

FW: [RFC] A more general timeout specification

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2005-08/8036.html>

From: Perez-Gonzalez, Inaky (inaky.perez-gonzalez_at_intel.com)

Date: 08/31/05

Date: Wed, 31 Aug 2005 13:55:54 -0700

To: <akpm@osdl.org>

Hi Andrew

I would like to ask for the inclusion of the general timeout specification API in the -mm tree, looking forward it to bubble up to mainline.

For more context, please visit:

http://groups.google.com/group/linux.kernel/browse_frm/thread/a426e5b0d0f17860/b53000aa01304c4a?vc=1&q=a+more+general+timeout+specification#b53000aa01304c4a

This was developed by Joe Korty <joe.korty@ccur.com>, greatly enhancing something I had done before, so I am signing it out (although Joe should too, Joe?).

--

The fusyn (robust mutexes) project proposes the creation of a more general data structure, 'struct timeout', for the specification of timeouts in new services. In this structure, the user specifies:

- a time, in timespec format.

- the clock the time is specified against (eg, CLOCK_MONOTONIC).

- whether the time is absolute, or relative to 'now'.

That is, all combinations of useful timeout attributes become possible.

Also proposed are two new kernel routines for the manipulation of timeouts:

- `timeout_validate()`

- `timeout_sleep()`

`timeout_validate()` error-checks the syntax of a timeout argument and returns either zero or `-EINVAL`. By breaking `timeout_validate()` out from `timeout_sleep()`, it becomes possible to error check the timeout 'far away' from the places in the code where we would actually do the timeout, as well as being able to perform such checks only at those places we know the timeout specification is coming from an unsafe source.

`timeout_sleep()` puts the caller to sleep until the specified end time is in the past, as measured against the given clock, or until the caller is awakened by other means (such as `wake_up_process()`). Like `schedule_timeout()`,

Linux-Kernel: FW: [RFC] A more general timeout specification

TASK_INTERRUPTIBLE or TASK_UNINTERRUPTIBLE must be set ahead of time; if TASK_INTERRUPTIBLE is set then signals will also break the caller out of the sleep.
timeout_sleep() returns either 0 (returned early) or -ETIMEDOUT (returned due to timeout). It is up to the caller to resolve, in the "returned early" case, why it returned early.
Timeout_sleep has a return argument, endtime, which is also in 'struct timeout' format. If the input time was relative, then it is converted to absolute and returned through this argument. This can be used when an early-terminated service must be restarted and side effects of the early termination-n-restart (such as end time drift) are to be avoided.

Joe

"Money can buy bandwidth, but latency is forever" -- John Mashey

Signed-off-by: Inaky Perez-Gonzalez <inaky.perez-gonzalez@intel.com>

```
2.6.12-rc4-jak/include/linux/time.h | 6 +
2.6.12-rc4-jak/include/linux/timeout.h | 48 ++++++++
2.6.12-rc4-jak/kernel/posix-timers.c | 7 +
2.6.12-rc4-jak/kernel/timer.c | 184
+++++
4 files changed, 245 insertions(+)
diff -puNa include/linux/time.h~a.more.flexible.timeout.approach
include/linux/time.h
--- 2.6.12-rc4/include/linux/time.h~a.more.flexible.timeout.approach
2005-05-18 13:53:14.204417169 -0400
+++ 2.6.12-rc4-jak/include/linux/time.h 2005-05-18 13:53:14.212416002
-0400
@@ -25,6 +25,8 @@ struct timezone {
    int      tz_dsttime;      /* type of dst correction */
};

+#include <linux/timeout.h>
+
+#ifdef __KERNEL__

/* Parameters used to convert the timespec values */
@@ -103,6 +105,10 @@ struct itimerval;
extern int do_setitimer(int which, struct itimerval *value, struct
itimerval *ovalue);
extern int do_getitimer(int which, struct itimerval *value);
extern void getnstimeofday (struct timespec *tv);
+extern long clock_gettime(int which, struct timespec *tp);
+
+extern int FASTCALL(abs_timespec_to_abs_jiffies (clockid_t clock, const
struct timespec *tp, unsigned long *jp));
+extern int FASTCALL(rel_to_abs_timespec(clockid_t clock, const struct
timespec *tsrel, struct timespec *tsabs));

extern struct timespec timespec_trunc(struct timespec t, unsigned
gran);

diff -puNa /dev/null include/linux/timeout.h
--- /dev/null 2004-06-24 14:04:38.000000000 -0400
+++ 2.6.12-rc4-jak/include/linux/timeout.h 2005-05-18
13:53:14.212416002 -0400
@@ -0,0 +1,48 @@
+/*
+ * Extended timeout specification
+ *
+ * (C) 2002-2005 Intel Corp
+ * Inaky Perez-Gonzalez <inaky.perez-gonzalez@intel.com>.
+ */
```

Linux-Kernel: FW: [RFC] A more general timeout specification

```
+ * Licensed under the FSF's GNU Public License v2 or later.
+ *
+ * Generic extended timeout specification. Broken out by Joe Korty
+ * <joe.korty@ccur.com> from linux/time.h so that it can be included
+ * by userspace applications in conjunction with #include "time.h".
+ */
+
+#ifndef _LINUX_TIMEOUT_H
+#define _LINUX_TIMEOUT_H
+
+/* 'struct timeout' flag values. OR these into clock_id along with
+ * a clock specification such as CLOCK_REALTIME or CLOCK_MONOTONIC.
+ */
+enum {
+    TIMEOUT_RELATIVE    = 0x10000000,        /* relative timeout */
+
+    TIMEOUT_FLAGS_MASK = 0xf0000000,        /* flags mask for
clock_id */
+    TIMEOUT_CLOCK_MASK = 0x0fffffff,        /* clock mask for
clock_id */
+};
+
+/* Magic values a 'struct timeout' pointer can have */
+
+#define TIMEOUT_MAX    ((struct timeout *) ~OUL) /* never time out */
+#define TIMEOUT_NONE   ((struct timeout *) OUL)  /* time out
immediately */
+
+/**
+ * struct timeout - general timeout specification
+ *
+ * @clock_id: which clock source to use ORed with flags describing use.
+ * @ts:      timespec for the timeout
+ */
+struct timeout {
+    clockid_t clock_id;
+    struct timespec ts;
+};
+
+#ifdef __KERNEL__
+extern int FASTCALL(timeout_validate (const struct timeout *));
+extern int FASTCALL(timeout_sleep (const struct timeout *, struct
timeout *));
+#endif
+
+#endif
diff -puNa kernel/posix-timers.c~a.more.flexible.timeout.approach
kernel/posix-timers.c
--- 2.6.12-rc4/kernel/posix-timers.c~a.more.flexible.timeout.approach
2005-05-18 13:53:14.207416731 -0400
+++ 2.6.12-rc4-jak/kernel/posix-timers.c      2005-05-18
13:53:14.214415710 -0400
@@ -1288,6 +1288,13 @@ sys_clock_settime(clockid_t which_clock,
return CLOCK_DISPATCH(which_clock, clock_set, (which_clock,
&new_tp));
}

+long clock_gettime(clockid_t which_clock, struct timespec *tp)
+{
+    if (invalid_clockid(which_clock))
+        return -EINVAL;
+    return CLOCK_DISPATCH(which_clock, clock_get, (which_clock,
```

Linux-Kernel: FW: [RFC] A more general timeout specification

```
tp));
+}
+
+ asmlinkage long
+ sys_clock_gettime(clockid_t which_clock, struct timespec __user *tp)
+ {
diff -puNa kernel/timer.c~a.more.flexible.timeout.approach
kernel/timer.c
--- 2.6.12-rc4/kernel/timer.c~a.more.flexible.timeout.approach
2005-05-18 13:53:14.209416440 -0400
+++ 2.6.12-rc4-jak/kernel/timer.c      2005-05-18 15:27:06.363710594
-0400
@@ -1129,6 +1129,190 @@ fastcall signed long __sched schedule_ti

EXPORT_SYMBOL(schedule_timeout);

+/**
+ * timeout_validate - verify that a timeout specification is
+ self-consistant.
+ * @tp - pointer to the 'struct timeout' to verify
+ * @returns - 0 if no error, <0 errno code if there was
+ */
+fastcall int timeout_validate(const struct timeout *tp)
+{
+    int result;
+    unsigned flags, clock_id;
+
+    result = 0;
+    if (tp == TIMEOUT_MAX || tp == TIMEOUT_NONE)
+        goto out;
+
+    flags = tp->clock_id & TIMEOUT_FLAGS_MASK;
+    clock_id = tp->clock_id & TIMEOUT_CLOCK_MASK;
+
+    result = -EINVAL;
+    if (flags & ~TIMEOUT_RELATIVE)
+        goto out;
+
+    /* someday, we should support *all* clocks available to us */
+    if (clock_id != CLOCK_REALTIME && clock_id != CLOCK_MONOTONIC)
+        goto out;
+    if ((unsigned long)tp->ts.tv_nsec >= NSEC_PER_SEC)
+        goto out;
+    result = 0;
+out:
+    return result;
+}
+EXPORT_SYMBOL_GPL(timeout_validate);
+
+/** rel_to_abs_timespec - convert a relative timespec into an absolute
+ timespec
+ * @clock - the system clock the timespecs are specified in.
+ * @tsrel - a pointer to the relative timespec to be converted.
+ * @tsabs - a pointer to where the absolute timespec is to be stored.
+ * @returns - 0 if converted, <0 if error.
+ */
+fastcall int rel_to_abs_timespec(clockid_t clock, const struct timespec
+*tsrel,
+
+                                struct timespec *tsabs)
+{
+    int result;
+    struct timespec now;
```

Linux-Kernel: FW: [RFC] A more general timeout specification

```
+
+     result = clock_gettime(clock, &now);
+     if (result >= 0) {
+         set_normalized_timespec(&tsabs, now.tv_sec +
+tsrel->tv_sec,
+                                 now.tv_nsec + tsrel->tv_nsec);
+     }
+     return result;
+}
+EXPORT_SYMBOL_GPL(rel_to_abs_timespec);
+
+/**
+ * abs_timespec_to_abs_jiffies - convert an absolute timespec into
+ * absolute jiffies.
+ *
+ * @clock - select which clock @tp is specified against.
+ * CLOCK_MONOTONIC and
+ * CLOCK_REALTIME are two possibilities.
+ * @tp - absolute timespec that is to be converted to jiffies.
+ * @jp - absolute jiffies returned through this pointer.
+ * @returns - 0 if converted, <0 if error.
+ */
+fastcall int abs_timespec_to_abs_jiffies (clockid_t clock,
+                                         const struct timespec *tp, unsigned long *jp)
+{
+     int result;
+     unsigned long jiffies_abs, seq;
+     struct timespec oc, now;
+
+     do {
+         seq = read_seqbegin(&xtime_lock);
+         result = clock_gettime(clock, &now);
+         if (result < 0)
+             return result;
+         jiffies_abs = jiffies;
+     } while (read_seqretry(&xtime_lock, seq));
+
+     set_normalized_timespec(&oc, tp->tv_sec - now.tv_sec,
+                             tp->tv_nsec - now.tv_nsec);
+
+     if (oc.tv_sec > 0 || (oc.tv_sec == 0 && oc.tv_nsec > 0))
+         jiffies_abs += timespec_to_jiffies(&oc) + 1;
+     *jp = jiffies_abs;
+     return 0;
+}
+EXPORT_SYMBOL_GPL(abs_timespec_to_abs_jiffies);
+
+/* Helper function.  Handles the proper setting of @endtime when
+ * an @endtime needs to be returned.  Returns an approximation for
+ * the ending time in jiffies.  The approximation is guaranteed to
+ * be on or just past the true ending time.
+ */
+static inline
+unsigned long __timeout_sleep_timespec (const struct timeout *timeout,
+                                       struct timeout *endtime)
+{
+     unsigned long jiffies_abs;
+     struct timespec end;
+     clockid_t clock_id = timeout->clock_id & TIMEOUT_CLOCK_MASK;
+
+     if (timeout->clock_id & TIMEOUT_RELATIVE) {
+         rel_to_abs_timespec(clock_id, &timeout->ts, &end);
+     }
+}
```

Linux-Kernel: FW: [RFC] A more general timeout specification

```
+         if (endtime) {
+             endtime->clock_id = clock_id;
+             endtime->ts = end;
+         }
+     }
+     else
+         end = timeout->ts;
+
+     abs_timespec_to_abs_jiffies(clock_id, &end, &jiffies_abs);
+     return jiffies_abs;
+}
+
+/**
+ * timeout_sleep - sleep until woken up, interrupted, or the
+ *                designated wakeup time is past.
+ * @timeout:
+ *   time to wait.  const struct timeout pointer.  May take
+ *   on the special values TIMEOUT_MAX (never time out) or
+ *   TIMEOUT_NONE (times out instantly).  @timeout must be
+ *   error-checked by timeout_validate() beforehand.
+ * @endtime:
+ *   time remaining.  A 'struct timeout' value, holding the
+ *   end time of the period in absolute format and specified
+ *   against the same clock that @timeout was specified against,
+ *   is returned through this pointer.  No value is returned
+ *   if @endtime is NULL, if @timeout is TIMEOUT_MAX or
+ *   TIMEOUT_NONE, or if @timeout is itself specified in
+ *   absolute time.
+ * @returns:
+ *   -ETIMEDOUT if awakened due to timeout, 0 otherwise.
+ */
+fastcall int timeout_sleep (const struct timeout *timeout, struct
+timeout *endtime)
+{
+     unsigned long jiffies_endtime;
+     struct timer_list timer;
+     int result;
+
+     if (timeout == TIMEOUT_NONE)
+         goto simulate_timeout;
+
+     if (timeout == TIMEOUT_MAX)
+         goto never_timeout;
+
+     if (timeout->ts.tv_sec == MAX_SCHEDULE_TIMEOUT)
+         goto never_timeout;
+
+     jiffies_endtime = __timeout_sleep_timespec(timeout, endtime);
+
+     if (time_after_eq(jiffies, jiffies_endtime))
+         goto simulate_timeout;
+
+     init_timer(&timer);
+     timer.expires = jiffies_endtime;
+     timer.data = (unsigned long)current;
+     timer.function = process_timeout;
+     add_timer(&timer);
+     schedule();
+     result = -ETIMEDOUT;
+     if (del_singleshot_timer_sync (&timer))
+         result = 0;
+     goto out;
+}
```

Linux-Kernel: FW: [RFC] A more general timeout specification

```
+
+     /*
+     * simulate a timeout without actually expiring a timer
+     */
+simulate_timeout:
+     process_timeout((unsigned long)current);
+     result = -ETIMEDOUT;
+     goto out;
+
+     /*
+     * Sleep without a timeout scheduled.
+     */
+never_timeout:
+     schedule();
+     result = 0;
+out:
+     return result;
+}
+EXPORT_SYMBOL_GPL (timeout_sleep);
+
+/* Thread ID - the internal kernel "pid" */
+asmlinkage long sys_gettid(void)
+{
```

```
--
Inaky
```

-

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in the body of a message to majordomo@vger.kernel.org

More majordomo info at <http://vger.kernel.org/majordomo-info.html>

Please read the FAQ at <http://www.tux.org/lkml/>