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

Posted by [serue](#) on Tue, 08 Jul 2008 19:32:54 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 Derbey <Nadia.Derbey@bull.net>

Acked-by: Serge Hallyn <serue@us.ibm.com>

thanks,

-serge

>
> ---
> fs/exec.c | 6 +
> fs/proc/base.c | 75 ++++++
> include/linux/next_syscall_data.h | 32 +
> 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, 278 insertions(+), 1 deletion(-)
>
> Index: linux-2.6.26-rc8-mm1/include/linux/sched.h
> ======
> --- linux-2.6.26-rc8-mm1.orig/include/linux/sched.h 2008-07-08 09:04:21.000000000 +0200
> +++ linux-2.6.26-rc8-mm1/include/linux/sched.h 2008-07-08 09:13:43.000000000 +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>
>
> @@ -1296,6 +1297,11 @@ 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-rc8-mm1/include/linux/next_syscall_data.h
> =====
> --- /dev/null 1970-01-01 00:00:00.000000000 +0000
> +++ linux-2.6.26-rc8-mm1/include/linux/next_syscall_data.h 2008-07-08 09:24:38.000000000
+0200
> @@ -0,0 +1,32 @@
> +/*
> + * include/linux/next_syscall_data.h
> + *
> + * Definitions to support fixed data for next syscall to be called.
> + */
> +
> +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 void 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-rc8-mm1/fs/proc/base.c
> =====
> --- linux-2.6.26-rc8-mm1.orig/fs/proc/base.c 2008-07-08 09:05:13.000000000 +0200
> +++ linux-2.6.26-rc8-mm1/fs/proc/base.c 2008-07-08 09:18:12.000000000 +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 *,
> +    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-rc8-mm1/kernel/Makefile
> =====
> --- linux-2.6.26-rc8-mm1.orig/kernel/Makefile 2008-07-08 09:04:35.000000000 +0200
> +++ linux-2.6.26-rc8-mm1/kernel/Makefile 2008-07-08 09:19:14.000000000 +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-rc8-mm1/kernel/next_syscall_data.c
> =====
> --- /dev/null 1970-01-01 00:00:00.000000000 +0000
> +++ linux-2.6.26-rc8-mm1/kernel/next_syscall_data.c 2008-07-08 09:35:27.000000000 +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->nssd;
> + if (!nsd || !nsd->nssdata)
> +   return sprintf(buffer, size, "UNSET\n");
> +
> + count = sprintf(bufptr, size, "LONG%d ", nsd->nssdata);
> +
> + for (i = 0; i < nsd->nssdata - 1; i++) {
> +   rc = sprintf(&bufptr[count], size - count, "%ld ",
> +     nsd->nssdata[i]);
> +   if (rc >= size - count)
> +     return -ENOMEM;
> +   count += rc;
> +
> + }
> +
> + rc = sprintf(&bufptr[count], size - count, "%ld\n", nsd->nssdata[i]);
> + if (rc >= size - count)
> +   return -ENOMEM;
> + count += rc;
> +
> + return count;
> +
> +
> +static int fill_next_syscall_data(struct task_struct *task, int nssdata,
> +    char *buffer)
> +{
> +  char *token, *buff = buffer;
> +  char *end;
> +  struct next_syscall_data *nsd = task->nssd;
> +  int i;
> +
> +  if (!nsd) {
> +    nsd = kmalloc(sizeof(*nsd), GFP_KERNEL);
> +    if (!nsd)
> +      return -ENOMEM;
> +    task->nssd = nsd;
> +  }
> +
> +  nsd->nssdata = nssdata;
> +
> +  i = 0;
> +  while ((token = strsep(&buff, " ")) != NULL && i < nssdata) {
> +    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);
> + task->nsd = NULL;
> + 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);
> +}
> +
> +void reset_next_syscall_data(struct task_struct *task)
> +{
> +    struct next_syscall_data *nsd = task->nsd;
> +
> +    if (nsd) {
> +        task->nsd = NULL;
> +        kfree(nsd);
> +    }
> +
> +

```

```

> +#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))) {
> +        reset_next_syscall_data(task);
> +        return 0;
> +    }
> +
> +    return -EINVAL;
> +}
> Index: linux-2.6.26-rc8-mm1/kernel/fork.c
> =====
> --- linux-2.6.26-rc8-mm1.orig/kernel/fork.c 2008-07-08 09:04:35.000000000 +0200
> +++ linux-2.6.26-rc8-mm1/kernel/fork.c 2008-07-08 09:25:35.000000000 +0200
> @@ -1085,6 +1085,8 @@ static struct task_struct *copy_process(
>     p->blocked_on = NULL; /* not blocked yet */
> #endif
>
> + p->nssd = 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-rc8-mm1/fs/exec.c
> =====
> --- linux-2.6.26-rc8-mm1.orig/fs/exec.c 2008-07-08 09:05:13.000000000 +0200
> +++ linux-2.6.26-rc8-mm1/fs/exec.c 2008-07-08 09:26:21.000000000 +0200

```

```
> @@ -1016,6 +1016,12 @@ 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->nscd))  
> +     reset_next_syscall_data(current);  
> +  
>     return 0;  
>  
> out:  
> Index: linux-2.6.26-rc8-mm1/kernel/exit.c  
> ======  
> --- linux-2.6.26-rc8-mm1.orig/kernel/exit.c 2008-07-08 09:04:35.000000000 +0200  
> +++ linux-2.6.26-rc8-mm1/kernel/exit.c 2008-07-08 09:27:31.000000000 +0200  
> @@ -1066,6 +1066,10 @@ NORET_TYPE void do_exit(long code)  
>  
>     proc_exit_connector(tsk);  
>     exit_notify(tsk, group_dead);  
> +  
> + if (unlikely(tsk->nscd))  
> +     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>
