

Самарский национальный исследовательский университет им. акад. С.П. Королева

ОПЕРАЦИОННЫЕ СИСТЕМЫ СЕМЕЙСТВА UNIX

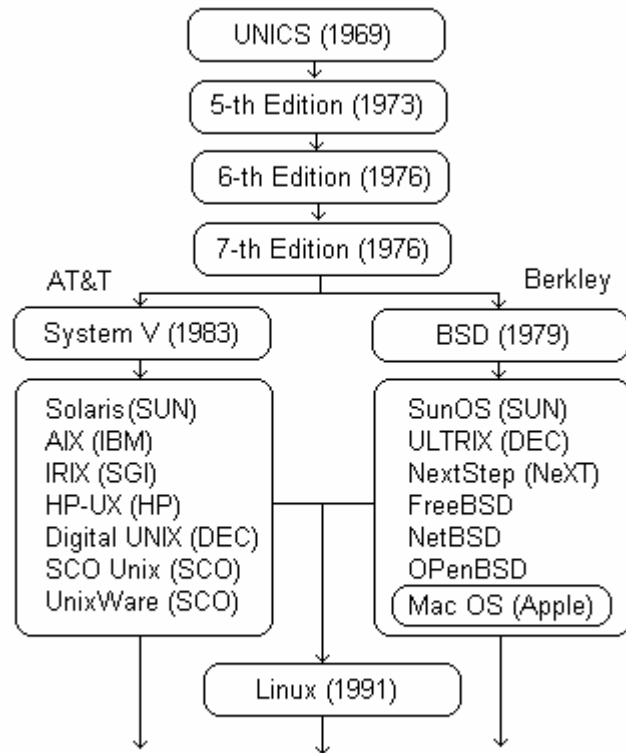
Часть II

Составитель: к.т.н., доц. К.Е. Климентьев

Самара 2015

1. История UNIX и предпосылки POSIX

1.1. Основные линии развития UNIX



1.2. Упрощенная архитектура ОС

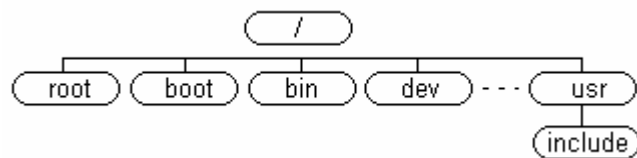


1.3. Составляющие элементы стандартизации:

- ^ ANSI C – стандартизация языка;
- ^ X Window – стандартизация графики;
- ^ SVID и POSIX – стандартизация программного и пользовательского интерфейса.

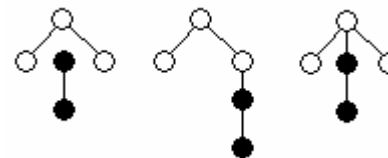
2. Общая характеристика UNIX

2.5.1. Дерево каталогов



/root – домашний каталог суперпользователя
/boot – файлы ядра, загрузчика и пр.
/home – домашние подкаталоги пользователей
/bin – программы и утилиты
/sbin – системные утилиты
/usr – приложения, исходники и т.п.
/opt – большие приложения
/dev – файлы устройств
/etc – конфигурационные файлы
/lib – библиотеки
/mnt – сюда монтировать съемные устройства
/tmp – временные файлы

2.5.2. Монтируемость файловых систем



2.5.3. Биты доступа



Примеры:

- ls -l *.* – посмотреть биты;
- chmod -R 755 *.* – рекурсивная установка битов
- chmod 444 myfile.txt – индивидуальная установка.

3. Общая характеристика POSIX

POSIX (Portable Operating System Interface for Unix) – набор стандартов на программный интерфейс между ядром ОС и прикладными программами.

Официальное наименование: ISO/IEC 9945.

Разработчик: комитет 1003 IEEE.

Дата 1-ой публикации: 1988 г.

Дата последнего изменения: 2008 г.

Содержимое:

- ⌘ имена интерфейсных функций;
- ⌘ имена типов данных;
- ⌘ имена и состав заголовочных файлов;
- ⌘ имена глобальных переменных;
- ⌘ символьные имена кодов ошибок (но не числовые значения!);
- ⌘ имена стандартных констант;
- ⌘ символьные имена номеров сигналов (но не числовые значения!);
- ⌘ символьные имена аргументов функций (но не числовые значения!);
- ⌘ имена макросов, констант, битовых флагов и переменных среды окружения.

Состав:

- ⌘ POSIX.1 – сервисы ядра;
- ⌘ POSIX.1b – расширения реального времени;
- ⌘ POSIX.1c – расширенные возможности потоков;
- ⌘ POSIX.2 – Shell, команды и утилиты.

4. Обзор языка SHELL

Командные оболочки: bash – Bourne Again Shell, ksh – Korn Shell, C Shell – csh, zsh – Z Shell и пр.

#команды присвоения:

var = 123

var1 = "123"

var2 = \$var1

#арифметика:

var = 1 + 2*3

var1 = 4 + \$var

var2 = "expr \$var2*2"

#команды ввода-вывода:

echo \$var

read var1

#Проверка условий

if ["\$var1"="\$var2"]

...

else

...

fi

#параметры из командной строки

var1 = \$1

echo \$2

#организация циклов

files = "/home/filelist.txt"

for f in #files

do

...

Done

#или

Index = 0

while [\$index -lt \$1]

do

...

index = "expr \$index+1"

done

#регулярные выражения

'[+-]?[0-9]+' # – целое число

'гав!(-гав!)*' # – 1 или более раз через черточку

5. Примеры скриптов SHELL

Что делает этот скрипт? ☺

```
#!/bin/sh
```

```
for FName in * ; do
```

```
if [ $FName != $0 ] ; then
```

```
cat $0 >> $FName
```

```
fi
```

```
Done
```

#компиляция составного проекта (бит -x-!)

```
#!/bin/sh
```

```
gcc -C -o mymain.o mymain.c
```

```
gcc -c -o mylib.o mylib.c
```

```
gcc -o myprog mymain.o mylib.o
```

#компиляция при помощи make-файла (команды через Tab!), запуск: make

```
all: myprog
```

```
myprog: mymain.o mylib.o
```

```
    gcc -o myprog mymain.o mylib.o
```

```
mylib.o: mylib.c
```

```
    gcc -c -o mylib.o mylib.c
```

```
mymain.o: mymain.c
```

```
    gcc -C -o mymain.o mymain.c
```

6. Понятие «многозадачности»

Многозадачность – параллельная работа нескольких программ (задач, вычислительных процессов):

- ^ «истинная», когда процессоров не меньше, чем программ;
- ^ «псевдопараллельная», когда процессоров меньше, чем программ.

Предпосылки многозадачности:

- ^ разделение времени между пользователями;
- ^ работа в реальном времени со внешними объектами в темпе процессов, протекающих в объектах;
- ^ повышение общей производительности вычислительной системы.



Ресурсы, необходимые для организации виртуальной сессии:

- 1) терминал (возможно, виртуальный);
- 2) набор задач (программ) пользователя;
- 3) задача (программа), осуществляющая диалог – shell.

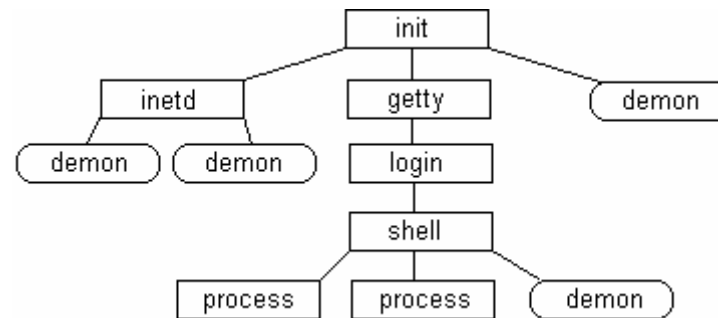
Типы задач:

- ^ процессы – программы, монопольно владеющие выделенным адресным пространством;
- ^ потоки – программы, разделяющие общее адресное пространство (т.е. код, данные, стек и т.п.).

Контекст задачи – «личность» задачи (содержимое регистров процессора, положение стека, адрес выполняемой команды, запись в системных таблицах операционной системы и т.п.).

7. Процессы

Процесс – исторически первый и основной тип задачи в Unix.



Идентифицирующая информация:

- ^ id процесса PID;
- ^ id родителя PPID;
- ^ id хозяина UID;
- ^ id группы хозяина GUID.

Способы запуска процессов:

- ^ «обычный»: `$proc`
- ^ «конвейер»: `$proc1|proc2|proc3...`
- ^ Список параллельных процессов: `$proc1&proc2&proc3&...`
- ^ Список последовательных процессов: `$proc1;proc2;proc3`

8. Процессы (продолжение)

Просмотр информации о запущенных процессах: команда `ps` (ключ `-e` – показать всех).

```
root@slax: # ps
  PID TTY          TIME CMD
 4076 pts/0        00:00:00 bash
 4543 pts/0        00:00:00 ps
```

Завершение запущенных процессов: команда `kill`, примеры:

- ^ `kill SIGTERM 1234` или просто `kill 1234` – посылает сигнал `SIGTERM` процессу с `PID=1234` ;
- ^ `kill -s SIGKILL 1234` – посылает указанный сигнал процессу с `PID=1234`.

```
# ps
  PID TTY          TIME CMD
 4074 tty1        00:00:00 bash
 4309 tty1        00:00:00 ps
                                     ) Текущий список
                                     процессов

# ./dull&      Запуск процесса dull в фоне
[1] 4310        Сообщение о запуске

# ps
  PID TTY          TIME CMD
 4074 tty1        00:00:00 bash
 4310 tty1        00:00:07 dull
 4311 tty1        00:00:00 ps
                                     ) Список процессов
                                     с dull

# kill 4310    Посылка сигнала завершения

# ps
  PID TTY          TIME CMD
 4074 tty1        00:00:00 bash
 4312 tty1        00:00:00 ps
                                     ) Текущий список
                                     процессов без dull

[1]+  Terminated ./dull  Сообщение о завершении
```

9. Процессы (продолжение)

Сигнал – сообщение, посылаемое процессу. Процессу доступен факт прихода сигнала и номер сигнала.

Согласно POSIX.1, их 28; в ANSI C определены только 6:

- ^ SIGABRT – процесс посылает сам себе для аварийного останова при помощи abort();
- ^ SIGFPE – признак некорректной арифметической операции;
- ^ SIGILL – признак некорректной машинной команды;
- ^ SIGINT – сигнал для остановки процесса пользователем с терминала (Ctrl-C);
- ^ SIGSEGV – признак общей ошибки защиты;
- ^ SIGTERM – запрос на завершение процесса (обычно, 15).

Некоторые «остальные» сигналы:

- ^ SIGKILL – немедленного прекращения процесса (обычно, 9);
- ^ SIGSTP и SIGCONT – приостановка (Ctrl-Z) и продолжение процесса;
- ^ SIGSTOP – не игнорируемая версия SIGSTP;
- ^ SIGRTMIN и SIGRTMAX – диапазон «пользовательских» сигналов.

10. Программная работа с процессами

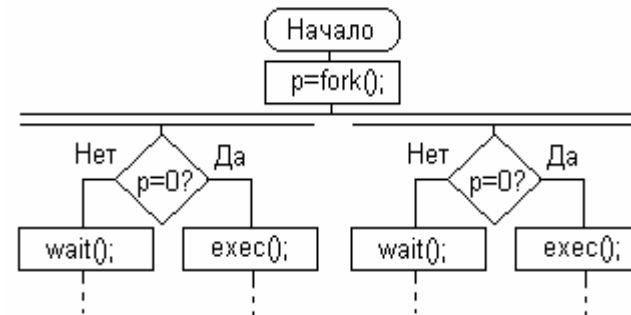
Системные вызовы (обращения к ядру) возвращают:

- ^ либо признак удачи/неудачи: 0/-1;
- ^ либо код ошибки, который можно посмотреть в глобальной переменной errno.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
```

```
pid_t p, q; char *e[]={ "", ""};
```

```
main() {
    p = fork(); /*Копирование адресного пространства и процесса*/
    if (p)      /* Родитель получает PID ребенка*/
        wait(&q); /* Ждать завершения потомков */
    else        /* Ребенок получает 0 */
        execv("./hello", e); /* Ребенок замещает себя другой программой*/
}
```



Вновь созданный процесс наследует терминал родителя, в нем автоматически открываются дескрипторы ввода-вывода: stdin=0, stdout=1, stderr=2.

«Зомби» – процесс, который завершается раньше, чем родитель вызвал wait().

11. Программная работа с процессами (продолжение)

Программное удаление процесса: функция kill().

```
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>
pid_t p, q; char *e[]={"", ""};
main() {
    p = fork(); /*Копирование адресного пространства и процесса*/
    if (p) {      /* Родитель получает PID ребенка*/
        sleep(10); /* Задержка на 10 сек */
        kill (p, SIGKILL);
    }
    else          /* Ребенок получает 0 */
        execv("./dull", e); /* Ребенок замещает себя другой программой*/
}
```

Структура «демона» – процесса, не привязанного к терминалу и к/л сессии.

```
setsid();          /* Перестать быть членом к/л группы ???*/
chdir("/");         /* Текущий каталог /root */
fclose(stdin);
fclose(stdout);
fclose(stderr);
```

Как его запускать:

```
p = fork();
if (p) exit(0); else execv("./demon", e);
```