
Subject: Re: [RFC PATCH 1/5] adds the procfs facilities

Posted by [serue](#) on Mon, 07 Jul 2008 18:30:30 GMT

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

Quoting Nadia.Derbey@bull.net (Nadia.Derbey@bull.net):

> [PATCH 01/05]

>

> This patch adds the procfs facility needed to feed some data for the
> next syscall to be called.

>

> The effect of issuing

> echo "LONG<Y> <XX>" > /proc/self/task/<tid>/next_syscall_data

> is that <XX> will be stored in a new field of the task structure

> (next_syscall_data). This field, in turn will be taken as the data to feed

> next syscall that supports the feature.

>

> <Y> is the number of values provided on the line.

> For the sake of simplicity it is now fixed to 1, but this can be extended as

> needed, in the future.

>

> This is particularly useful when restarting an application, as we need

> sometimes the syscalls to have a non-default behavior.

>

> Signed-off-by: Nadia Derby <Nadia.Derbey@bull.net>

>

> ---

> fs/exec.c | 6 +

> fs/proc/base.c | 75 ++++++

> include/linux/next_syscall_data.h | 35 ++++++

> include/linux/sched.h | 6 +

> kernel/Makefile | 3

> kernel/exit.c | 4 +

> kernel/fork.c | 2

> kernel/next_syscall_data.c | 151 ++++++

> 8 files changed, 281 insertions(+), 1 deletion(-)

>

> Index: linux-2.6.26-rc5-mm3/include/linux/sched.h

> =====

> --- linux-2.6.26-rc5-mm3.orig/include/linux/sched.h 2008-06-25 17:10:38.000000000 +0200

> +++ linux-2.6.26-rc5-mm3/include/linux/sched.h 2008-06-27 14:18:56.000000000 +0200

> @@ -87,6 +87,7 @@ struct sched_param {

> #include <linux/task_io_accounting.h>

> #include <linux/kobject.h>

> #include <linux/latencytop.h>

> +#include <linux/next_syscall_data.h>

>

> #include <asm/processor.h>

>

```

> @@ -1312,6 +1313,11 @@ struct task_struct {
> int latency_record_count;
> struct latency_record latency_record[LT_SAVECOUNT];
> #endif
> + /*
> + * If non-NULL indicates that next operation will be forced, e.g.
> + * that next object to be created will have a predefined id.
> + */
> + struct next_syscall_data *nsd;
> };
>
> /*
> Index: linux-2.6.26-rc5-mm3/include/linux/next_syscall_data.h
> =====
> --- /dev/null 1970-01-01 00:00:00.000000000 +0000
> +++ linux-2.6.26-rc5-mm3/include/linux/next_syscall_data.h 2008-07-01 10:25:48.000000000
> +0200
> @@ -0,0 +1,35 @@
> +/*
> + * include/linux/next_syscall_data.h
> + *
> + * Definitions to support fixed data for next syscall to be called. The
> + * following is supported today:
> + *   . object creation with a predefined id.
> + *
> + */
> +
> + #ifndef _LINUX_NEXT_SYSCALL_DATA_H
> + #define _LINUX_NEXT_SYSCALL_DATA_H
> +
> + #define NDATA 1
> +
> + /*
> + * If this structure is pointed to by a task_struct, next syscall to be called
> + * by the task will have a non-default behavior.
> + * For example, it can be used to pre-set the id of the object to be created
> + * by next syscall.
> + */
> + struct next_syscall_data {
> + int ndata;
> + long data[NDATA];
> + };
> +
> + extern ssize_t get_next_syscall_data(struct task_struct *, char *, size_t);
> + extern int set_next_syscall_data(struct task_struct *, char *);
> + extern int reset_next_syscall_data(struct task_struct *);
> +
> + static inline void exit_next_syscall_data(struct task_struct *tsk)

```

```

> +{
> + reset_next_syscall_data(tsk);
> +}
> +
> +#endif /* _LINUX_NEXT_SYSCALL_DATA_H */
> Index: linux-2.6.26-rc5-mm3/fs/proc/base.c
> =====
> --- linux-2.6.26-rc5-mm3.orig/fs/proc/base.c 2008-06-25 17:11:04.000000000 +0200
> +++ linux-2.6.26-rc5-mm3/fs/proc/base.c 2008-07-01 09:09:30.000000000 +0200
> @@ -1158,6 +1158,76 @@ static const struct file_operations proc
> };
> #endif
>
> +static ssize_t next_syscall_data_read(struct file *file, char __user *buf,
> +   size_t count, loff_t *ppos)
> +{
> + struct task_struct *task;
> + char *page;
> + ssize_t length;
> +
> + task = get_proc_task(file->f_path.dentry->d_inode);
> + if (!task)
> +   return -ESRCH;
> +
> + if (count >= PAGE_SIZE)
> +   count = PAGE_SIZE - 1;
> +
> + length = -ENOMEM;
> + page = (char *) __get_free_page(GFP_TEMPORARY);
> + if (!page)
> +   goto out;
> +
> + length = get_next_syscall_data(task, (char *) page, count);
> + if (length >= 0)
> +   length = simple_read_from_buffer(buf, count, ppos,
> +   (char *)page, length);
> + free_page((unsigned long) page);
> +
> +out:
> + put_task_struct(task);
> + return length;
> +}
> +
> +static ssize_t next_syscall_data_write(struct file *file,
> +   const char __user *buf,
> +   size_t count, loff_t *ppos)
> +{
> + struct inode *inode = file->f_path.dentry->d_inode;

```

```

> + char *page;
> + ssize_t length;
> +
> + if (pid_task(proc_pid(inode), PIDTYPE_PID) != current)
> + return -EPERM;
> +
> + if (count >= PAGE_SIZE)
> + count = PAGE_SIZE - 1;
> +
> + if (*ppos != 0) {
> + /* No partial writes. */
> + return -EINVAL;
> + }
> + page = (char *)__get_free_page(GFP_TEMPORARY);
> + if (!page)
> + return -ENOMEM;
> + length = -EFAULT;
> + if (copy_from_user(page, buf, count))
> + goto out_free_page;
> +
> + page[count] = '\0';
> +
> + length = set_next_syscall_data(current, page);
> + if (!length)
> + length = count;
> +
> +out_free_page:
> + free_page((unsigned long) page);
> + return length;
> +}
> +
> +static const struct file_operations proc_next_syscall_data_operations = {
> + .read = next_syscall_data_read,
> + .write = next_syscall_data_write,
> +};
>
> #ifdef CONFIG_SCHED_DEBUG
> /*
> @@ -2853,6 +2923,11 @@ static const struct pid_entry tid_base_s
> #ifdef CONFIG_TASK_IO_ACCOUNTING
> INF("io", S_IRUGO, tid_io_accounting),
> #endif
> + /*
> + * NOTE that this file is not added into tgid_base_stuff[] since it
> + * has to be specified on a per-thread basis.
> + */
> + REG("next_syscall_data", S_IRUGO|S_IWUSR, next_syscall_data),
> };

```

```

>
> static int proc_tid_base_readdir(struct file * filp,
> Index: linux-2.6.26-rc5-mm3/kernel/Makefile
> =====
> --- linux-2.6.26-rc5-mm3.orig/kernel/Makefile 2008-06-25 17:10:41.000000000 +0200
> +++ linux-2.6.26-rc5-mm3/kernel/Makefile 2008-06-27 09:03:01.000000000 +0200
> @@ -9,7 +9,8 @@ obj-y    = sched.o fork.o exec_domain.o
>    rcupdate.o extable.o params.o posix-timers.o \
>    kthread.o wait.o kfifo.o sys_ni.o posix-cpu-timers.o mutex.o \
>    hrtimer.o rwsem.o nsproxy.o srcu.o semaphore.o \
> -   notifier.o ksysfs.o pm_qos_params.o sched_clock.o
> +   notifier.o ksysfs.o pm_qos_params.o sched_clock.o \
> +   next_syscall_data.o
>
> CFLAGS_REMOVE_sched.o = -pg -mno-spe
>
> Index: linux-2.6.26-rc5-mm3/kernel/next_syscall_data.c
> =====
> --- /dev/null 1970-01-01 00:00:00.000000000 +0000
> +++ linux-2.6.26-rc5-mm3/kernel/next_syscall_data.c 2008-07-01 10:39:43.000000000 +0200
> @@ -0,0 +1,151 @@
> +/*
> + * linux/kernel/next_syscall_data.c
> + *
> + *
> + * Provide the get_next_syscall_data() / set_next_syscall_data() routines
> + * (called from fs/proc/base.c).
> + * They allow to specify some particular data for the next syscall to be
> + * called.
> + * E.g. they can be used to specify the id for the next resource to be
> + * allocated, instead of letting the allocator set it for us.
> + */
> +
> +#include <linux/sched.h>
> +#include <linux/ctype.h>
> +
> +
> +
> +ssize_t get_next_syscall_data(struct task_struct *task, char *buffer,
> +    size_t size)
> +{
> +    struct next_syscall_data *nsd;
> +    char *bufptr = buffer;
> +    ssize_t rc, count = 0;
> +    int i;
> +
> +    nsd = task->nsd;
> +    if (!nsd || !nsd->ndata)

```

```

> + return snprintf(buffer, size, "UNSET\n");
> +
> + count = snprintf(bufptr, size, "LONG%d ", nsd->ndata);
> +
> + for (i = 0; i < nsd->ndata - 1; i++) {
> + rc = snprintf(&bufptr[count], size - count, "%ld ",
> + nsd->data[i]);
> + if (rc >= size - count)
> + return -ENOMEM;
> + count += rc;
> + }
> +
> + rc = snprintf(&bufptr[count], size - count, "%ld\n", nsd->data[i]);
> + if (rc >= size - count)
> + return -ENOMEM;
> + count += rc;
> +
> + return count;
> +}
> +
> +static int fill_next_syscall_data(struct task_struct *task, int ndata,
> + char *buffer)
> +{
> + char *token, *buff = buffer;
> + char *end;
> + struct next_syscall_data *nsd = task->nsd;
> + int i;
> +
> + if (!nsd) {
> + nsd = kmalloc(sizeof(*nsd), GFP_KERNEL);
> + if (!nsd)
> + return -ENOMEM;
> + task->nsd = nsd;
> + }
> +
> + nsd->ndata = ndata;
> +
> + i = 0;
> + while ((token = strsep(&buff, " ")) != NULL && i < ndata) {
> + long data;
> +
> + if (!*token)
> + goto out_free;
> + data = simple_strtol(token, &end, 0);
> + if (end == token || (*end && !isspace(*end)))
> + goto out_free;
> + nsd->data[i] = data;
> + i++;

```

```

> + }
> +
> + if (i != ndata)
> + goto out_free;
> +
> + return 0;
> +
> +out_free:
> + kfree(nsd);

```

Shouldn't you also reset task->nsd to NULL here? :-)

```

> + return -EINVAL;
> +}
> +
> +/*
> + * Parses a line with the following format:
> + * <x> <id0> ... <idx-1>
> + * Currently, only x=1 is accepted.
> + * Any trailing character on the line is skipped.
> + */
> +static int do_set_next_syscall_data(struct task_struct *task, char *nb,
> + char *buffer)
> +{
> + int ndata;
> + char *end;
> +
> + ndata = simple_strtol(nb, &end, 0);
> + if (*end)
> + return -EINVAL;
> +
> + if (ndata > NDATA)
> + return -EINVAL;
> +
> + return fill_next_syscall_data(task, ndata, buffer);
> +}
> +
> +int reset_next_syscall_data(struct task_struct *task)

```

Why have this return an int? It always returns 0, and callers ignore the return value.

```

> +{
> + struct next_syscall_data *nsd;
> +
> + nsd = task->nsd;
> + if (!nsd)
> + return 0;

```

```

> +
> + task->nsd = NULL;
> + kfree(nsd);
> + return 0;
> +}
> +
> + #define LONG_STR "LONG"
> + #define RESET_STR "RESET"
> +
> + /*
> + * Parses a line written to /proc/self/task/<my_tid>/next_syscall_data.
> + * this line has the following format:
> + * LONG<x> id          --> a sequence of id(s) is specified
> + *                    currently, only x=1 is accepted
> + */
> + int set_next_syscall_data(struct task_struct *task, char *buffer)
> + {
> +     char *token, *out = buffer;
> +     size_t sz;
> +
> +     if (!out)
> +         return -EINVAL;
> +
> +     token = strsep(&out, " ");
> +
> +     sz = strlen(LONG_STR);
> +
> +     if (!strncmp(token, LONG_STR, sz))
> +         return do_set_next_syscall_data(task, token + sz, out);
> +
> +     if (!strncmp(token, RESET_STR, strlen(RESET_STR)))
> +         return reset_next_syscall_data(task);
> +
> +     return -EINVAL;
> + }
> Index: linux-2.6.26-rc5-mm3/kernel/fork.c
> =====
> --- linux-2.6.26-rc5-mm3.orig/kernel/fork.c 2008-06-25 17:10:41.000000000 +0200
> +++ linux-2.6.26-rc5-mm3/kernel/fork.c 2008-07-01 10:25:46.000000000 +0200
> @@ -1077,6 +1077,8 @@ static struct task_struct *copy_process(
>     p->blocked_on = NULL; /* not blocked yet */
> #endif
>
> + p->nsd = NULL; /* no next syscall data is the default */
> +
> /* Perform scheduler related setup. Assign this task to a CPU. */
> sched_fork(p, clone_flags);
>

```



```

> Index: linux-2.6.26-rc5-mm3/fs/exec.c
> =====
> --- linux-2.6.26-rc5-mm3.orig/fs/exec.c 2008-06-25 17:11:05.000000000 +0200
> +++ linux-2.6.26-rc5-mm3/fs/exec.c 2008-06-27 14:53:08.000000000 +0200
> @@ -1014,6 +1014,12 @@ int flush_old_exec(struct linux_binprm *
>  flush_signal_handlers(current, 0);
>  flush_old_files(current->files);
>
> + /*
> +  * the next syscall data is not inherited across execve()
> +  */
> + if (unlikely(current->nsd))
> +  reset_next_syscall_data(current);
> +
>  return 0;
>
> out:
> Index: linux-2.6.26-rc5-mm3/kernel/exit.c
> =====
> --- linux-2.6.26-rc5-mm3.orig/kernel/exit.c 2008-06-25 17:10:41.000000000 +0200
> +++ linux-2.6.26-rc5-mm3/kernel/exit.c 2008-06-27 14:57:55.000000000 +0200
> @@ -1069,6 +1069,10 @@ NORET_TYPE void do_exit(long code)
>
>  proc_exit_connector(tsk);
>  exit_notify(tsk, group_dead);
> +
> + if (unlikely(tsk->nsd))
> +  exit_next_syscall_data(tsk);
> +
>  #ifdef CONFIG_NUMA
>  mpol_put(tsk->mempolicy);
>  tsk->mempolicy = NULL;
>
> --

```

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