie nie generować przerwań w module wyświetlacza. Dobrano częstotliwość wznawiania powiadomień raz na sekundę.

5. Wnioski.

- Tworzenie sterownika bez trudnych algorytmów w przestrzeni jądra jest prostsze i jeśli to jest możliwe tak powinno się tworzyć, ponieważ czas poświęcony na debuggowanie jest znacznie mniejszy i kod łatwiej kontrolować.
- Nie ma potrzeby stosować opóźnień przy zmianach stanów na pinie przy tej parze urządzeń.
- Celem dopuszczenia pracy wielowątkowej należałoby zastosować mutex w funkcji ioctl.