Урок 6. Сетевые функции ядра

1. Как можно классифицировать архитектуры ЭВМ по признакам наличия параллелизма в потоках команд и данных?
2. В чем отличительная особенность процессов-демонов в Linux (Daemons)?
3. Какие минусы у реального режима адресации?
4. Что показывает колонка Total в выводе утилиты free?

***Практическое задание (по желанию):**
1) Вывести информацию о количестве оперативной памяти в системе с помощью команды free в гигабайтах
2) Каким образом можно получить ту же самую информацию без использования команды free? (как минимум, 2 способа)

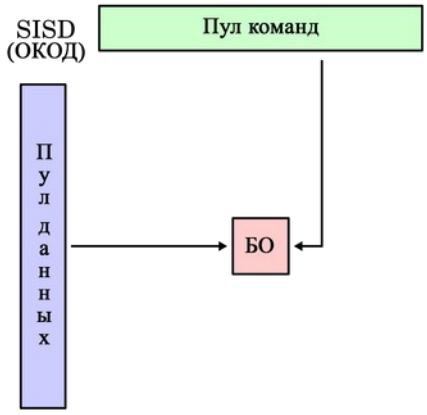
1. Как можно классифицировать архитектуры ЭВМ по признакам наличия параллелизма в потоках команд и данных?

Параллельные вычислительные системы - это физические компьютерные, а также программные системы, реализующие тем или иным способом параллельную обработку данных на многих вычислительных узлах.
Параллельные вычисления существуют в нескольких формах:

- параллелизм на уровне битов;
- параллелизм на уровне инструкций;
- параллелизм данных;
- параллелизм задач.

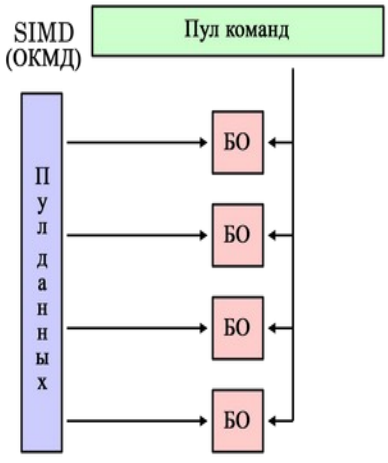
Классификации архитектур вычислительных систем по *Флинну (Flynn)* основывается на понятии потока – последовательность команд, элементов или данных, обрабатываемая процессором. Флинн выделяет четыре класса архитектур на основе числа потоков команд и потоков данных:

SISD (Single Instruction, Single Data или ОКОД) - один из наиболее распространённых типов параллельных ЭВМ. К данному классу относят векторные процессоры; обычные современные процессоры, когда выполняют команды векторных расширений; матричные процессоры. Здесь один процессор загружает одну команду, набор данных к ним и выполняет операцию, описанную в этой команде, над всем набором данных одновременно. Как пример, архитектура CISC, RISC



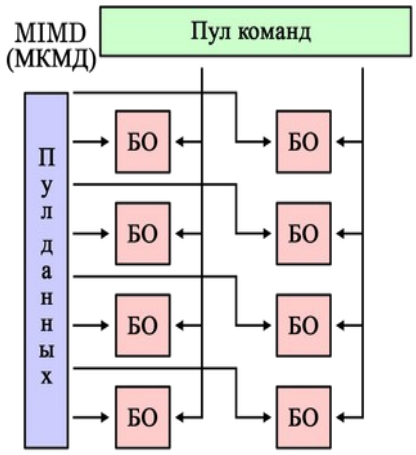
SIMD (single instruction stream / multiple data stream или ОКМД) – одиночный поток команд и множественный поток данных. Подобного рода архитектуры сохраняют один поток команд, включающий, векторные команды, в отличие от предыдущего класса. Это позволяет выполнять одну арифметическую операцию сразу над многими данными – элементами вектора. Как пример, системы CPP DAP, Gamma II и Quadrics Apemille.
SM-SIMD (shared memory SIMD) — подкласс SIMD с общей памятью с точки зрения программиста. Сюда относятся векторные процессоры.

DM-SIMD (distributed memory SIMD) — подкласс SIMD с распределённой памятью с точки зрения программиста. Сюда относятся матричные процессоры — особый подвид с большим количеством процессоров.



SIMD - обычная операция во многих мультимедийных приложений, как пример, изменение яркости изображения. Каждый пиксель изображения состоит из трех значений яркости RGB, считываются из памяти, значение добавляется (или вычитается из него) их, а полученные значения записываются обратно в память.

MIMD (multiple instruction stream / multiple data stream или МКМД) – распространённый тип параллельных ЭВМ. Включает в себя многопроцессорные системы, где процессоры обрабатывают множественные потоки данных. Сюда, как правило, относят традиционные мультипроцессорные машины, многоядерные и многопоточные процессоры, а также компьютерные кластеры.
MIMD компьютер имеет **N** процессоров, **N** потоков команд и **N** потоков данных. Каждый процессор функционирует под управлением собственного потока команд.

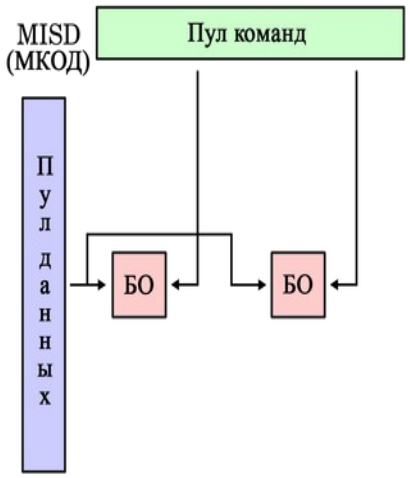


SM-MIMD (shared memory MIMD) — подкласс MIMD с общей памятью с точки зрения программиста. К нему относятся мультипроцессорные машины и многоядерные процессоры с общей памятью. Мультипроцессоры легко программировать, поддержка SMP (симметричной мультипроцессорности) давно присутствует во всех основных операционных системах. Однако у таких ЭВМ невысокая масштабируемость: увеличение процессоров в системе чревато высокой нагрузкой на общую шину.

DSM-MIMD (distributed shared memory MIMD) подкласс. То есть программисту память видна как одно общее адресное пространство, но физически она может быть распределена по узлам системы. В этом подклассе у каждого процессора есть своя локальная память, а к другим участкам памяти процессор обращается через высокоскоростное соединение. Поскольку доступ к разным участкам общей памяти неодинаков, такие системы называются NUMA (англ. Non-Uniform Memory Access). Возникает проблема: надо, чтобы процессор видел в памяти изменения, сделанные другими процессорами. Для её решения возникли ccNUMA (через согласование кэша) и nccNUMA (без согласования кэша). NUMA-системы имеют более высокую масштабируемость, по сравнению с мультипроцессорами, что позволяет создавать массово-параллельные вычислительные системы, где число процессоров достигает нескольких тысяч.
DM-MIMD (distributed memory MIMD) — подкласс MIMD с распределённой памятью с точки зрения программиста. Сюда относятся многопроцессорные ЭВМ с распределённой памятью и компьютерные кластеры типа Beowulf как Network of Workstations. Локальная память отдельно взятого процессора не видна другим. У каждого процессора своя задача. Если же ему

необходимы данные из памяти другого процессора, он обменивается с ним сообщениями. У таких машин высокая масштабируемость, как у NUMA.

MISD (multiple instruction stream / single data stream или МКОД) – множественный поток команд и одиночный поток данных. Данный класс предполагает, что в архитектуре имеется множество процессоров, которые обрабатывают один и тот же поток данных.
Является гипотетическим классом, поскольку реальных систем-представителей данного типа пока не существует. Некоторые исследователи относят к нему конвейерные ЭВМ.

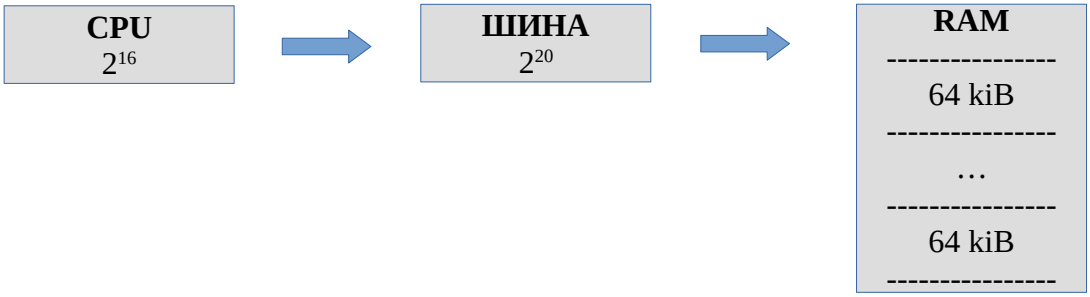


2. В чем отличительная особенность процессов-демонов в Linux (Daemons)?

Процесс — это программа, выполняющаяся в оперативной памяти компьютера.
Демоны отличаются от обычных процессов тем, что они работают в неинтерактивном режиме. Если с обычным процессом всегда ассоциирован какой-то терминал или псевдотерминал, через который осуществляется взаимодействие процесса с пользователем, то демон такого терминала не имеет. Демоны обычно используются для выполнения сервисных функций, обслуживания запросов от других процессов, причем не обязательно выполняющихся на данном компьютере. Пользователь не может непосредственно управлять демонами, он может влиять на их работу, только посылая им какие-то задания, например, отправляя документ на печать. Одним из главных демонов в системе является демон init – он является прародителем всех процессов в системе и имеет идентификатор 1. Выполнив задачи, поставленные в ему в файле inittab, демон init не завершает свою работу – он постоянно находится в памяти и отслеживает выполнение других процессов.

3. Какие минусы у реального режима адресации?

Рассмотрим на примере интел-процессора 8086 (i80x86):



Реальный режим поддерживается на всех x86-совместимых процессорах: от 8086 до современных 64-разрядных процессоров Intel. В процессоре 8086 используется *20-битная шина* адресов, то есть он может работать с адресным пространством 0-0xFFFFF или 1 мегабайт. Но у него есть только 16-битные регистры с максимальным адресом 2^{16}-1 или 0xffff (64 килобайта). **Поскольку 2^{20} = 1048576, то фактически доступная память составляет 1 МБ.*

Адрес состоит из двух частей:

1. селектор сегмента с базовым адресом;
2. смещение от базового адреса.

В реальном режиме базовым адресом селектора сегмента является селектор сегмента * 16. Таким образом, чтобы получить физический адрес в памяти, нужно умножить часть селектора сегмента на 16 и добавить к нему смещение:

*Физический адрес = Селектор сегмента * 16 + Смещение*

Выделим **два основных недостатка схемы адресации памяти** реального режима:

- ограниченное адресное пространство (до 1 Мбайта и еще примерно 64 Кбайта старшей области памяти для процессоров 80286 и старше);
- свободный доступ любых программ к любым областям данных, что представляет потенциальную опасность для целостности операционной системы.

Дополнительно: под контекстом процесса понимают совокупность той информации, которая необходима для организации переключения между процессами, а именно:

- Указатели на адресное пространство процесса в режиме задачи. Сюда входят указатели на сегменты кода, данных и стека, а также указатели на области разделяемой памяти и динамических библиотек.
- Окружение процесса, т. е. перечень заданных для данного процесса переменных с их текущими значениями.
- Аппаратный контекст процесса, то есть значения общих и ряда системных регистров процессора. Сюда относятся состояния счетчика выполняемых команд (указатель на адрес очередной исполняемой инструкции), указатель стека и так далее.
- Указатели на каждый открытый процессом файл, а также указатели на два каталога – домашний (или корневой) каталог процесса и его текущий каталог. Счетчики числа обращений в индексных дескрипторах этих каталогов увеличиваются на единицу при создании процесса (при смене текущего каталога), в силу чего вы (или другой процесс) не можете удалить эти каталоги, пока процесс их не «освободит».
-

Поэтому можно сказать, что контекст процесса и его виртуальное адресное пространство образуют как бы «виртуальный компьютер», в котором и исполняется процесс.

4. Что показывает колонка Total в выводе утилиты free?

- `free`
- **total** - это число представляет собой общий объем памяти, который может использоваться приложениями.
- **used** - использованная память, рассчитывается как:
  used = total - free - buffers - cache
- **free**- Свободная / Неиспользуемая память.
- **shared** - этот столбец можно игнорировать, так как он не имеет смысла. Это здесь только для обратной совместимости.
- **buff / cache** - объединенная память, используемая буферами ядра, кешем страниц и слэбами. / Эта память может быть восстановлена в любое время, если это необходимо приложениям.
  -w: для отображения буфера и кеша в двух отдельных столбцах.
- **available** - оценка объема памяти, доступной для запуска новых приложений без замены.

```
┌──(kali㉿kali)-[~]
└─$ free --gibi
               total        used        free      shared  buff/cache   available
Mem:               1           0           0           0           0           1
Swap:              0           0           0

┌──(kali㉿kali)-[~]
└─$ free -g
               total        used        free      shared  buff/cache   available
Mem:               1           0           0           0           0           1
Swap:              0           0           0

┌──(kali㉿kali)-[~]
└─$ free -l
               total        used        free      shared  buff/cache   available
Mem:         2028812      590500      971280        6336      467032     1285340
Low:         2028812     1057532      971280
High:              0           0           0
Swap:         998396           0      998396

┌──(kali㉿kali)-[~]
└─$ free -lh
               total        used        free      shared  buff/cache   available
Mem:           1.9Gi       576Mi       948Mi       6.0Mi       456Mi       1.2Gi
Low:           1.9Gi       1.0Gi       948Mi
High:             0B          0B          0B
Swap:          974Mi          0B       974Mi

┌──(kali㉿kali)-[~]
└─$ free -g -w
               total        used        free      shared     buffers       cache   available
Mem:               1           0           0           0           0           0           1
Swap:              0           0           0

┌──(kali㉿kali)-[~]
└─$ free -lh -w
               total        used        free      shared     buffers       cache   available
Mem:           1.9Gi       582Mi       942Mi       6.0Mi        55Mi       401Mi       1.2Gi
Low:           1.9Gi       1.0Gi       942Mi
High:             0B          0B          0B
Swap:          974Mi          0B       974Mi
```

Практическое задание:

1) Вывести информацию о количестве оперативной памяти в системе с помощью команды free в гигабайтах

- free -g

```
┌──(kali㉿kali)-[~]
└─$ free --gibi
               total        used        free      shared  buff/cache   available
Mem:               1           0           0           0           0           1
Swap:              0           0           0

┌──(kali㉿kali)-[~]
└─$ free -g
               total        used        free      shared  buff/cache   available
Mem:               1           0           0           0           0           1
Swap:              0           0           0
```

```
┌──(kali㉿kali)-[~]
└─$ free -lh
               total        used        free      shared  buff/cache   available
Mem:           1.9Gi       568Mi       969Mi       6.0Mi       442Mi       1.2Gi
Low:           1.9Gi       1.0Gi       969Mi
High:             0B          0B          0B
Swap:          974Mi          0B       974Mi

┌──(kali㉿kali)-[~]
└─$ free
               total        used        free      shared  buff/cache   available
Mem:         2028812      582836      992660        6340      453316     1293236
Swap:         998396           0      998396
```

2) Каким образом можно получить ту же самую информацию без использования команды free? (как минимум, 2 способа)

- cat /proc/meminfo
- vmstat -s
- top
- htop

```
┌──(kali㉿kali)-[~]
└─$ cat /proc/meminfo
MemTotal:        2028812 kB
MemFree:          158292 kB
MemAvailable:     773448 kB
Buffers:          138844 kB
Cached:           350044 kB
SwapCached:        21856 kB
Active:           294348 kB
Inactive:        1100092 kB
Active(anon):      67164 kB
Inactive(anon):   863296 kB
```

```
┌──(kali㉿kali)-[~]
└─$ vmstat -s
      2028812 K total memory
      1044716 K used memory
       326012 K active memory
      1121224 K inactive memory
       118292 K free memory
       140716 K buffer memory
       725088 K swap cache
       998396 K total swap
        70728 K used swap
       927668 K free swap
        20624 non-nice user cpu ticks
           24 nice user cpu ticks
        10066 system cpu ticks
       498535 idle cpu ticks
         1538 IO-wait cpu ticks
            0 IRQ cpu ticks
         1010 softirq cpu ticks
            0 stolen cpu ticks
```

```
top - 05:33:58 up 10 min,  1 user,  load average: 0.78, 0.61, 0.31
Tasks: 168 total,   1 running, 167 sleeping,   0 stopped,   0 zombie
%Cpu(s): 20.0 us,  0.0 sy,  0.0 ni, 80.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   1981.3 total,    242.2 free,   1071.7 used,    667.3 buff/cache
MiB Swap:    975.0 total,    974.0 free,      1.0 used.    727.4 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
     14 root      20   0       0      0      0 S   0.0   0.0   0:00.04 ksoftirqd/0
     15 root      20   0       0      0      0 I   0.0   0.0   0:00.36 rcu_preempt
     16 root      rt   0       0      0      0 S   0.0   0.0   0:00.00 migration/0
     17 root      20   0       0      0      0 I   0.0   0.0   0:00.16 kworker/0:1-events
     18 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/0
     19 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/1
     20 root      rt   0       0      0      0 S   0.0   0.0   0:00.13 migration/1
```

```
  0[||||||||||                                                  14.3%] Tasks: 92, 293 thr, 79 kthr; 1 running
  1[||||||||||||                                                16.1%] Load average: 0.30 0.17 0.12
Mem[||||||||||||||||||||||||||||||||||||||||||||||||||||||||1.01G/1.93G] Uptime: 00:46:48
Swp[|||||                                                     46.4M/975M]

  Main   I/O
    PID USER      PRI  NI  VIRT   RES   SHR S  CPU%▽MEM%   TIME+  Command
    795 root       20   0  443M 99504 39744 S  19.9  4.9  1:01.76 /usr/lib/xorg/Xorg :0 -seat seat0 -auth /var/run/lightdm/root/:0
  14115 kali       20   0  469M 46652 35028 S   6.2  2.3  0:00.50 xfce4-screenshooter
   1036 kali       20   0  911M 42936 24072 S   4.1  2.1  0:16.09 xfwm4
    802 root       20   0  443M 99504 39744 S   1.4  4.9  0:03.29 /usr/lib/xorg/Xorg :0 -seat seat0 -auth /var/run/lightdm/root/:0
    926 kali       -6   0  893M 16740  6068 S   1.4  0.8  0:13.69 /usr/bin/pulseaudio --daemonize=no --log-target=journal
  13955 kali       20   0 9240  5136  3532 R   1.4  0.3  0:00.78 htop
   1083 kali       20   0  200M 22836  9392 S   0.7  1.1  0:20.57 /usr/lib/x86_64-linux-gnu/xfce4/panel/wrapper-2.0 /usr/lib/x86_6
   1086 kali       20   0  406M 21100 11204 S   0.7  1.0  0:08.79 /usr/lib/x86_64-linux-gnu/xfce4/panel/wrapper-2.0 /usr/lib/x86_6
```