ПРОКЛЯТЫЕ ОПЕРАЦИОННЫЕ СИСТЕМЫ

Naszódi Mátyás

Основными целями операционных систем являются следующие:

1. Дать единообразное обращение к многообразному железу (драйверы, файл— и арифметические процессоры и т.д.).
2. Увеличить эффективность машины (многопользовательские системы, параллельные процессы, очереди и т.д.).
3. Обеспечить безопасность пользователям (друг от друга, от ошибок программ и устройств).

Часто решая эти задачи, операционная система представляет больше трудности программистам, чем облегчения работы. Мой пример: организация вывода—ввода в системах ДОС—РВ на машинах типа СМ.

Программы, написанные на языках высокого уровня, используют следующие примитивы:

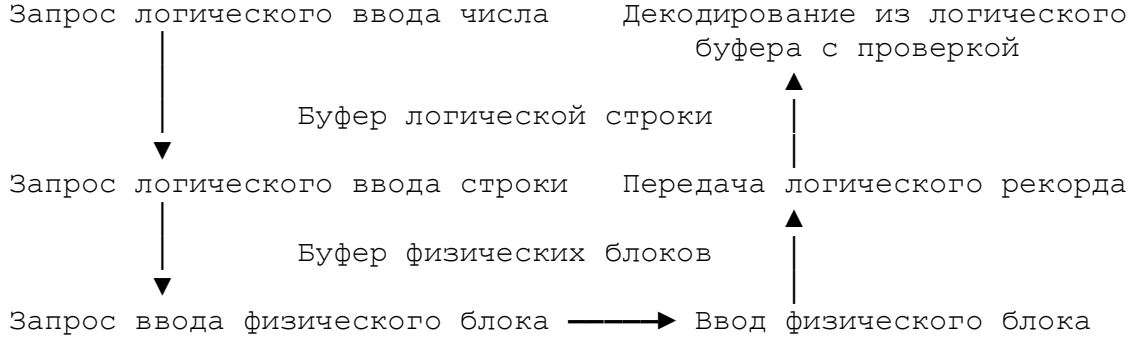
1. Открытие файла
2. Операция ввода и вывода логических рекордов
3. Закрытие файла

В принципе пользователь не должен знать ничего о том, какого типа физически накопитель "В практике это не так" программы должен знать, например, о том, что можно ли перемотать логический файл, или нет. О таких проблемах (вопрос физической-логической связи) говорят, что они на ответственности пользователей. К сожалению, единообразное обращение и файлам представляет более серьезные проблемы. Пример на это ввод с терминала.

Терминал, это устройство сильно отличающееся от других:

1. Во время ввода генерируется и вывод (метод ЭХО)
2. Важный "узел" терминала, это человек.

Можно ожидать, что система поддерживает интерактивное программирование, но это не так, обычно иерархия ввода числа следующая:



Хотя у терминала Логическая строка совпадает с физическим блоком, при такой организации пользователь узнает о своей ошибке только после того, как он нажал кнопку возврата и началу строки (cr).

Хороший пример решения найдется в системе IDOS.RIO. В ней используются два основных примитива для терминала:

1. DPI — ввод с клавиатуры без ЭХО
2. DPO — вывод знака

Здесь примитивы находятся очень близко к физическим возможностям. С помощью этих функции можно построить все общие функции ввода и вывода с терминала„ удобные для программирования интерактивно по знакам (а не только по строкам). Непосредственно написать эти примитивы на языках высокого уровня невозможно. Даже при попытке построить эти функции из примитивов имеющихся в системе ДОС—РВ, появляются временные барьеры (время ответа на нажатую кнопку станет очень длинным). Единственным решением является "симуляция" прерывания с помощью трепа (AST).

Почему нельзя использовать непосредственно возможности железа?

Что потеряем и что выиграем в этом случае?

Потери:

1. Единообразное обращение к файлам.
2. Безопасность

Выигрыш:

1. Оптимальное обращение и устройствам
2. Не потеряем специфические возможности устройств. Даже потеря может устроить, если изменим метод подготовки программ.

Обычно интерфейс программ с системой выглядит следующим образом:

программа	система	устройство
единообразное обращение к устройствам		трансляция из общего обращения в специальное

Предлагаю, чтобы интерфейс был такого же низкого уровня, как это требуется логикой устройства:

программа	система	устройство
обращение специально к устройству		передача без сложной трансляции

А если язык требует единообразного контроля файлов, тогда:

программа	рутины к программе		система	устройство
	единообразное обращение к файлу	специальное обращение к устройству		передача без сложной трансляции

В этом случае резидентная часть системы потребует меньше места, и практически ничего не потеряно из возможности аппаратуры.

Надежность и безопасность системы тоже можно сохранить, примером пусть будет опытный контроль крейта КАМАКА.

Этот "ДРАЙВЕР" КАМАКА Я приготовил, чтобы единообразным и надежным способом управлять модулями КАМАК независимо друг от друга. Основные рутины (примитивы) Я написал посредством макро-определении, даже для контроллеров разного типа (CI06, CI 22) Этим драйвером выполняются все основные функции КАМАКА, модули крейта можно программировать независимо друг от друга, хотя контроллер может быть общим.

Например, еще покажу, как выглядят рутины драйвера графического дисплея для OMSI PASCAL.