Процессы UNIX

UNIX создан в полноценном виде для PDP-11 на языке Си. Стоит отметить, что язык был написан для себя, а не для продажи.

Для изучения будем использовать **Linux** - UNIX-подобную ОС. Это значит, что Linux построен на парадигмах UNIX. Также, Linux - с открытым исходным кодом.

В Linux есть все:

- системные вызовы UNIX BSD;
- поддерж. сообщ. прогр. (UNIX sys 5);
- POSIX (Portable Operating System Interface);

Основной абстракцией ОС является **процесс** - программа в стадии выполнения.

Исполняться может только **исполняемый файл** - программа (файл), полученная с помощью компиляции и линковки (прошедшая компиляцию и линковку).

В UNIX понятие процесса кардинально. Для UNIX процесс - самая главная абстракция. Процесс часть времени выполняется в режиме пользователя, [...], в ядре, в реентерабельной части ОС.

Любой процесс может быть вызван с помощью системного вызова **fork** (в данном случае - «развилка»). Системный вызов fork создает новый процесс (**процесс-потомок**), который является копией процесса-предка в том смысле, что потомок копирует код предка, дескрипторы открытых файлов, сигнальную маску, маску создания и т.п.

В старых UNIX код предка копировался в адресноое пространство потомка, то есть создавалось собственное виртуальное адресное пространство, в которое копировался код. Как следствие, в одной системе могло быть несколько копий программы. Поэтому в современных системах используется **оптимизированный fork**.

Небольшое отступление

Что значит «операционная система»?

операционная - вместо оператора.

система - состоит из подсистем (управления памятью, процессором, файлами, внешними устройствами).

Не существует ОС без файловой системы!

Что значит «виртуальное адресное пространство»?

виртуальное - кажущееся, возможное (но оно существует :/)

Обязательно должны быть описаны:

- карта [чего]
- таблицы страниц

Любая **таблица** - это массив структур. Любая **ОС** - множество структур, связанных между собой.

Основная таблица (по сути, основа ОС) - **таблица процессов**. Однако ее нет и быть не может (иначе система была бы неповоротливой). Поэтому система оперирует **системными списками** (в основном - с двусвязными).

Оптимизация fork

Универсальное решение - флаг **copy-on-write** (COW).

В отличие от старой версии, код не копируется в адресное пространство, а дескрипторы страниц потомка ссылаются на страницы адресного пространства предка. При этом для страниц адресного пространства предка обычные права доступа (read, write) меняются на only-read и ставится флаг COW. В результате, если или предок, или потомок попытаются изменить значение страниц, то возникнет исключение по правам доступа. Обрабатывая это исключение, система обнаружит флаг COW и создаст копию данной страницы в адоесном пространстве того процесса, который пытался ее изменить (создаст копии только нужных страниц).

Из-за флага COW все ОС поддерживают виртуальную страничную память (управление памятью страницами по запросу). Таким образом, была решена проблема коллективного использования страниц.

Страницы удобны, так как подмена не требует много ресурсов (*paging*).

Код «развилки»

```
pid_t childid;

if ((childid = fork()) == -1)
{
        /* error */
}
else
{
        /* child */
}
else { /* parent */ }
```

Потомку - 0, предку - id потомка.

Процесс может попродить любое количество потомков. Следствие - **fork-бомба** (в итоге ресурсы «уходят» и теряется работоспособность).

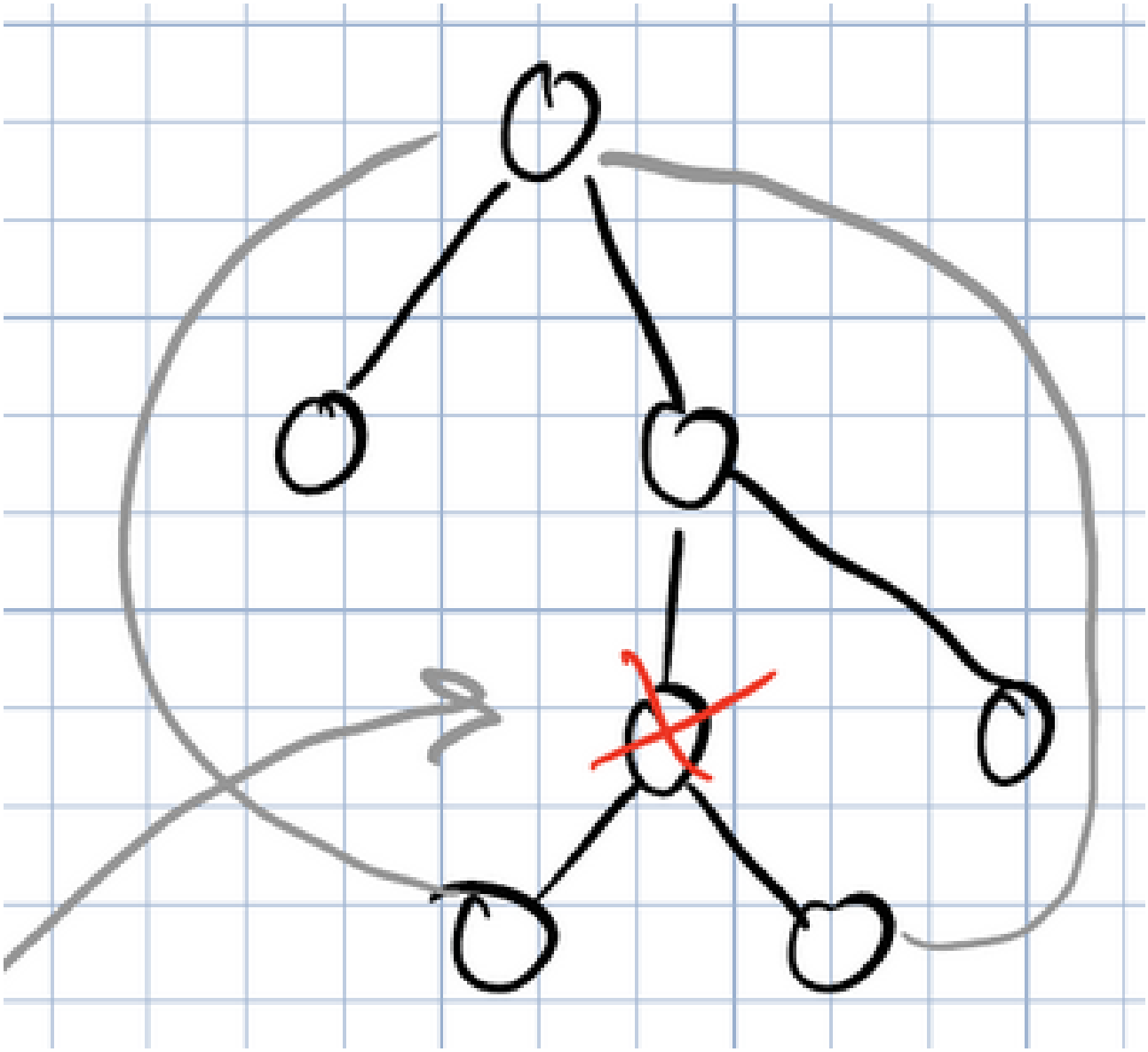
В результате вызова fork создается иерархия процессов в отношении «предок-потомок», поддерживаемая системой при помощи указателей в дескрипторе процесса.

В Linux: **struct task_struct { ... }**

В UNIX BSD: **struct proc { ... }**

В этой странице имеются указатели на предка и потомка.

Завершение процесса



Процессы сироты - процессы без родителей.

Система продолжает сохранять иерархию процессов. Поэтому сироты усыновляются процессом, открывшим терминал ($cpid = 1$).

Но, как правило, все-таки есть посредники.

В системе всегда есть процессы с идентификатором 0 (**процесс, запустивший систему**) и 1 (**процесс, открывший терминал**) вне зависимости от количества терминалов (не нужно резервировать место под каждый терминал).

Чтобы сироты не возникали, вызывается **wait(&status)** - блокирует процесс-предка до завершения процесса-потомка (процесс помечается ожидающим до завершения процесса-потомка).

Такая ситуация с правами доступа к адресному пространству предка будет существовать в системе до тех пор, пока потомок не вызовет **exec** или **exit**. **exit** завершает процесс, а **exec** переводит процесс-потомок на адресное пространство программы, которое передано в качестве аргумента; в результате потомок начинает выполнять другую программу.

Другими словами, в UNIX, чтобы запустить программу, нужно вызвать fork, и потомок должен вызвать exec (fork -> процесс -> exec -> программа).

Момент: Предки могут анализировать статус выполнения потомков.

Кроме иерархии, система UNIX поддерживает **группы процессов**. Процессы одной группы могут получать одни и те же сигналы.

Сигналы - важнейшее средство информирования о событиях в системе.

Важнейшее событие - завершение процесса.

Прерывание INT 8h

<дописать>