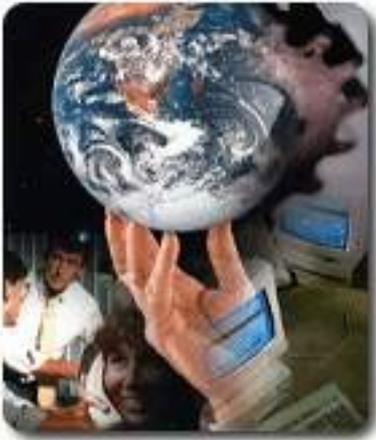


Embedded Alley

Solutions for Intelligent Devices



Memory

A Most Precious Resource

Dan Malek
Chief Technology
Officer

6 April 2009



Introduction

- Why Memory?
 - Capacity isn't time-based
 - If you need more, someone has to free
 - CPU too slow, it just takes longer
- What to do?
 - Protect my stash
 - Ask Linux OS for assistance



How Do We Do It?

- Cgroups Overview
- Memory Cgroup
- Memory Usage Notification
- Memory Cgroup Example
- Out of Memory (OOM) Killer
- Future Directions



Cgroups Overview

- Referenced by several terms
 - Containers (or Container Group)
 - Control, Controller Group
- We'll just say "cgroups"
 - What are cgroups?
 - How do they work?
 - Why do I care?



What Are Cgroups?

- A mechanism for partitioning sets of tasks
- Managed in a mounted virtual file system
- Can create a group hierarchy
 - Multiple groups within a tree
 - Root of tree is management point for sub-trees
- No system cost if not used, insignificant when used

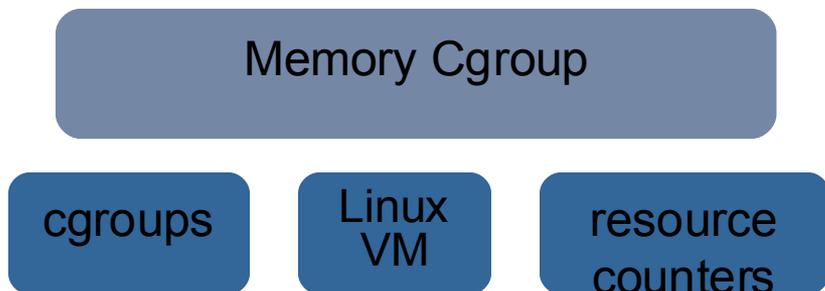
/cgroup

g_0/

g_1/



How Do Cgroups Work?



- A resource controller is the policy
- Tracks the reference counted objects
 - cpu usage
 - cpu sets
 - memory pages
- Works with a kernel subsystem for resource management



Why Do I Care?

- Powerful system resource management concept
- Resource consumers become part of the management
- OS doesn't have to guess (sometimes poorly)
 - Thread priorities are just hints
 - VM tuning knobs can be a research career
- Tuning moves to the application space
 - Ill-behaved tasks in their own container
 - Easily accommodate feature enhancement
 - Sensible system tuning perspective



Linux Cgroup Implementation

- Mounted virtual file system (i.e. /cgroups/<restype>)
 - echo pid > /cgroups/<restype>/<userclass>/tasks
- Easily and dynamically change resource controls
 - echo pid > /cgroups/<restype>/<new_class>/tasks
- Threads can determine cgroup information
 - /proc/<pid>/cgroup



Memory Cgroup

- Memory Resource Controller
 - Don't confuse with hardware memory controller
 - Track RSS and page cache pages, swap cache option
 - Reclaims through cgroup LRU
- Isolates memory behavior to a group of tasks
 - Prevent “memory hungry” tasks from consuming entire system memory resource
 - Control memory consumption for virtualization
 - Provide a protected container for critical embedded tasks



Memory Usage Notification

- Previous, stand-alone /dev/mem_notify
- New approach builds upon the memory cgroup resource tracking
- Kernel configuration option (CGROUP_MEM_NOTIFY)
- Select a cgroup usage limit notification percentage
 - Percentage rather than absolute value
 - Prevents need to update if cgroup is resized
- Task operates normally until notification
 - Can block-wait until limit
 - Can poll as part of normal processing



Memory Notification Cgroup Example

- Create a virtual file system
- Set resource limits
- Memory allocation code fragment
- Notification thread code fragment
- Example program message output



Create Virtual File System

- `mkdir -p /cgroups/memcg`
- `mount -t cgroup none /cgroups/memcg -o memory`
- `mkdir /cgroups/memcg/0`
- `echo $$ > /cgroups/memcg/0/tasks`



Set Resource Limits

```
# ls /cgroups/memcg/0
```

```
memory.failcnt
```

```
memory.force_empty
```

```
memory.limit_in_bytes
```

```
memory.max_usage_in_bytes
```

```
memory.notify_limit_lowait
```

```
memory.notify_limit_percent
```

```
memory.notify_limit_usage
```

```
memory.stat
```

```
memory.usage_in_bytes
```

```
notify_on_release
```

```
tasks
```

- Set the memory usage limit
 - `echo 10M > /cgroups/memcg/0/memory.limit_in_bytes`
- Set the notification limit to 80%
 - `echo 80 > /cgroups/memcg/0/memory.notify_limit_percent`



Memory Resource Notification Example

- Multi-threaded application
 - Main thread allocates 10 segments
 - Main thread frees segments if they still exist
 - Continues in a loop
- Notification thread
 - Blocks on `memory.notify_limit_lowait`
 - Frees allocated segments until `usage < limit`



Memory Allocation Code Fragment

```
k = 10;
while (k-- > 0) {
    for(i = 0; i<NSEGS; i++) {
        if ((mp = malloc(SEGSIZE)) == NULL) {
            perror("malloc");
            exit(2);
        }
        memptr[i] = mp;
        for (j = 0; j < SEGSIZE; j++)
            *mp++ = j;
        printf("Alloc seg %d\n", i);
        sleep(5);
    }
    for(i = 0; i<NSEGS; i++) {
        if ((mp = memptr[i]) != NULL)
            free(mp);
        printf("Free seg %d\n", i);
        sleep(5);
    }
}
```



Notification Thread Code Fragment

```
for (;;) {
    /* Open/read /cgroups/memcg/0/memory.notify_limit_lowait
    * Blocks while usage is below limit
    */
    percent = get_memcg_val("lowait");
    limit = get_memcg_val("percent");
    printf("Notify wakeup percent %d, limit %d\n", percent, limit);
    i = 0;
    do {
        if ((mp = memptr[i]) != NULL) {
            memptr[i] = NULL;
            free(mp);
            printf("Notify free seg %d\n", i);
        }
        i++;
        usage = get_memcg_val("usage");
    } while ((usage > limit) && (i < NSEGS));
}
```



Example Program Message Output

Alloc seg 0
Alloc seg 1
Alloc seg 2
Alloc seg 3
Alloc seg 4
Alloc seg 5
Notify wakeup percent 80, limit 80
Notify free seg 0
Alloc seg 6
Notify wakeup percent 80, limit 80
Notify free seg 1
Alloc seg 7
Notify wakeup percent 80, limit 80
Notify free seg 2
Alloc seg 8
Notify wakeup percent 80, limit 80
Notify free seg 3
Alloc seg 9
Free seg 0
Free seg 1
Free seg 2
.... and continues



Memory Resource Controller Challenges

- Moving a task doesn't migrate old allocations
 - Reclaims will deplete old cgroup allocation
 - New allocation charge to new cgroup
- Notification doesn't carry information
 - Normally, wake up is due to reaching notify limit
 - Wake up on thread migrate to new cgroup
 - Cgroup is forced empty (`memory.force_empty`)
 - Task must interrogate cgroup state to determine action



Out of Memory (OOM) Killer

- Linux chosen method of managing memory overload
 - Often nothing to kill in an embedded system
 - Difficult to make the right choice
- Operation is based upon kernel tuning
 - Memory overcommit
 - OOM adjustment knobs (per process)
 - Policy choices always under discussion
- Memory cgroup is subject to OOM
 - Overload will trigger OOM within cgroup
 - Can leverage OOM cgroup (<http://lwn.net/Articles/315949/>)



Future Direction

- Improve notification API
- Additional cgroup subsystems
- Increased granularity
 - requires lower overhead
 - asynchronous notification
 - information arguments passed with notification
- Active resource management programming model
 - Application states “..this is what I plan to need ...”
 - At end of block the need is revoked
- Find some assist for legacy applications



Summary

- All system resources are precious and must be managed
- Cgroups provide the mechanism for task partitioning
- A subsystem resource controller provides the policy
- Enables a powerful application-centric resource management
- Memory notification patch is in the e-mail queue to lkml