Jan Kopański

Wersja bez użycia bloku DMA, operacje asynchroniczne.

Struktury danych:

```
struct monter_dev {
   struct pci_dev *pdev;
   struct cdev cdev;
   void __iomem *bar0;
   struct monter_context *current_context;
   struct list_head cmd_queue;
   struct list_head device_list;
   struct mutex write_mutex;
   spinlock_t slock;
};
```

Struktura reprezentująca pojedyncze urządzenie fizyczne. Pole current_context jest wskaźnikiem na kontekst urządzenie, w którym aktualnie wykonuje operacje aktualnie, cmd_queue jest początkiem listy bloków komend, które będą przesłane do kolejki urządzenia. Dzięki device_list urządzenia są spięte w listę. Natomiast write_mutex służy do synchronizacji wewnątrz funkcji monter_write, a slock to blokada wirująca wykorzystywana przy dostępie do listy cmd_queue w funkcjach monter_write, monter_release i funkcji obsługi przerwań monter_irq_handler.

```
struct monter_context {
   struct monter_dev *mdev;
   void *kern_pages[MONTER_MAX_PAGE_NUM];
   dma_addr_t dma_pages[MONTER_MAX_PAGE_NUM];
   size_t page_num, size;
   int state, operation, incr_batch_num;
   unsigned batch_num, done_batch_num;
   uint32_t addr_a, addr_b;
   wait_queue_head_t fsync_wait_queue;
};
```

Struktura reprezentuje kontekst w obrębie jednego urządzenia fizycznego monter_dev. Pole mdev jest wskaźnikiem na urządzenie, do którego kontekst należy, kern_pages to wskaźnik na tablicę stron w z adresami w przestrzeni jądra, a dma_pages to tablica stron z adresami dla DMA. Pola page_num to liczba zaalokowanych stron, size to łączny zaalokowany rozmiar, batch_num to liczba bloków komend, które trafią do kolejki urządzenia w obrębie danego

kontekstu, podobnie done_batch_num to liczba wykonanych bloków komend. Pola addr_a, addr_b są pomocnicze dla funkcji parse_addr_ab i parse_run_op. Natomiast fsync_wait_queue służy do czekania w funkcji monter_fsync aż wszystkie bloki poleceń w obrębie danego kontekstu zostaną wykonane.

```
struct cmd_batch {
  uint32_t cmds[32];
  int num;
  struct monter_context *context;
  struct list_head queue;
};
```

Struktura reprezentująca blok poleceń wczytanych prze funkcję monter_write, które zostaną zlecone do wykonania przez urządzenie. Pole cmds to tablica poleceń, num to liczba poleceń w tablicy, context to wskaźnik na kontekst, do którego należy blok zadań. Pole queue wiąże bloki poleceń w listę.

Algorytm zlecania poleceń do urządzenia:

Polecenia wczytywane są przez funkcję monter_write, następnie parsowane i sprawdzana jest ich poprawność. Jednocześnie grupowane są po w bloki po 32 polecenia, zapisane w postaci struktury cmd_batch i które dodane są do kolejki cmd_queue dla danego urządzenia fizycznego. Pod konie każdego bloku cmd_batch ostatnie polecenie wyzwala przerwanie NOTIFY, jeżeli liczba poleceń wystarcza na zapełnienie całego bloku. W przeciwnym wypadku jako ostatnie dodawane jest polecenie COUNTER wyzwalające przerwanie NOTIFY. Jeżeli urządzenie nie wykonuje w danym momencie żadnej pracy to dodatkowo wysyłane jest pojedyncze polecenie COUNTER wyzwalające przerwanie NOTIFY. Dalsza część wykonywana jest wewnątrz funkcji obsługi przerwań monter_irq_handler. W przypadku pojawienia się przerwania NOTIFY sprawdzane jest czy lista bloków poleceń cmd_queue czekających na wykonanie nie jest pusta. Dalej wykonywana jest zmiana kontekstu, w którym urządzenie wykonuje polecenia jeśli jest to potrzebne. Finalnie bloki poleceń z listy dodawane są do kolejki poleceń urządzenia. Po każdym wykonanym bloku budzona jest kolejka fsync_wait_queue.

Funkcje:

Funkcje monter_init, monter_exit służą do inicjalizacji i wyjścia z modułu, ponadto są dość standardowe.

Funkcja monter_probe inicjalizuje wszystko, co związane jest z pojedynczym

urządzeniem fizycznym. Funkcja alokuje pamięć na strukturę monter_dev, ustawia i inicjalizuje jej pola, inicjalizuje pola struktury pci_dev i cdev oraz aktywuje urządzenie PCI i sprawia, że jest ono widoczne w sysfs. Dalej ustawia odpowiednie rejestry urządzenia i dodaje je do listy urządzeń device_list_begin.

Funkcja monter_remove wycofuje operacje wykonane w monter_probe.

Funkcja monter_open tworzy nowy kontekst urządzenia.

Funkcje monter_ioctl pozwala na zaalokowanie pamięci do DMA.

Funkcja monter_mmap pozwala na zmapowanie pamięci do przestrzenie użytkownika.

Funkcja monter_fsync czeka na wykonane wszystkich poprawnych bloków poleceń wczytanych przez funkcję monter_write za pomocą kolejki oczekiwania, która budzona jest z funkcji obsługi przerwań.

Funkcja monter_write wczytuje polecenia od użytkownika, parsuje je, tworzy bloki poleceń cmd_batch i dodaje je do kolejki cmd_queue danego urządzenia.

Funkcja monter_release zwalnia zasoby zajęte przez monter_open dla danego kontekstu.

Funkcja obsługi przerwań monter_irq_handler sprawdza czy przerwanie przyszło od urządzenia MONTER, a następnie obsługuje je zgodnie z opisem w algorytmie.

Pozostałe funkcje są funkcjami pomocniczymi.