Solaris Containers. Управление ресурсами.

Юрий Кажаров

В статье описываются возможности механизмов управления ресурсами системы с использованием технологии Resource Management, которая является частью инновационной технологии Solaris Containers, появившейся в Sun Solaris 10.

Введение

Управление ресурсами является важным механизмом в администрировании любой системы и непосредственно влияет на эффективность использования аппаратных средств и качество предоставления сервиса.

Использование технологии управления ресурсами позволяет:

- Распределять аппаратные ресурсы системы между различными задачами
- Контролировать эффективность работы процессов и динамически выделять им необходимые ресурсы
- Вести статистику работы процессов

Технология управления ресурсами появилась для операционной системы Solaris несколько лет назад в виде самостоятельных коммерческих продуктов: Solaris Resource Manager, который использовался для контроля над системными ресурсами (процессор, память, вход в систему), и Solaris Bandwidth Manager, в задачи которого входил контроль за сетевым трафиком. Частично функциональные возможности данных продуктов интегрировались в операционную систему Sun Solaris (начиная с Solaris 8) и от версии к версии их список постоянно расширялся. Так, например, в Solaris 10 было добавлено 15 новых параметров для управления ресурсами.

В Solaris 10 данные механизмы объединены в единую инновационную технологию Solaris Containers. Основной идеей данной технологии является полная изоляция всех процессов, функционирующих в системе, что существенно повышает надежность и безопасность ее работы.

Технология Solaris Containers состоит из двух основных компонентов: технологии управления ресурсами - Resource Management и технологии виртуальных зон - Solaris Zones, с помощью которой можно создавать независимые виртуальные копии операционной системы (подробно данная технология описывается в статье «Solaris Containers. Конфигурирование зон.»)

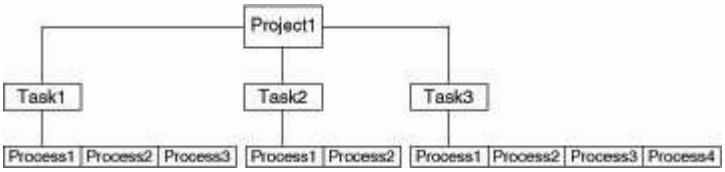
Resource Management использует в своем арсенале ряд алгоритмов, позволяющих наиболее эффективно распределять ресурсы системы и предотвращать возможные критические ситуации, возникающие по причине их неконтролируемого использования.

Контроль над ресурсами осуществляется по трем основным механизмам:

- **Scheduling.** Данный механизм позволяет распределять ресурсы небольшими предустановленными фрагментами между всеми процессами в заданном интервале времени. Если какому-либо процессу ресурсы в данный момент не требуются, то он освобождает их для других процессов.
- **Partitioning.** В данном случае ресурсы разбиваются на предопределенные фрагменты и выставляются рабочей области. Механизм обеспечивает гарантированное предоставление процессам необходимых ресурсов.
- **Constraint.** Данный механизм позволяет очерчивать границы потребления ресурсов для области выполнения задач.

Описание технологии

Для того чтобы четко определить функциональность используемых рабочих областей в операционной системе Solaris их гипотетически разделили на два основных типа: project(проекты) и task(задачи). Проекты представляют собой совокупность распределенных по сети задач, а сами задачи включают в себя рабочие процессы. Данная структура наглядно представлена на следующем рисунке:



Как видно из рисунка, проекты являются вершиной данного дерева. По своей сути их можно сопоставить с пользователями и группами пользователей, зарегистрированными в системе. Любой пользователь при входе в систему обязательно прикрепляется к определенному для него проекту либо проекту по умолчанию. Пользователь, также, может ассоциироваться и с несколькими проектами одновременно.

Система определяет к какому проекту прикреплен пользователь в момент его входа в нее по следующему алгоритму:

1. Просматривается файл */etc/user_attr* (база расширенных атрибутов пользователей) и если этот пользователь в нем присутствует, то использует для него объявленный в этом файле параметр project
2. Просматривается файл */etc/project* и если в нем объявлен проект для user.*user_id*, то система использует этот проект в качестве «default» для данного пользователя
3. Просматривается файл */etc/project* и если в нем объявлен проект для group.*group-name*, то система использует этот проект в качестве «default» для данного пользователя, если эта группа является главной(primary) для данного пользователя
4. Если прописан проект default в файле */etc/project,* то он назначается проектом по умолчанию для всех пользователей

Локальная база проектов, как вы уже очевидно догадались, находится в */etc/project*. Ее можно делегировать в различные службы имен: NIS, NIS+, LDAP.

Формат этого файла довольно прост:

projname:projid:comment:user-list:group-list:attributes

где

projname – имя проекта
projid – уникальный идентификатор проекта
comment – описание проекта (не обязательное поле)
user-list – список пользователей, использующих данный проект
group-list – список групп пользователей, использующих данный проект
attributes – параметры, определяющие возможность использования ресурсов

Пример содержимого файла /etc/project:

```
# cat /etc/project
system:0::::
user.root:1::::
noproject:2::::
default:3::::
group.staff:10::::
```

При успешном входе в систему проект в рамках процедуры входа создает новую задачу, которая включает в себя все процессы, ассоциированные с ней (см. рисунок выше). Задачи автоматически получают свой уникальный ID и однозначно привязываются только к одному проекту.

Задачи создаются при следующих действиях/командах:

- login
- cron
- su
- newtask
- setproject

Рассмотрим теперь, как можно на практике создавать, управлять и анализировать работу с проектами и задачами:

1. Используя команды id и ps можно определить с какими проектами ассоциирован пользователь:

```
bash-2.05b$ id -p
uid=101(urix) gid=1(other) projid=3(default)

bash-2.05b$ ps -o user,pid,uid,projid
    USER   PID   UID PROJID
    urix  2311   101      3
    urix  2369   101      3
```

и для пользователя root:

```
# id -p
uid=0(root) gid=1(other) projid=1(user.root)

# ps -o user,pid,uid,projid
    USER   PID   UID PROJID
    root  2203     0      1
    root  2352     0      1
```

2. С помощью команды prstat можно динамически просматривать и процессы, работающие в системе, и проекты (опция -J):

```
# prstat -J

   PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU PROCESS/NLWP
  2185 urix       61M   10M sleep   59    0   0:00:01 0.5% gnome-terminal/1
  2397 root     6136K 5072K cpu1    59    0   0:00:00 0.1% prstat/1
  2169 urix       56M 8440K sleep   59    0   0:00:00 0.1% metacity/1
  2392 urix     6136K 5080K sleep   59    0   0:00:00 0.0% prstat/1
  2175 urix       62M   12M sleep   59    0   0:00:01 0.0% gnome-panel/1
  2195 urix       60M   10M sleep   59    0   0:00:00 0.0% mixer_applet2/1
  2127 urix     2464K 1040K sleep   59    0   0:00:00 0.0% dsdm/1
  2166 urix     3616K 2544K sleep   59    0   0:00:00 0.0% gnome-smproxy/1
  2189 urix       54M 6128K sleep   59    0   0:00:00 0.0% galf-server/1
   611 root     3656K 2272K sleep   59    0   0:00:00 0.0% snmpXdmid/2
   571 root     2216K 1056K sleep   59    0   0:00:00 0.0% rpc.bootparamd/1
   462 root     1880K  536K sleep   59    0   0:00:00 0.0% smcboot/1
   463 root     1872K  352K sleep   59    0   0:00:00 0.0% smcboot/1
   464 root     1872K  368K sleep   59    0   0:00:00 0.0% smcboot/1
   499 root     5360K 2024K sleep   59    0   0:00:00 0.0% htt_server/2
   788 root     2088K 1328K sleep   59    0   0:00:00 0.0% init/1
   410 root     2544K  592K sleep   59    0   0:00:00 0.0% cron/1
   423 root     2968K 2128K sleep   59    0   0:00:00 0.0% nscd/24
   406 root     3944K 1680K sleep   59    0   0:00:00 0.0% syslogd/15
   407 root     3840K 1664K sleep   59    0   0:00:00 0.0% automountd/2
   491 root     3752K 1496K sleep   59    0   0:00:00 0.0% utskid/1
   328 root       16M 7224K sleep   59    0   0:00:00 0.0% ns-httpd/59
   326 root       12M 6720K sleep   59    0   0:00:00 0.0% ns-httpd/2
   440 root     1576K  648K sleep   59    0   0:00:00 0.0% powerd/3
   325 root     4376K 1184K sleep   59    0   0:00:00 0.0% uxwdog/1
   424 root     2384K  816K sleep   59    0   0:00:00 0.0% inetd/1
   315 nobody     75M   63M sleep   59    0   0:00:00 0.0% ns-slapd/40
PROJID    NPROC  SIZE   RSS MEMORY      TIME  CPU PROJECT
     3       21  485M  120M   6.0%   0:00:02 0.7% default
     1        2 8960K 7200K   0.4%   0:00:00 0.1% user.root
     0       90  417M  234M    12%   0:00:04 0.0% system

Total: 113 processes, 373 lwps, load averages: 0.02, 0.02, 0.03
```

и задачи (опция -T):

```
# prstat -T

   PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU PROCESS/NLWP
  2185 urix       61M   11M sleep   59    0   0:00:02 0.3% gnome-terminal/1
  2424 root     6136K 5056K cpu0    59    0   0:00:00 0.1% prstat/1
  2173 urix       71M   21M sleep   59    0   0:00:01 0.1% nautilus/8
  2175 urix       62M   12M sleep   59    0   0:00:01 0.0% gnome-panel/1
  2169 urix       56M 8440K sleep   59    0   0:00:00 0.0% metacity/1
  2195 urix       60M   10M sleep   59    0   0:00:00 0.0% mixer_applet2/1
  2203 root     2824K 2128K sleep   59    0   0:00:00 0.0% bash/1
  2189 urix       54M 6128K sleep   59    0   0:00:00 0.0% galf-server/1
   611 root     3656K 2272K sleep   59    0   0:00:00 0.0% snmpXdmid/2
   571 root     2216K 1056K sleep   59    0   0:00:00 0.0% rpc.bootparamd/1
   462 root     1880K  536K sleep   59    0   0:00:00 0.0% smcboot/1
   463 root     1872K  352K sleep   59    0   0:00:00 0.0% smcboot/1
   464 root     1872K  368K sleep   59    0   0:00:00 0.0% smcboot/1
   499 root     5360K 2024K sleep   59    0   0:00:00 0.0% htt_server/2
   788 root     2088K 1328K sleep   59    0   0:00:00 0.0% init/1
   410 root     2544K  592K sleep   59    0   0:00:00 0.0% cron/1
   423 root     2968K 2128K sleep   59    0   0:00:00 0.0% nscd/24
   406 root     3944K 1680K sleep   59    0   0:00:00 0.0% syslogd/15
   407 root     3840K 1664K sleep   59    0   0:00:00 0.0% automountd/2
   491 root     3752K 1496K sleep   59    0   0:00:00 0.0% utskid/1
   328 root       16M 7224K sleep   59    0   0:00:00 0.0% ns-httpd/59
   326 root       12M 6720K sleep   59    0   0:00:00 0.0% ns-httpd/2
   440 root     1576K  648K sleep   59    0   0:00:00 0.0% powerd/3
   325 root     4376K 1184K sleep   59    0   0:00:00 0.0% uxwdog/1
   424 root     2384K  816K sleep   59    0   0:00:00 0.0% inetd/1
   315 nobody     75M   63M sleep   59    0   0:00:00 0.0% ns-slapd/40
   292 daemon   2424K  760K sleep   59    0   0:00:00 0.0% rpcbind/1
TASKID    NPROC  SIZE   RSS MEMORY      TIME  CPU PROJECT
     6       20  479M  115M   5.7%   0:00:04 0.5% default
     7        2 8960K 7184K   0.4%   0:00:00 0.1% user.root
     2        1    0K    0K   0.0%   0:00:00 0.0% system
     3       33  112M   63M   3.1%   0:00:00 0.0% system
     1       56  305M  171M   8.5%   0:00:04 0.0% system


Total: 112 processes, 371 lwps, load averages: 0.01, 0.02, 0.03
```

3. Создадим новый проект с именем test и идентификатором 150 для пользователя urix:

```
# projadd -U urix -p 150 test
```

4. Добавим комментарии для вновь созданного проекта:

```
# projmod -c "Test project for user urix" test

# cat /etc/project
system:0::::
user.root:1::::
noproject:2::::
default:3::::
group.staff:10::::
test:150:Test project for user urix:urix::
```

5. Войдем под пользователем, используя команду su, и проверим, с какими проектами он ассоциирован:

```
# su - urix
Sun Microsystems Inc.   SunOS 5.10  s10_55      May 2004
bash-2.05b$ projects
default test
```

6. Теперь создадим новую задачу в проекте test. Система автоматически перенесет пользовательский shell-процесс в вновь созданную задачу:

```
bash-2.05b$ newtask -v -p test
9
bash-2.05b$ id -p
uid=101(urix) gid=1(other) projid=150(test)
```

7. Выйдем из только что созданной задачи:

```
bash-2.05b$ exit
exit
bash-2.05b$ id -p
uid=101(urix) gid=1(other) projid=3(default)
```

Управление ресурсами

Традиционно все UNIX системы имеют в своем арсенале механизмы наложения лимитов на ряд ресурсов. Администраторы системы применяют их для ограничения пользовательским процессам использования таких ресурсов как: время доступа к CPU, количество открытых файлов, размер core-файла, объем памяти для сегмента данных процесса и т.д.

В операционной системе Sun Solaris концепция распределения ресурсов системы разделяется по уровням: проектам, задачам и процессам. Механизмы управления ресурсами базируются на специальных параметрах «тонкой» настройки (tunable parameters) ядра системы.

Эти параметры предназначены для оптимальной настройки операционной системы для решения конкретных задач, поставленных перед системой в целом. Прописываются они в конфигурационном файле ядра - */etc/system*. (Подробно про тюнинг системы можно прочитать в «Solaris Tunable Parameters Reference Manual»).

Технология управления ресурсами, к сожалению, на сегодняшний день позволяет управлять только частью параметрами «тонкой» настройки ядра системы. Но, как я уже отмечал выше, от версии к версии количество поддерживаемых параметров увеличивается. Параметры, использующиеся в механизмах управления ресурсами, переопределяют значения, установленные ранее в файле */etc/system*.

Механизмы распределения ресурсов настраиваются через базу проектов - */etc/project*.

Приведу несколько стандартных параметров для управления ресурсами:

Имя параметра	Описание	Значение
project.cpu-shares	Гарантированное количество частей-интервалов использования процессора	количество
project.max-shm-ids	Максимальное количество частей памяти доступных проекту	количество
project.max-sem-ids	Максимальное количество семафоров для данного проекта	количество
project.max-mes-ids	Максимальная очередь сообщений для данного проекта	количество
project.max-shm-memory	Общее количество памяти, выделенное для данного проекта	размер (байты)
project.max-lwps	Максимальное количество LWP одновременно возможных в данном проекте	количество
project.max-tasks	Максимальное количество задач в данном проекте	количество

task.max-cpu-time	Максимальное время использования процессора для данной задачи	время (секунды)
task.max-lwps	Максимальное количество LWP одновременно возможных в данной задаче	количество
process.max-cpu-time	Максимальное время использования процессора для данного процесса	время (секунды)
process.max-file-descriptor	Максимальный номер файлового дескриптора доступного для данного процесса	индекс
process.max-file-size	Максимальный размер файла, который может создавать данный процесс	размер (байты)
process.max-core-size	Максимальный размер core-файла, который может создавать данный процесс	размер (байты)
process.max-data-size	Максимальный размер памяти, выделяемой для области данных данного процесса	размер (байты)
process.max-stack-size	Максимальный размер сегмента памяти для стека, который выделяется для данного процесса	размер (байты)

Политика назначения параметров для управления ресурсами подразделяется на три основных типа:

- basic (базовый) – может изменяться владельцем процесса
- privileged (привилегированный) – изменяется только администраторами
- system (системный) – устанавливается на операционную систему

Гарантированно для системы назначается один параметр системного типа, который и определяет величину данного ресурса в операционной системе. Привилегированных значений может быть установлено несколько для данного параметра, а вот базовых значений – только одно. Если привилегированные значения не установлены, то используются базовые, определенные по умолчанию.

При превышении значений, установленных в параметрах, механизмы управления ресурсами выполняют определенные действия (actions). Эти действия подразделяются на глобальные и локальные. Глобальные действия применяются ко всем контролируемым ресурсам системы, а локальные только к конкретному процессу, который достиг установленного лимита.

Вы можете одновременно устанавливать несколько действий для одного параметра ограничения ресурса. Существует три типа локальных действий:

- none – при превышении лимита, установленного для данного параметра, никаких действий не выполняется. Это действие используют, в основном, при мониторинге использования ресурсов системы.
- deny – запрещает использовать ресурсы, превышающие установленный лимит.
- signal= - при превышении лимита посылает сигнал данному процессу. Для технологии управления ресурсами доступны следующие сигналы:
    - SIGABRT – прерывает процесс
    - SIGHUP – сигнал разрыва линии
    - SIGTERM – прерывает процесс
    - SIGKILL – «жесткое» прерывание работы процесса
    - SIGSTOP – останавливает процесс
    - SIGXRES – сигнал превышения установленного лимита
    - SIGXPSZ – завершает процесс по причине превышения лимита по размеру файла
    - SIGXCPU – завершает процесс по причине превышения временного лимита использования CPU

Устанавливать лимиты вы можете либо с помощью ряда функций командной строки, либо с помощью Solaris Management Console (*/usr/sbin/smc*), либо напрямую редактируя */etc/project.*

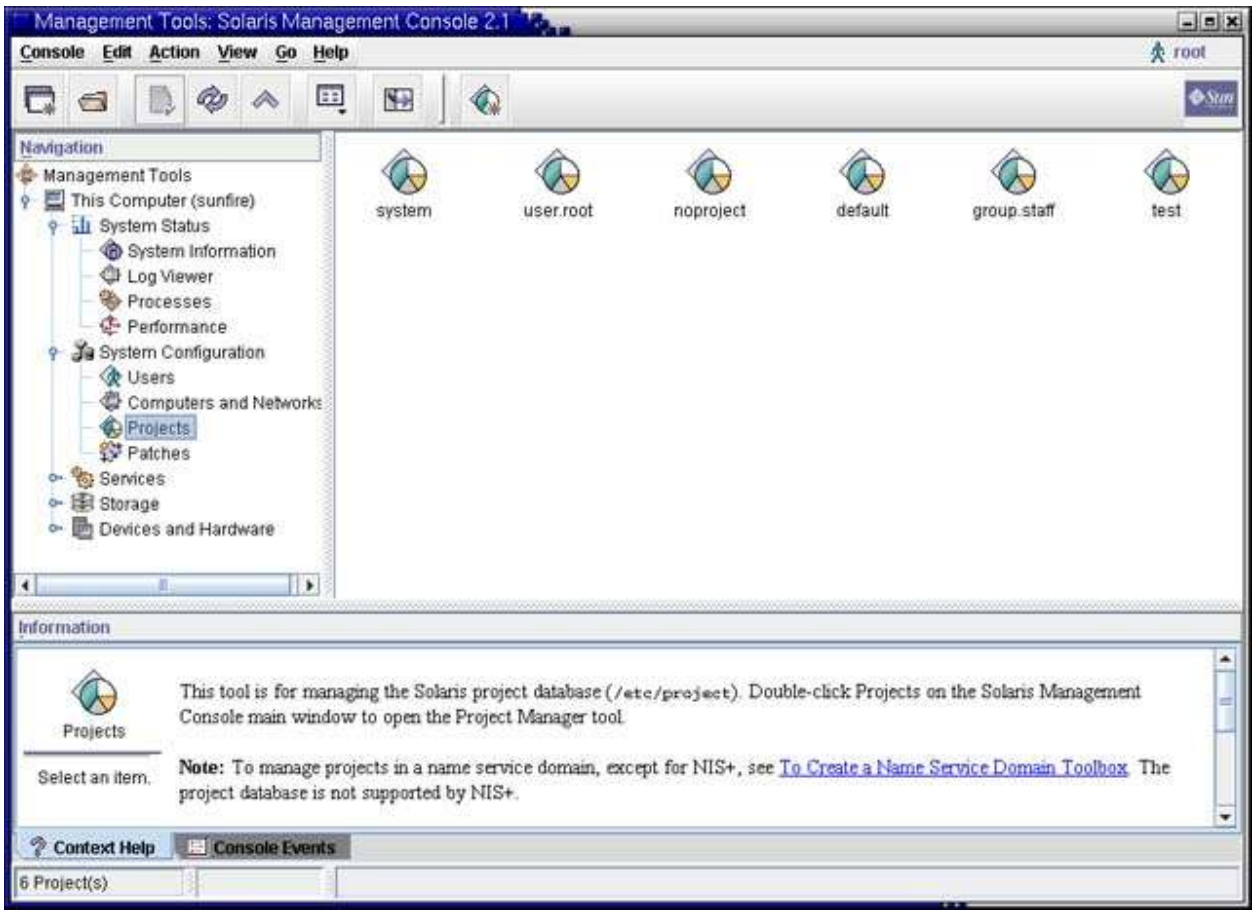


Рис.2 Управление ресурсами в Solaris Management Console

Рассмотрим теперь практическое применение описанных механизмов:

1. Добавим к нашему проекту **test** параметр, ограничивающий количество выполняющихся процессов (task.max-lwps):

```
# projadd -K 'task.max-lwps=(privileged,3,deny)' test

# cat /etc/project
system:0::::
user.root:1::::
noproject:2::::
default:3::::
group.staff:10::::
test:150:Test project for user urix:urix::task.max-lwps=(privileged,3,deny)
```

2. Создадим новую задачу с шеллом *csh;* с помощью команды *prctl* посмотрим установленные значения для параметра task.max-lwps и попробуем «расплодить» несколько процессов:

```
# newtask -p test csh

sunfire# prctl -n task.max-lwps $$
1925: csh
task.max-lwps
                        3 privileged deny
                2147483647 system     deny           [ max ]

sunfire# id -p
uid=0(root) gid=1(other) projid=150(test)

sunfire# ps -o project,taskid -p $$
 PROJECT TASKID
    test     7

sunfire# csh
sunfire# csh
sunfire# csh
Vfork failed
sunfire#
```

Мы поставили ограничение до 3 процессов и как только попытались запустить четвертый процесс *csh* – система нам в этом отказала!

3. Добавим теперь дополнительно еще несколько ограничивающих параметров в наш проект:

```
# projmod -s -K 'task.max-lwps=(basic,10,none),(privileged,500,deny);
process.max-file-descriptor=(basic,128,deny)' test

# cat /etc/project
system:0::::
user.root:1::::
noproject:2::::
default:3::::
group.staff:10::::
test:150:Test project for user urix:urix::task.max-lwps=(basic,10,none),
(privileged,500,deny);process.max-file-descriptor=(basic,128,deny)
```

посмотрим на значения, установленные в системе по умолчанию:

```
# prctl -n process.max-file-descriptor $$
2061: bash
process.max-file-descriptor              [ lowerable deny ]
                      256 basic      deny
                    65536 privileged deny
                2147483647 system     deny           [ max ]
#
# prctl -n task.max-lwps $$
2061: bash
task.max-lwps
                2147483647 system     deny           [ max ]
#
```

создадим новую задачу и посмотрим, изменились или нет значения параметров:

```
# newtask -p test csh

sunfire# prctl -n process.max-file-descriptor $$
2283: csh
process.max-file-descriptor              [ lowerable deny ]
                       128 basic      deny
                2147483647 system     deny           [ max ]

sunfire# prctl -n task.max-lwps $$
2283: csh
task.max-lwps
                        10 basic      none
                       500 privileged deny
                2147483647 system     deny           [ max ]
sunfire#
```

как видно на примере, все значения, которые мы задавали для данного проекта, успешно установились.

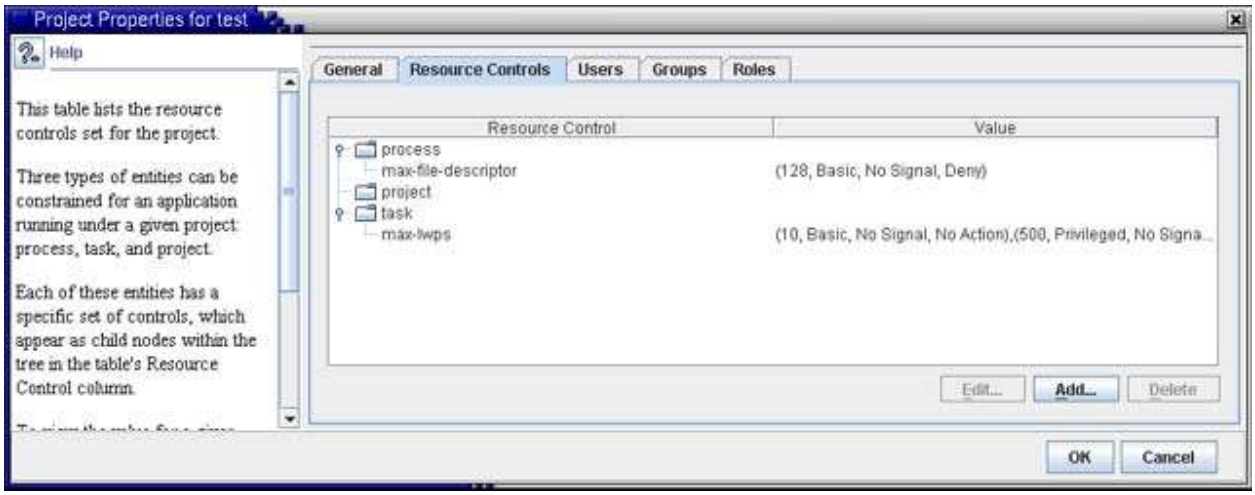


Рис.3 Установленные параметры для проекта test

4. Используя команду *prctl*, можно «в горячую» изменять значения параметров. Посмотрим, вначале, какие значения у нас были установлены:

```
sunfire# prctl -n task.max-lwps $$
2676: csh
task.max-lwps
                        10 basic      none
                       500 privileged deny
                2147483647 system     deny           [ max ]

sunfire# prctl -n process.max-file-descriptor $$
2676: csh
process.max-file-descriptor              [ lowerable deny ]
                       128 basic      deny
                2147483647 system     deny           [ max ]
sunfire#
```

теперь наложим новые значения и проверим результаты:

```
sunfire# prctl -n project.max-lwps -t privileged -v 3 -e deny -i project test

sunfire# prctl -n task.max-lwps $$
2676: csh
task.max-lwps
                        3 privileged deny
                       10 basic      none
                      500 privileged deny
                2147483647 system     deny           [ max ]


sunfire# prctl -n process.max-file-descriptor -r -v 256 $$

sunfire# prctl -n process.max-file-descriptor $$
2676: csh
process.max-file-descriptor              [ lowerable deny ]
                      256 basic      deny
                2147483647 system     deny           [ max ]
sunfire#
sunfire# exit
#
```

5. Вы можете «в горячую» изменять состояние глобальных параметров с помощью команды *rctladm*:

```
# rctladm
process.max-port-events     syslog=off     [ deny ]
process.crypto-buffer-limit syslog=off     [ deny ]
process.max-crypto-sessions syslog=off     [ deny ]
process.add-crypto-sessions syslog=off     [ deny ]
process.min-crypto-sessions syslog=off     [ deny ]
process.max-msg-messages    syslog=off     [ deny ]
process.max-msg-qbytes      syslog=off     [ deny ]
process.max-sem-ops         syslog=off     [ deny ]
process.max-sem-nsems       syslog=off     [ deny ]
process.max-address-space   syslog=off     [ lowerable deny no-local-action ]
process.max-file-descriptor syslog=off     [ lowerable deny ]
process.max-core-size       syslog=off     [ lowerable deny no-local-action ]
process.max-stack-size      syslog=off     [ lowerable deny no-local-action ]
process.max-data-size       syslog=off     [ lowerable deny no-local-action ]
process.max-file-size       syslog=off     [ lowerable deny file-size ]
process.max-cpu-time        syslog=off     [ lowerable no-deny cpu-time inf ]
task.max-cpu-time           syslog=off     [ no-deny cpu-time no-obs inf ]
task.max-lwps               syslog=off
project.max-device-locked-memory syslog=off     [ no-basic deny ]
project.max-port-ids        syslog=off     [ no-basic deny ]
project.max-shm-memory      syslog=off     [ no-basic deny ]
project.max-shm-ids         syslog=off     [ no-basic deny ]
project.max-msg-ids         syslog=off     [ no-basic deny ]
project.max-sem-ids         syslog=off     [ no-basic deny ]
project.cpu-shares          syslog=off     [ no-basic no-local-action ]
zone.cpu-shares             syslog=off     [ no-basic no-local-action ]
#
```

добавим теперь возможность отслеживать через *syslog-сервис* сообщения по параметру task.max-lwps:

```
# rctladm -e syslog task.max-lwps

# rctladm
.
task.max-lwps               syslog=notice
.
#
```

6. С помощью Solaris Management Console возможно управлять ресурсами системы. Зададим ограничение по использованию времени работы процессора для проекта test:

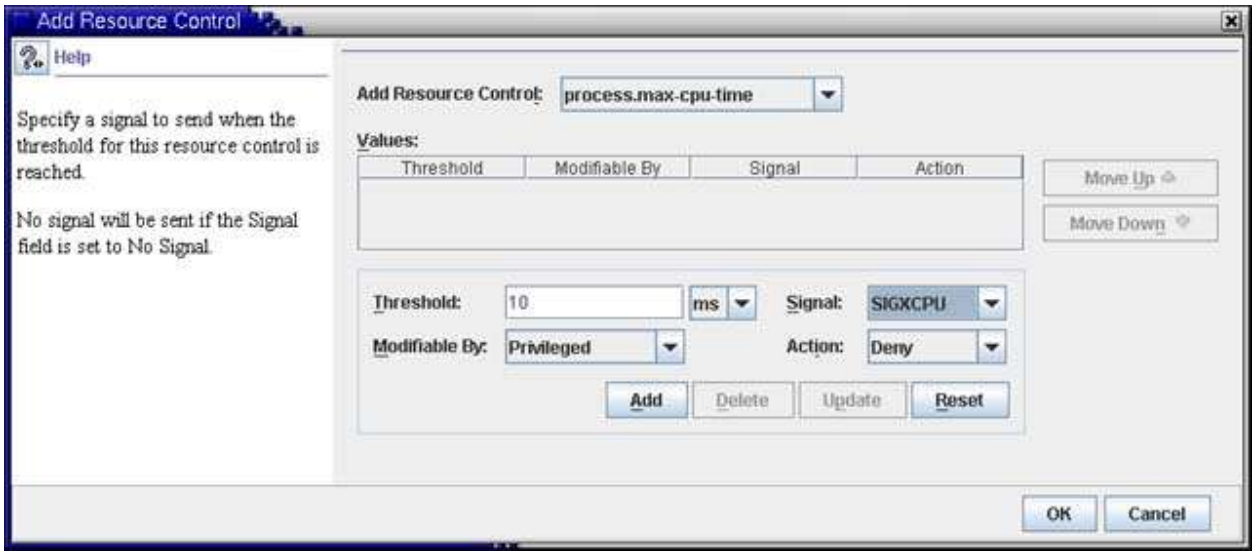


Рис.4 Установка параметра в SMC

Теперь посмотрим на изменения в файле /etc/project и на список установленных параметров для проекта test:

```
# cat /etc/project
system:0::::
user.root:1::::
noproject:2::::
default:3::::
group.staff:10::::
test:150:Test project for user urix:urix::process.max-cpu-time=(privileged,
10ms, signal=SIGXCPU, deny);process.max-file-descriptor=(basic, 128, deny);
task.max-lwps=(basic, 10, none),(privileged, 500, deny)
```

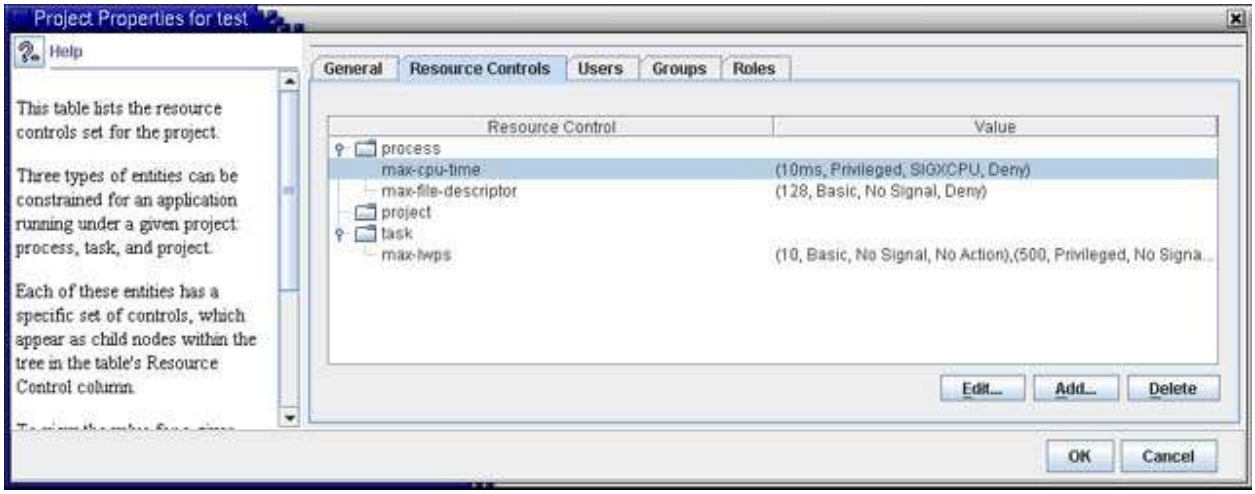


Рис.5 Установленные параметры для проекта test

Управление ресурсами CPU. Fair Share Scheduler (FSS)

При распределении ресурсов процессора, в Sun Solaris, по умолчанию используется механизм временного разделения (TS - timesharing scheduling). Согласно этому механизму, система пытается распределить ресурсы процессора приблизительно равными долями

между всеми запущенными процессами. С использованием ряда параметров *. max-cpu-time можно устанавливать время использования процессора для необходимых задач и процессов. Но время использования процессора не позволяет четко задать, в процентном отношении, ресурс его использования.

Если перед вами стоит задача гарантированного выделения необходимой части вычислительной мощности процессоров, то в этом случае, применяется технология долевого распределения (FSS - Fair Share Scheduler). С помощью механизмов FSS можно выдавать определенное количество частей (shares) использования процессорных ресурсов системы для необходимых вам проектов.

Части, распределяемые между проектами в FSS, не равносильны процентам использования процессорных ресурсов системы. Они определяют долю использования ресурса между различными проектами, и важным критерием является не количество выделенных частей, а их отношение по сравнению с другими проектами.

Проекты в системе могут находиться в одном из двух состояний: неработающий (idle) либо активный (active). Первое означает, что ни один процесс в данном проекте не использует ресурсов процессора. Обычно, это происходит, когда процессы находятся в «спящем» состоянии или остановлены. Активным проект становится в случае, когда хоть один из его процессов использует ресурсы процессора.

Для того чтобы подсчитать в численном эквиваленте количество используемых ресурсов процессора для данного проекта, нужно воспользоваться довольно простой формулой:

$$\text{allocation}_{\text{project}^i} = \frac{\text{shares}_{\text{project}^i}}{\sum_{j=1\ldots n} (\text{shares}_{\text{project}^j})}$$

n – количество активных проектов

В тех случаях, когда в вашей системе установлено более 2 процессоров, их можно разделять на процессорные группы (processor sets). В этом случае формула для расчета примет следующий вид:

$$\text{allocation}_{\text{project}_x^i} = \frac{\text{shares}_{\text{project}_x^i}}{\sum_{\text{processor set } X,\ j=1\ldots n} (\text{shares}_{\text{project}^j})}$$

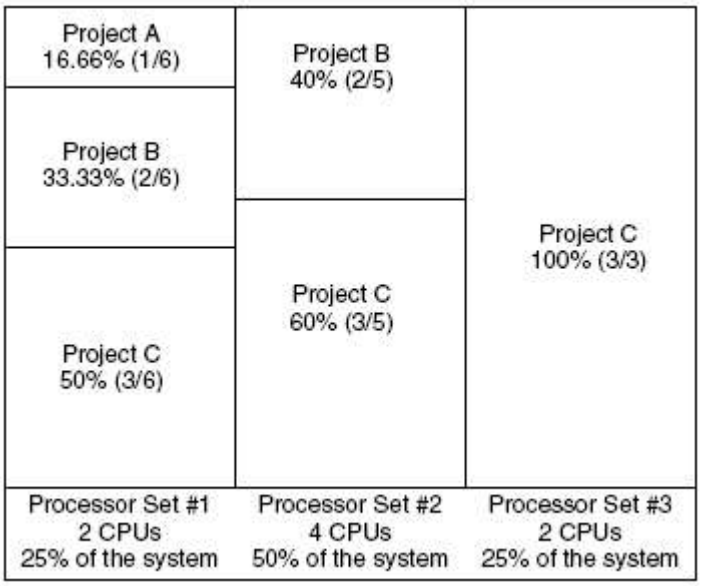
n – количество активных проектов, а X – номер процессорной группы

Рассмотрим теперь пример использования данных формул. Пусть в нашей системе функционируют три активных проекта: A, B, C. Для начала проведем расчет для системы без назначенных процессорных групп. Выделим проекту A - одну часть ресурсов, B – две части, C – три части. В сумме получаем 1+2+3=6. Таким образом, получаем следующий результат:

Project	Allocation
Project A	16.66% = (1/6)
Project B	33.33% = (2/6)
Project C	50% = (3/6)

Если в нашей системе, например, восемь процессоров, то можно их разделить на группы (как это сделать описывается ниже в разделе «Пулы ресурсов»). Создадим три процессорные группы: 2cpu+4cpu+2cpu. В первой группе функционируют все три проекта из нашего первого примера. Во второй группе: проект B- с двумя частями ресурсов и проект C - с тремя частями. В третьей группе – только проект C – с тремя частями.

Функционально это будет выглядеть следующим образом:



и расчет, в этом случае, дает следующие значения:

Project	Allocation
Project A	$4\% = (1/6 \times 2/8)_{pset1}$
Project B	$28\% = (2/6 \times 2/8)_{pset1} + (2/5 * 4/8)_{pset2}$
Project C	$67\% = (3/6 \times 2/8)_{pset1} + (3/5 \times 4/8)_{pset2} + (3/3 \times 2/8)_{pset3}$

Перейдем теперь от теории к практической реализации механизма FSS.

1. Для начала, используя команду *dispadmin*, посмотрим, какие классы в нашей системе установлены:

```
# dispadmin -l
CONFIGURED CLASSES
==================
SYS    (System Class)
TS     (Time Sharing)
FX     (Fixed Priority)
IA     (Interactive)
RT     (Real Time)
```

2. Установим в нашей системе возможность использования класса FSS. Это действие вступит в «полную силу» при следующей перезагрузке системы:

```
# dispadmin -d FSS

# dispadmin -l
CONFIGURED CLASSES
==================
SYS    (System Class)
TS     (Time Sharing)
FX     (Fixed Priority)
IA     (Interactive)
RT     (Real Time)
FSS    (Fair Share)
```

3. Понятно, что мы не хотим ждать, когда наша система будет в очередной раз «перезагружена», чтобы процессы могли использовать механизм FSS. Для этого воспользуемся командой *priocntl,* с помощью которой можно «в горячую» изменять класс для процессов. Перенесем процесс init (pid 1) в FSS класс, и посмотрим, появились или нет процессы в этом классе:

```
# priocntl -s -c FSS -i pid 1

# ps -ef -o pset,class | grep -v CLS | sort | uniq
  -   FX
  -   TS
  -  FSS
  -  SYS
```

Можно переносить процессы и проекты из одного класса в другой, но необходимо помнить, что при следующей перезагрузке системы все они будут запущены в своем предопределенном классе:

```
# priocntl -s -c FSS -i class TS
# priocntl -s -c FSS -i projid 150

# ps -ef -o pset,class | grep -v CLS | sort | uniq
  -   FX
  -  FSS
  -  SYS
```

Как видно на примере, после выполнения первой команды, в системе не осталось процессов, функционирующих в классе TS.

4. Назначить часть процессорного ресурса для проекта можно с помощью параметра *project.cpu-shares*:

```
# prctl -n project.cpu-shares -r -v 5 -i project test

# cat /etc/project
system:0::::
user.root:1::::
noproject:2::::
default:3::::
group.staff:10::::
test:150:Test project for user urix:urix::process.max-cpu-time=(privileged,
10ms, signal=SIGXCPU, deny);process.max-file-descriptor=(basic, 128, deny);
task.max-lwps=(basic, 10, none),(privileged, 500, deny);project.cpu-shares=
(privileged,5,none)
```

Пулы ресурсов (Resource pools)

Технология Resource management позволяет создавать пулы ресурсов. Основная задача пулов состоит в предоставлении рабочим областям необходимых ресурсов системы. Эта возможность позволяет задавать гарантированную производительность системы для решения поставленных задач.

С помощью пулов можно создавать процессорные группы и назначать классы FSS. По умолчанию система устанавливает пул «default pool» и одну процессорную группу - «default processor set». Эти параметры не могут быть удалены.

Использовать пулы удобно в следующих случаях:

- Когда необходимо разделить систему на две части. Одна часть используется для пользователей и их процессов, а другая для системных ресурсов
- Когда на сервере функционируют несколько критически важных задач. Таким образом, с помощью пулов можно разделить ресурсы системы между задачами
- Когда необходимо гарантированно выделять отдельным пользователям системы определенные ресурсы
- Для работы real-time приложений

При наложении пула на проект необходимо установить для него атрибут *project.pool* в файле */etc/project* либо выполнить команду:

```
# projmod -s -K project.pool=pool_test test
```

Каждый проект может быть ассоциированным только с одним пулом.

Для конфигурирования пулов используется статическая таблица параметров */etc/pooladm.conf*. По умолчанию, в системе, механизм использования пулов отключен и данного файла нет. При включении механизма пулов этот файл можно создать с установленными параметрами «по умолчанию». Если в момент загрузки операционная система «видит» этот файл, то она активирует автоматически механизм пулов. Формат таблицы представляет собой стандартный XML и не редактируется «в ручную». Все изменения, которые вы осуществляете, накладываются с помощью команды *poolcfg*. Можно, также, использовать batch-файл - заготовку в виде текстового файла, с прописанными в нем параметрами.

Используя различные статические таблицы отличные от */etc/pooladm.conf*, и с помощью механизма запуска приложения по расписанию (cron) вы можете изменять политику распределения ресурсов вашей системы, например: дневную, ночную, выходного дня и т.д.

1. Активируем механизм пулов (опция –e активирует механизм пулов, а –d отключает его):

```
# pooladm -e
```

2. Создадим статическую таблицу */etc/pooladm.conf* с параметрами «по умолчанию» и посмотрим на эти параметры:

```
# pooladm -s

# poolcfg -c info

system sunfire
      string      system.comment
      int         system.version 1
      boolean     system.bind-default true
      int         system.poold.pid 1606

      pool pool_default
            int         pool.sys_id 0
            boolean     pool.active true
            boolean     pool.default true
            int         pool.importance 1
            string      pool.comment
            pset        pset_default

      pset pset_default
            int         pset.sys_id -1
            boolean     pset.default true
            uint        pset.min 1
            uint        pset.max 65536
            string      pset.units population
            uint        pset.load 9
            uint        pset.size 2
            string      pset.comment

            cpu
                  int         cpu.sys_id 1
                  string      cpu.comment
                  string      cpu.status on-line

            cpu
                  int         cpu.sys_id 0
                  string      cpu.comment
                  string      cpu.status on-line
```

Для того чтобы сохранить активные значения работающего механизма пулов (динамическую таблицу) в альтернативную статическую таблицу, необходимо выполнить команду:

```
# pooladm -s /путь/имя_таблицы
```

Активировать альтернативную таблицу можно с помощью команды:

```
# pooladm -c /путь/имя_таблицы
```

Ряд параметров, используемых при конфигурировании механизма пулов ресурсов, представлены в следующей таблице:

Имя параметра	Тип	Описание
system.poold.log-level	string	Уровень логов (по умолчанию **NOTICE**)
system.poold.log-location	string	Расположение логов (по умолчанию /var/log/pool/poold)
system.poold.monitor-interval	unsigned int	Мониторинг использования ресурсов (миллисекунды)
system.poold.pid	int	PID для poold
pset.max	int	Максимальное количество процессоров в данной процессорной группе. Процессор одновременно может быть привязан только к одной группе

pset.min	int	Минимальное количество процессоров в данной процессорной группе. Не может быть выше количества установленных в вашем сервере процессоров
cpu.pinned	boolean	Указывает, что данный процессор не может быть динамически перенесен в другую процессорную группу
pool.importance	unsigned int	Значимость пула (по умолчанию =1)
system.poold.objectives	string	Задает параметры использования ресурсов для poold
pset.poold.objectives	string	Задает параметры использования ресурсов для процессорной группы

1. Выполним следующие действия:
   - создадим процессорную группу *pset_test*
   - создадим пул *pool_test*
   - ассоциируем процессорную группу с пулом
   - объявим, что наш пул использует механизм FSS
   - посмотрим на изменения в конфигурации, я выделил их наклонным шрифтом：

```
# poolcfg -c 'create pset pset_test (uint pset.min = 1; uint pset.max = 2)'

# poolcfg -c 'create pool pool_test'

# poolcfg -c 'associate pool pool_test (pset pset_test)'

# poolcfg -c 'modify pool pool_test (string pool.scheduler = "FSS")'


# poolcfg -c info

system sunfire
      string      system.comment
      int         system.version 1
      boolean     system.bind-default true
      int         system.poold.pid 1606

      pool pool_default
            int         pool.sys_id 0
            boolean     pool.active true
            boolean     pool.default true
            int         pool.importance 1
            string      pool.comment
            pset        pset_default

      pool pool_test
            boolean     pool.active true
            boolean     pool.default false
            string      pool.scheduler FSS
            int         pool.importance 1
            string      pool.comment
            pset        pset_test

      pset pset_default
            int         pset.sys_id -1
            boolean     pset.default true
            uint        pset.min 1
            uint        pset.max 65536
            string      pset.units population
            uint        pset.load 9
            uint        pset.size 2
            string      pset.comment

            cpu
                  int         cpu.sys_id 1
                  string      cpu.comment
                  string      cpu.status on-line

            cpu
                  int         cpu.sys_id 0
                  string      cpu.comment
                  string      cpu.status on-line

      pset pset_test
            int         pset.sys_id -2
            boolean     pset.default false
            uint        pset.min 1
            uint        pset.max 2
            string      pset.units population
            uint        pset.load 0
            uint        pset.size 0
            string      pset.comment


# pooladm -c      /*сохраним изменения*/
```

2. Зададим параметры распределения ресурсов. Опция *wt-load* – контролирует распределение ресурсов, *locality none* – не влияет на распределение ресурсов, *locality tight* – выделять максимально возможные ресурсы.

```
# poolcfg -c 'modify system sunfire (string system.poold.objectives = "wt-
load")'

# poolcfg -c 'modify pset pset_default (string pset.poold.objectives =
"locality none")'

# poolcfg -c 'modify pset pset_test (string pset.poold.objectives = "locality
tight; utilization < 80")'

# pooladm -c       /*сохраним изменения*/
```

3. Создадим новую задачу в проекте *test* и поместим проект в наш пул:

```
# newtask -p test csh

# poolbind -i project -p pool_test test

sunfire# poolbind -q $$
2646  pool_test
sunfire#
```

4. Используя команду *poolstat* можно просматривать информацию о пулах:

```
# poolstat -r pset
 id pool            type rid rset                 min  max size used load
  1 pool_test       pset   1 pset_test               1    2    1 0.00 0.02
  0 pool_default    pset  -1 pset_default            1  66K    1 0.00 0.01
```

Заключение

С полной информацией о технологии Solaris Containers можно ознакомиться на сайте http://docs.sun.com, в материалах Solaris 10, “System Administration Guide: Solaris Containers – Resource Management and Solaris Zones”.