



ОНЛАЙН-ОБРАЗОВАНИЕ

Что такое неотзывчивая система



Что такое неотзывчивая система

- Долго отвечает определенный сервис
- Определенный сервис пропускает запросы
- Система в целом любой запрос выполняет долго
- Система в принципе не отвечает ни на какой запрос



Самая базовая метрика



Самая базовая метрика

LA - Load Average

Число процессов стоящих в очереди на исполнение

```
# cat /proc/loadavg  
0.13 0.05 0.13 1/89 5903
```

```
# uptime  
12:23:23 up 9 days, 23:57, 2 users, load average: 0,11, 0,05, 0,13
```

```
# top  
top - 12:24:23 up 9 days, 23:58, 2 users, load average: 0,04, 0,04, 0,12
```



Самая базовая метрика - LA

Число процессов стоящих в очереди на исполнение

Причина высокого LA - борьба за ресурсы.

- * CPU (cpu usage/process count)**
- * Disk (IO/SWAP)**
- * Сеть**



CPU Usage

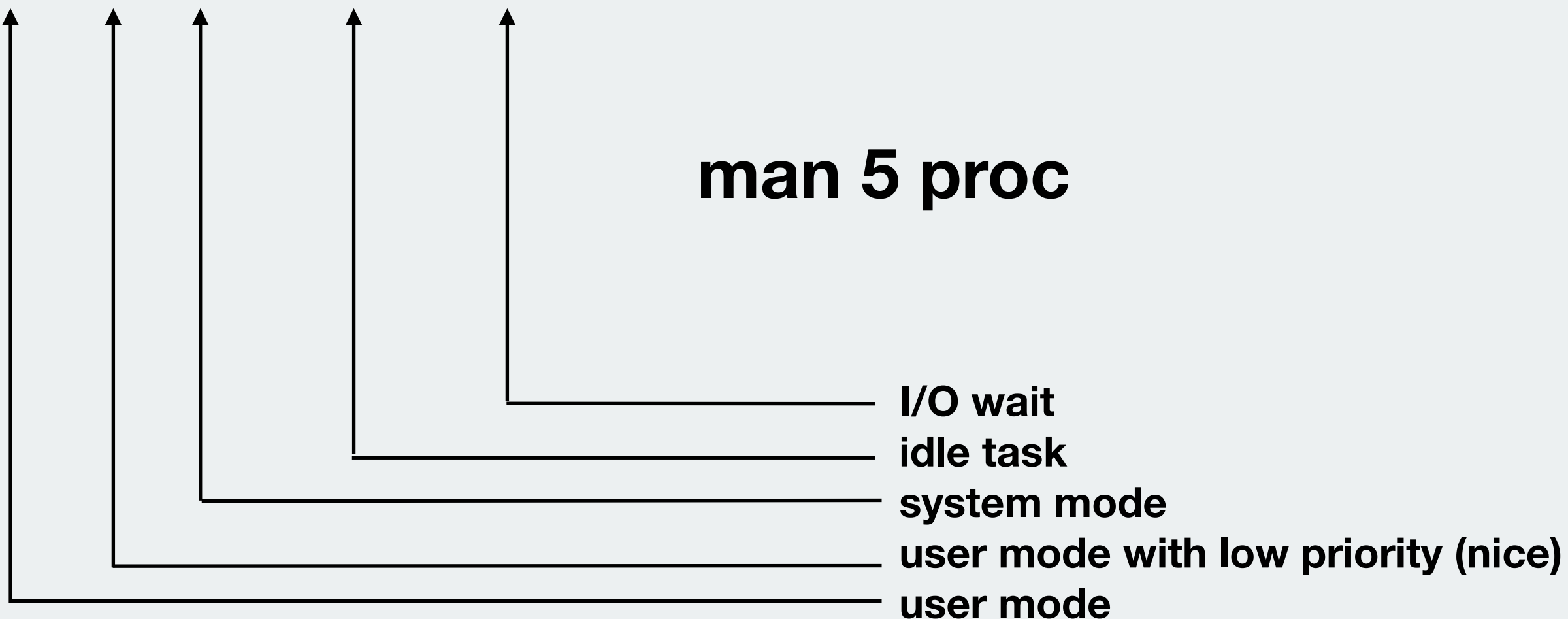
Борьба за ресурс CPU

Cpu usage большой, LA большой, мало процессов

cat /proc/stat

cpu 12318 488 9095 86328268 47261 0 102 3946 0 0

man 5 proc



CPU Usage

Борьба за процессорное время

```
# cat /proc/loadavg  
0.00 0.01 0.05 1/89 5928
```

Активных процессов

Всего процессов

Cpu usage маленький, LA большой, много процессов



CPU Usage

top



Disk Usage

CPU usage маленький, процессов мало, LA большой - мы упёрлись в IO

**vmstat
iostat
iotop
cpu iowait (top)**



Disk Usage

<https://www.kernel.org/doc/Documentation/iostats.txt>

```
# cat /proc/diskstats
```

```
8      0 sda 14569 1 514694 42456 116506 58787 15152456 1062240 0 539238 1104652
```

#13



Время, проведенное в IO,
мс

Мониторинг: $Dx/Dt \cdot 100$

<https://www.kernel.org/doc/Documentation/iostats.txt>



Disk Usage

CPU usage маленький, процессов мало, LA большой - мы упёрлись в IO

Но IO - это еще и сеть

Смотрим, что конкретно происходит с процессом
strace
/proc/<pid>/fd



Продолжаем про неотзывчивость

Всё неплохо, но клиенты не могут подключиться к сервису

Как обрабатываются соединения

```
listen_socket = listen()
fork()
client_socket = accept(listen_socket)
read(client_socket)/write(client_socket)/send(client_socket)/recv(client_socket)
close(client_socket)
```

Открытые дескрипторы наследуются при fork()



Продолжаем про неотзывчивость

Клиенты подключились, но соединения не обрабатываются

Сокеты в состоянии TIME_WAIT

`sysctl net.ipv4.ip_local_port_range` - диапазон портов для исходящих соединения

`net.ipv4.tcp_tw_reuse` и `net.ipv4.tcp_tw_recycle` - не рекомендуется использовать



Обрабатываем всплески

socket backlog

net.core.somaxconn = 128 → 1024

ss -l

Recv-Q Send-Q

sysctl -w net.core.rmem_max=8388608

sysctl -w net.core.wmem_max=8388608

sysctl -w net.core.rmem_default=65536

sysctl -w net.core.wmem_default=65536

sysctl -w net.ipv4.tcp_rmem='4096 87380 8388608'

sysctl -w net.ipv4.tcp_wmem='4096 65536 8388608'

sysctl -w net.ipv4.tcp_mem='8388608 8388608 8388608'



Таймауты

Большие таймауты - мина замедленного действия. Всегда и везде стоит урезать таймауты до адекватных значений

- **connect timeout: 3 sec max**
- **write timeout: 3 sec max**
- **read timeout - зависит от приложения**



Swap - неизбежное зло

Многие процессы используют swap вполне "легально". В swap ядро превентивно "складывает" давно неиспользованные блоки. При нехватке памяти при переключении контекста память процесса выгружается на диск, что бы через полсекунды загрузить обратно.

Есть ряд решений для систем с ограниченной памятью

- zswap
- zram

Обе технологии "обменивают" дисковые операции на ресурс CPU

на production `vm.swappiness` можно ставить в 0

https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/performance_tuning_guide/s-memory-tunables



TCP_DEFER_ACCEPT

Ядро ядро передаёт управление в accept(), только когда появились данные
Экономим переключения контекста

apache:
AcceptFilter

nginx:
listen deferred



Статистические утилиты

atop vs **sar**

atop - больше информации и ncurses-based интерфейс

sar - интерфейс классический а-ля vmstat

оба инструменты используют подсистему аккаунтинга ядра



monit

<https://mmonit.com/wiki/Monit/Monit>

Часть мониторинговой системы, интересная тем, что вы можем наблюдать за процессом и принимать те или иные действия в зависимости от состояния процессов

```
check process cron with pidfile /var/run/cron.pid
group system
start program = "/etc/init.d/cron start"
stop  program = "/etc/init.d/cron stop"
depends on cron_rc
```

```
check file cron_rc with path /etc/init.d/cron
group system
if failed checksum then unmonitor
if failed permission 755 then unmonitor
if failed uid root then unmonitor
if failed gid root then unmonitor
```



strace

Утилита, показывающая, чем занимается процесс: какие системные вызовы дёргает процесс

`strace -p <pid>` - режим отображения в реальном времени

`strace -c -p <pid>` - режим с накоплением статистики

`strace -ff -p <pid>` - режим с отслеживаем форков



Ничего не помогает

```
echo 1 > /proc/sys/kernel/sysrq  
echo b > /proc/sysrq-trigger
```





Спасибо
за внимание!

Вопросы?