Семинар 2. Процессы в ОС UNIX

Все построение операционной системы UNIX основано на использовании концепции процессов. Каждый процесс в операционной системе получает свой собственный уникальный идентификационный номер PID (Process IDentificator). При создании нового процесса операционная система пытается присвоить ему свободный номер больший, чем у процесса, созданного перед ним. Если таких свободных номеров не оказывается (например, мы достигли максимально возможного номера для процесса), то операционная система выбирает минимальный из всех свободных номеров. В операционной системе Linux присвоение идентификационных номеров процессов начинается с номера 0, который получает процесс kernel при старте операционной системы.

Иерархия процессов. В операционной системе UNIX все процессы кроме одного, создающегося при старте операционной системы, могут быть порождены только какими-либо другими процессами. В качестве процесса прародителя всех остальных процессов в разных UNIX-ообразных системах могут выступать процессы с номерами 1 или 0. В операционной системе Linux таким родоначальником, существующим только при загрузке, является процесс kernel с идентификатором 0.

Таким образом, все процессы в UNIX связаны отношениями процесс-родитель - процесс-ребенок, образуя генеалогическое дерево процессов. Для сохранения целостности генеалогического дерева в ситуациях, когда процесс-родитель завершает свою работу до завершения выполнения процесса-ребенка, идентификатор родительского процесса в данных ядра процесса-ребенка (PPID - Parent Process IDentificator) изменяет свое значение на значение 1, соответствующее идентификатору процесса init, время жизни которого определяет время функционирования операционной системы. Тем самым процесс init как бы усыновляет осиротевшие процессы. Наверное, логичнее было бы изменять PPID не на значение 1, а на значение идентификатора ближайшего существующего процесса-прародителя умершего процесса-родителя, но в UNIX почему-то такая схема реализована не была.

Системные вызовы getppid() и getpid(). Данные ядра, находящиеся в контексте ядра процесса, не могут быть прочитаны процессом непосредственно. Для получения информации о них процесс должен совершить соответствующий системный вызов. Значение идентификатора текущего процесса может быть получено с помощью системного вызова *getpid(),* а значение идентификатора родительского процесса для текущего процесса - с помощью системного вызова *getppid()*.

Создание процесса в UNIX. Системный вызов *fork().* В операционной системе UNIX новый процесс может быть порожден единственным способом - с помощью системного вызова *fork()*. При этом вновь созданный процесс будет являться практически полной копией родительского процесса. У порожденного процесса по сравнению с родительским процессом (на уровне уже полученных знаний) изменяются значения следующих параметров:

1) идентификатор процесса - PID;

2) идентификатор родительского процесса - PPID;

Дополнительно к ним может измениться поведение порожденного процесса по отношению к некоторым сигналам, о чем подробнее будет рассказано позже, когда мы будем говорить о сигналах в операционной системе UNIX.

В процессе выполнения системного вызова *fork()* порождается копия родительского процесса и возвращение из системного вызова будет происходить уже как в родительском, так и в порожденном процессах. Этот системный вызов является единственным, который вызывается один раз, а при успешной работе возвращается два раза (один раз в процессе-родителе и один раз в процессе-ребенке)! После выхода из системного вызова оба процесса продолжают выполнение регулярного пользовательского кода, следующего за системным вызовом.

Для того, чтобы после возвращения из системного вызова *fork()* процессы могли определить, кто из них является ребенком, а кто родителем, и, соответственно, по-разному организовать свое поведение, он возвращает в них разные значения. При успешном создании нового процесса в процесс-родитель возвращается положительное значение равное идентификатору процесса-ребенка. В процесс-ребенок же возвращается значение 0. Если по какой-либо причине создать новый процесс не удалось, то системный вызов вернет в инициировавший его процесс значение -1. Таким образом, общая схема организации различной работы процесса-ребенка и процесса-родителя выглядит так:

```
pid = fork();
if(pid == -1){
...
/* ошибка */
...
} else if (pid == 0){
...
/* ребенок */
...
```

} else {

...

/ родитель */*

...

}

Параметры функции main() в языке С. Переменные среды и аргументы командной строки. У функции *main()* в языке программирования С существует три параметра, которые могут быть переданы ей операционной системой. Полный прототип функции *main()* выглядит следующим образом:

*int main(int argc, char *argv[], char *envp[]);*

Первые два параметра при запуске программы на исполнение командной строкой позволяют узнать полное содержание командной строки. Вся командная строка рассматривается как набор слов, разделенных пробелами. Через параметр *argc* передается количество слов в командной строке, которой была запущена программа. Параметр *argv* является массивом указателей на отдельные слова. Так, например, если программа была запущена командой

a.out 12 abcd

то значение параметра *argc* будет равно 3, *argv[0]* будет указывать на имя программы - первое слово - *"a.out", argv[1]* - на слово "*12*", *argv[2]* - на слово "*abcd*". Заметим, что, так как имя программы всегда присутствует на первом месте в командной строке, то *argc* всегда больше 0, а *argv[0]* всегда указывает на имя запущенной программы.

Анализируя в программе содержимое командной строки, мы можем предусмотреть ее различное поведение в зависимости от слов следующих за именем программы. Таким образом, не внося изменений в текст программы, мы можем заставить ее работать по-разному от запуска к запуску. Например компилятор *gcc*, вызванный командой *gcc 1.c* будет генерировать исполняемый файл с именем *a.out,* а при вызове командой *gcc 1.c -o 1.exe* - файл с именем *1.exe.*

Описание переменной *envp* будет дано на следующем семинаре.

Размер массива аргументов командной строки в функции *main()* мы получали в качестве ее параметра. Так как для массива ссылок на параметры окружающей среды такого параметра нет, то его размер определяется другим способом. Последний элемент этого массива содержит указатель *NULL*.

Изменение пользовательского контекста процесса. Семейство функций для системного вызова *exec().* Для изменения пользовательского контекста процесса используется системный вызов *exec()*, который пользователь не может вызвать непосредственно. Вызов *exec()* заменяет пользовательский контекст текущего процесса на содержимое некоторого исполняемого файла и устанавливает начальные значения регистров процессора (в том числе устанавливает программный счетчик на начало загружаемой программы). Этот вызов требует для своей работы задания имени исполняемого файла, аргументов командной строки и параметров окружающей среды. Для осуществления вызова программист может воспользоваться одной из 6 функций: *execlp(), execvp(), execl(), execv(), execle(), execve(),* отличающихся друг от друга представлением параметров, необходимых для работы системного вызова *exec()*. Взаимосвязь указанных выше функций изображена на рисунке.