
Subject: Re: Hang with fair cgroup scheduler (reproducer is attached.)

Posted by [Ingo Molnar](#) on Fri, 14 Dec 2007 09:48:46 GMT

[View Forum Message](#) <> [Reply to Message](#)

(Cc:-ed other folks as well)

* KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com> wrote:

> Hi,
>
> While I was testing 2.6.24-rc5-mm1's fair group scheduler (with cgroup),
> the system hangs. please confirm. it's reproducible on my box.
>
> My test program is attached.
>
> What happens:
> the system hangs. (panic ?)
>
> Environ:
> ia64/NUMA 8CPU systems. 4 cpus per node.
>
> How to reproduce:
> Compile attached one.
> # gcc -o reg reg.c
> Create group as following
> # mount -t cgroup none /opt/cgroup -o cpu
> # mkdir /opt/cgroup/group_1
> # mkdir /opt/cgroup/group_2
>
> And run attached program
> # ./reg 8 8
>
> What 'reg' does;
> usage : reg A B C...
> This program forks child process and assign
> A of processes to group_1
> B of processes to group_2
> C of processes to group_3
> kick and waitpid all and repeat.
>
> Thanks,
> -Kame

> #include <stdlib.h>
> #include <stdio.h>
> #include <strings.h>
> #include <sys/types.h>
> #include <unistd.h>

```

> #include <sched.h>
> #include <asm/intrinsics.h>
> #include <sys/ipc.h>
> #include <sys/shm.h>
> #include <errno.h>
> #include <sys/times.h>
>
> static char *shared;
> #define MAX_PROCS 32
> #define SHMSIZE (16384)
>
> struct start_stop {
> int go;
> };
>
> /* Assign PID to a group....
> * work as # echo PID > /opt/cgroup/group_%d/tasks
> */
> void assign_to(int pid, int group)
> {
> FILE *fp;
> char buf[32];
>
> memset(buf, 0, sizeof(buf));
> sprintf(buf, "/opt/cgroup/group_%d/tasks", group);
> fp = fopen(buf, "w");
> if (fp == NULL) {
> perror("fopen");
> fprintf(stderr, "failed : fopen");
> exit(0);
> }
> fprintf(fp, "%d", pid);
> fclose(fp);
> printf("%d to %s\n", pid, buf);
> }
>
> /*
> * spin wait and go into small loop.
> * # of loops are counted as score.
> * This process's utime is recorded in times[id]
> */
> int worker(int id)
> {
> struct start_stop *shared_flag;
>
> shared_flag = (struct start_stop*)shared;
> do {
> sched_yield();

```

```

> ia64_mf();
> } while (!shared_flag->go);
> }
>
> /*
> * If you want to assign..
> * 2 procs to group 1, 3 procs to group 2 -># ./a.out 2 3
> * 3 procs to group 1, 3 procs to group 2, 3 procs to group 3
> * -># ./a.out 3 3 3
> * Total 32 procs are supported.
> */
>
> int main(int argc, char *argv[])
> {
> int nprocs;
> int shmid, i;
> struct start_stop *shared_flag;
> int pids[MAX_PROCS];
> int groups[MAX_PROCS];
>
> memset(pids, 0, sizeof(pids));
> memset(groups, 0, sizeof(groups));
>
> again:
> for (nprocs = 0, i = 1; i < argc; i++) {
> int num = atoi(argv[i]);
> int j;
> for (j = 0; j < num; j++) {
> groups[nprocs + j] = i;
> }
> nprocs += num;
> }
>
> shmid = shmget(IPC_PRIVATE, SHMSIZE, IPC_CREAT | 0666);
> if (shmid == -1) {
> perror("shmget");
> exit(1);
> }
>
> shared = shmat(shmid, NULL, 0);
> shared_flag = (struct start_stop *)shared;
>
> memset(shared, 0, SHMSIZE);
> shmctl(shmid, IPC_RMID, 0);
>
> for (i = 0; i < nprocs; i++) {
> int ret;
> ret = fork();

```

```
> if (ret == 0) {  
>     worker(i);  
>     exit(0);  
> } else if (ret == -1) {  
>     perror("fork");  
>     exit(0);  
> }  
> pids[i] = ret;  
> }  
> sleep(1);  
> for (i = 0; i < nprocs; i++)  
>     assign_to(pids[i], groups[i]);  
> sleep(1);  
> ia64_mf();  
> shared_flag->go = 1;  
>  
> for (i = 0; i < nprocs; i++) {  
>     int status;  
>     waitpid(pids[i], &status, 0);  
> }  
> goto again;  
>  
> return 0;  
> }
```

Containers mailing list
Containers@lists.linux-foundation.org
<https://lists.linux-foundation.org/mailman/listinfo/containers>
