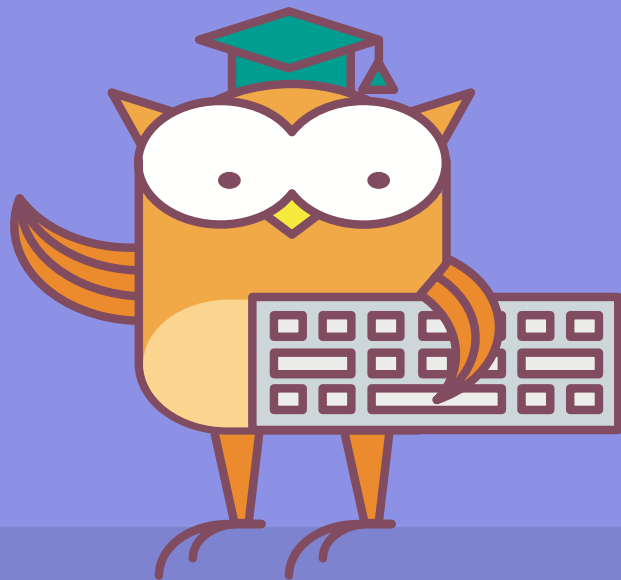


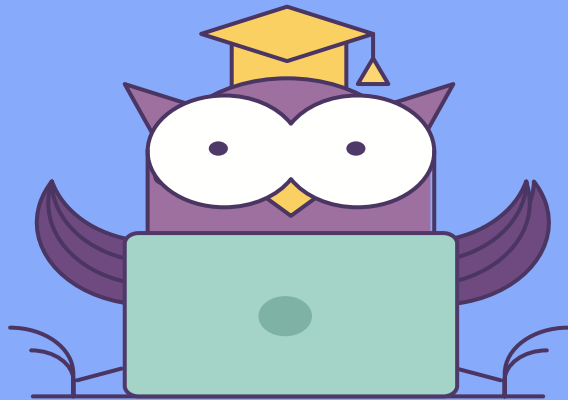


ОНЛАЙН-ОБРАЗОВАНИЕ


Управление процессами



Меня хорошо слышно && видно?



Напишите в чат, если есть проблемы!

Ставьте  если все хорошо

План вебинара

Процесс и его атрибуты

Сигналы



Жизнь процессов

Получение информации о
процессе

Процессы

Процесс как термин появился в ОС Multix и имеет несколько определений. Процесс - это

1. программа на стадии выполнения
2. объект которому выделено процессорное время
3. асинхронная работа

Процессы

Типы процессов:

1. Системные процессы - являются частью ядра и всегда расположены в оперативной памяти. Запускаются при инициализации ядра ОС.
2. Демоны - не интерактивные процессы, запуск путем загрузки исполняемого файла, выполняются в фоновом режиме. (dns, dhcp-server). Не связаны ни одним пользовательским сеансом, нет прямого пользовательского управления.
3. Пользовательские процессы - все остальные процессы, процессы запущенные пользователем. Порожденные в рамках пользовательского сеанса.

Атрибуты процесса

- Идентификатор процесса (**PID**).
- Идентификатор родительского процесса (**PPID**).
- Идентификатор владельца (**UID**) и эффективный идентификатор владельца (**EUID**).
- Идентификатор группы **GID** и эффективный идентификатор группы (**EGID**)

Атрибуты процесса

- Поправка приоритета (**NI**) (от -19 до 20)
- Терминальная линия (**TTY**) (управляющий терминал)
- **PRI** - приоритет планировщика
- Текущий каталог, корневой каталог, переменные программного окружения

- State
 - R - Running
 - D - Uninterruptable I/O
 - t - Trace
 - T - STOP

Атрибуты процесса

S - процесс не активен (sleeping);

Z - процесс зомби (defunct) =)

N - процесс с низким приоритетом (nice)

< - процесс с высоким приоритетом

S - процесс является лидером сессии

+ - процесс в группе foreground

Атрибуты процесса

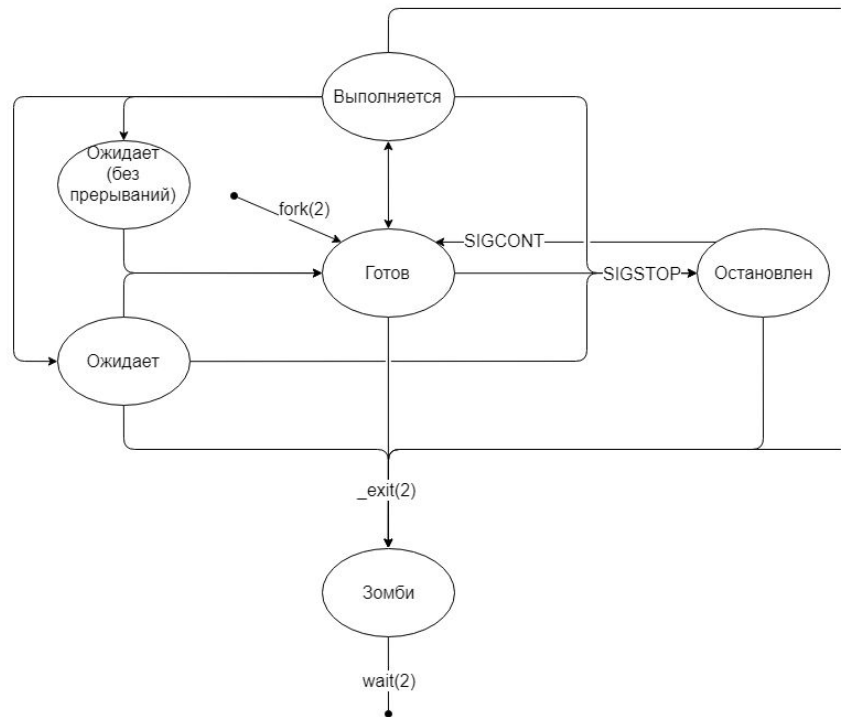
WCHAN - адрес события, которого ожидает процесс. У активного процесса этот столбец пустой.

STIME - время запуска процесса

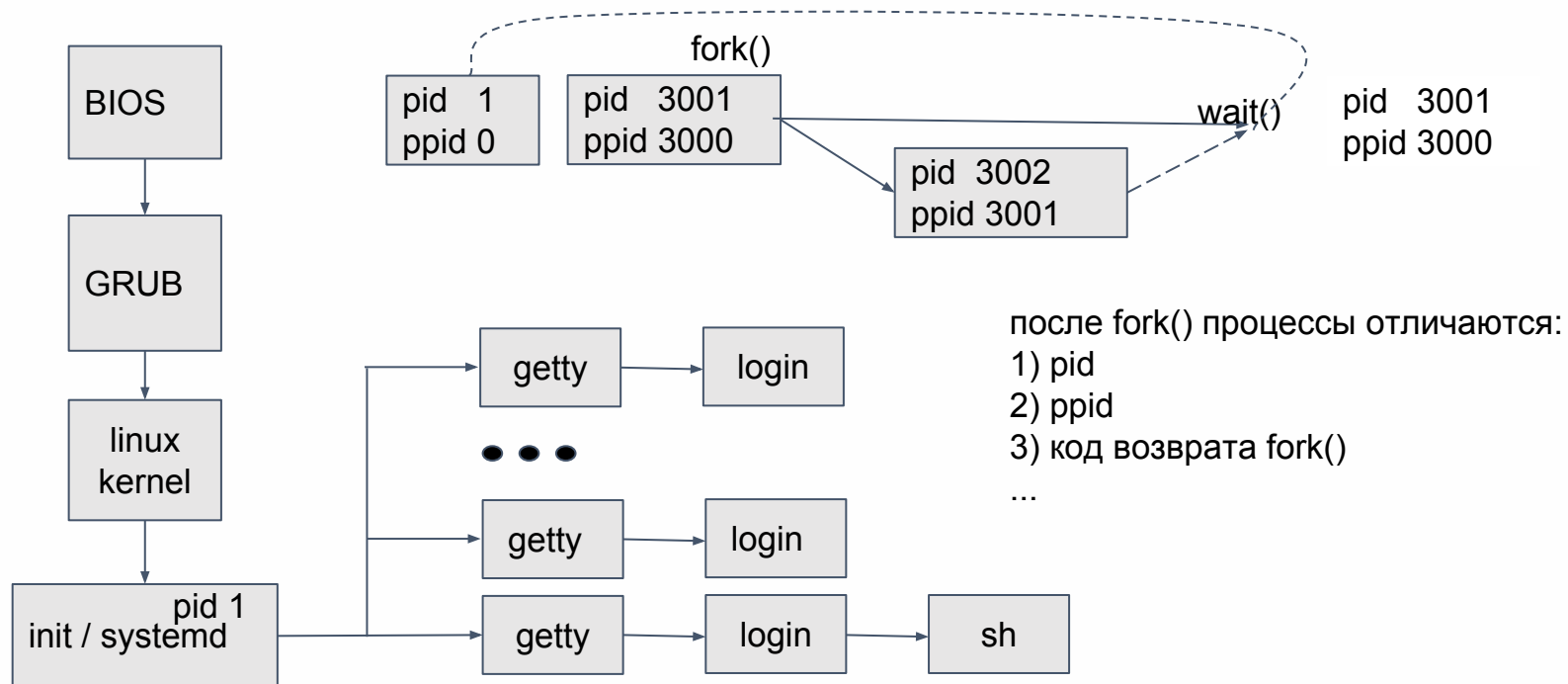
TIME - общее время выполнения процесса

SZ - Размер (в блоках по 512Байт) образа процесса в памяти

Жизненный цикл процесса



Ветвление процессов



Fork()

Выполнение fork()

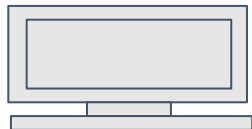
- Выделяется память для описания нового процесса в таблице процессов
- Назначается идентификатор процесса PID
- Создается логическая копия процесса, который выполняет fork() - полное копирование содержимого виртуальной памяти родительского процесса, копирование составляющих ядерного статического и динамического контекстов процесса-предка
- Увеличиваются счетчики открытия файлов (порожденный процесс наследует все открытые файлы родительского процесса).
- Возвращается PID в точку возврата из системного вызова в родительском процессе и 0 - в процессе-потомке.

Ресурсы дочернего процессы = ресурсы родительского процесса

Механизм Copy-on-write

Dirty COW (2007-2016)

Ветвление процессов



sh
pid 1000

tail -f /var/log/syslog
pid 1001

grep error > /tmp/log &
pid 1002

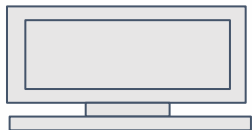
pgid 1001

tail -f /var/log/syslog
pid 1003

grep error
pid 1004

pgid 1003

sid 1000



sh
pid 1100

ps ax
pid 1101

grep http
pid 1102

pgid 1101

sid 1100

Процессы и сигналы



Сигналы – это программные прерывания.

- Сигнал является сообщением, которое система посылает процессу или один процесс посылает другому.
- Процесс получивший сигнал прерывает свою работу и передает управление обработчику сигнала. По окончании обработки процесс может продолжить работу.

Процессы и сигналы



Сигналы – это программные прерывания.

- Сигналы способны в случайное время (асинхронно) прерывать процесс для обработки какого-либо события.
- Процесс может быть прерван сигналом по инициативе другого процесса или ядра.
- Ядро использует сигналы для извещения процессов о различных событиях, например о завершении дочернего процесса.

Сигналы

#	NAME	значение
1	HUP	HangUP - чаще всего перечитать конфигурацию
2	INT	INTerrupt received
9	KILL	незамедлительное завершение процесса
11	SEGV	SEGmentation Violation
13	PIPE	broken PIPE - запись в pipe из которого никто не читает
15	TERM	TERMination signal
19	STOP	STOP Proces

Получение базовой информации о процессе

- `ps` - report process status
- `top` - display Linux process
- `pstree` - display a tree of process
- `/proc` - виртуальная (специальная) файловая система
- `lsof` - list open files
- `fuser` - показывает какой процесс держит файл =)

ps

Полезные опции программы ps:

- **-ef** или **ax** - расширенный вывод обо всех процессах
- **u** - информация о пользователе от которого запущен процесс (включено в -ef)
- **w** - расширить поле cmd (ww - не ограничивать)
- **f** - вывод дерева процессов
- **o** - определить формат вывода

ps

```
[root@linux-demo ~]# ps -eo pid,tid,class,rtprio,ni,pri,psr,pcpu,stat,wchan:14,comm,stime,time
```

PID	TID	CLS	RTPRIO	NI	PRI	PSR	%CPU	STAT	WCHAN	COMMAND	STIME	TIME
1	1	TS	- 0	19	3	0.0	Ss	ep_poll	systemd		12:49 00:00:01	
2	2	TS	- 0	19	1	0.0	S	kthreadd	kthreadd		12:49 00:00:00	
4	4	TS	- -20	39	0	0.0	l<	worker_thread	kworker/0:0H		12:49 00:00:00	
6	6	TS	- -20	39	0	0.0	l<	rescuer_thread	mm_percpu_wq		12:49 00:00:00	

ps

```
[root@linux-demo ~]# ps axfo pid,ppid,pgid,sid,stat,cmd | grep $$  
3375 3372 3375 3375 Ss      \_ -bash  
4025 3375 4025 3375 R+      \_ ps axfo pid,ppid,pgid,sid,stat,cmd  
4026 3375 4025 3375 S+      \_ grep --color=auto 3375
```

top

Полезные опции программы top:

- **-u username** - просмотр процессов пользователя
- **c** - показывает абсолютный путь к запущенному процессу
- **k** - убить выделенный процесс
- **Shift+p** - сортировка по процессору
- **h** - помощь =)

pstree

Полезные опции программы pstree:

- **-a** - аргументы командной строки
- **-p** - показывает PID
- **-u** - под каким пользователем запущен процесс
- **-h** - “подсветка” текущего процесса
- **-H PID_number** - подсветка процесса по PID

/proc

Специальная (виртуальная) файловая система. Позволяет получить доступ к информации из ядра о системных процессах.

`/proc/PID/limits` - лимиты процесса

`/proc/PID/status` - детальная информация о процессе

`/proc/PID/environ` - переменные окружения

`/proc/PID/exe` - ссылка на исполняемый файл

`/proc/PID/stat` - машиночитаемый файл с информацией о процессе

`/proc/PID/maps,statm,mem` - информация о памяти

- u - открытые файлы для пользователя
- i TCP:port - поиск процесса открытого на конкретном порту
- i 4 - только ipv4 соединения
- i TCP:port - открытые порты в диапазоне
- i - соединение в статус LISTEN ESTABLISHED
- p - поиск по PID

Управление фоновыми процессами

- jobs - список запущенных задач.
- **&** - Выполнить задачу в фоновом режиме.
- **Ctrl+Z** - Приостановить выполнение текущей (интерактивной) задачи.
- suspend - Приостановить командный процессор.
- fg - Перевести задачу в интерактивный режим выполнения.
- bg - Перевести приостановленную задачу в фоновый режим выполнения.

Расширенная информация о процессе

Помимо того, чтобы наблюдать за процессом со стороны можно заглянуть ему под капот с помощью программ **gdb** и **strace/ltrace**.

strace

Утилита для диагностики и отладки. Используется для мониторинга и вмешательства во взаимодействие между процессом и ядром. Включает взаимодействие с системными вызовами, сигналами.

`pr_name -h` - отладка процесса

`-p PID` - отладка процесса (выбор по PID)

`-i` - отображение указателя команд во время каждого системного вызова

`-T` - отобразить время потраченное на вызовы

`trase=sys_cals` - отображение только конкретных вызовов

`-o file.log` - перенаправить трассировку в файл

`-e` - фильтрация вывода

`man strace` =)

ltrace

Утилита для диагностики и отладки. Используется для мониторинга и диагностики вызов из пространства пользователя к shared библиотекам.

- f - трассировка включая дочернии процессы
- o - вывод в файл
- r - указывать время
- S - отображать системные вызовы как будто это вызовы к библиотекам
- u - имя пользователя

man ltrace =)

gdb

Когда не хватает информации **strace** можно прибегнуть к более “суровым”/тяжелым методам.

Например, **gdb -p pid** или **gdb /path/to/program /path/to/core** и команда **bt**.

С помощью **gdb** можно сделать куда более сложные и интересные вещи. Например, поменять лимиты у работающего процесса:

<https://gchp.ie/updating-ulimit-on-running-linux-process/>

Изменение приоритетов процесса

С помощью утилиты `nice` можно изменить “привлекательность” процесса для планировщика ядра. Большее значение `nice` уменьшает приоритет, а меньшее - увеличивает.

С помощью утилиты `ionice` можно изменить приоритет процесса для планировщика ввода-вывода.

Есть 3 класса (1: `realtime`, 2: `best-effort`, 3: `idle`) приоритетов и 8 уровней приоритета для классов 2 и 3 (меньше уровень - больше приоритет).

Ваши вопросы?

**Заполните, пожалуйста,
опрос в ЛК о занятии**

**Спасибо
за внимание!
До встречи в Slack и на вебинаре**

