```
#
# This file contains a few gdb macros (user defined commands) to extract
# useful information from kernel crashdump (kdump) like stack traces of
# all the processes or a particular process and trapinfo.
#
# These macros can be used by copying this file in .gdbinit (put in home
# directory or current directory) or by invoking gdb command with
# --command=<command-file-name> option
#
# Credits:
# Alexander Nyberg <alexn@telia.com>
# V Srivatsa <vatsa@in.ibm.com>
# Maneesh Soni <maneesh@in.ibm.com>
#

define bttnobp
        set $tasks_off=((size_t)&((struct task_struct *)0)->tasks)
        set $pid_off=((size_t)&((struct task_struct *)0)->pids[1].pid_list.next)
        set $init_t=&init_task
        set $next_t=(((char *)($init_t->tasks).next) - $tasks_off)
        while ($next_t != $init_t)
                set $next_t=(struct task_struct *)$next_t
                printf "\npid %d; comm %s:\n", $next_t.pid, $next_t.comm
                printf "==================\n"
                set var $stackp = $next_t.thread.esp
                set var $stack_top = ($stackp & ~4095) + 4096

                while ($stackp < $stack_top)
                        if (*($stackp) > _stext && *($stackp) < _sinittext)
                                info symbol *($stackp)
                        end
                        set $stackp += 4
                end
                set $next_th=(((char *)$next_t->pids[1].pid_list.next) -
$pid_off)
                while ($next_th != $next_t)
                        set $next_th=(struct task_struct *)$next_th
                        printf "\npid %d; comm %s:\n", $next_t.pid, $next_t.comm
                        printf "==================\n"
                        set var $stackp = $next_t.thread.esp
                        set var $stack_top = ($stackp & ~4095) + 4096

                        while ($stackp < $stack_top)
                                if (*($stackp) > _stext && *($stackp) <
_sinittext)
                                        info symbol *($stackp)
                                end
                                set $stackp += 4
                        end
                        set $next_th=(((char *)$next_th->pids[1].pid_list.next)
- $pid_off)
                end
                set $next_t=(char *)($next_t->tasks.next) - $tasks_off
        end
end
document bttnobp
```

```
        dump all thread stack traces on a kernel compiled with
!CONFIG_FRAME_POINTER
end

define btt
        set $tasks_off=((size_t)&((struct task_struct *)0)->tasks)
        set $pid_off=((size_t)&((struct task_struct *)0)->pids[1].pid_list.next)
        set $init_t=&init_task
        set $next_t=(((char *)($init_t->tasks).next) - $tasks_off)
        while ($next_t != $init_t)
                set $next_t=(struct task_struct *)$next_t
                printf "\npid %d; comm %s:\n", $next_t.pid, $next_t.comm
                printf "==================\n"
                set var $stackp = $next_t.thread.esp
                set var $stack_top = ($stackp & ~4095) + 4096
                set var $stack_bot = ($stackp & ~4095)

                set $stackp = *($stackp)
                while (($stackp < $stack_top) && ($stackp > $stack_bot))
                        set var $addr = *($stackp + 4)
                        info symbol $addr
                        set $stackp = *($stackp)
                end

                set $next_th=(((char *)$next_t->pids[1].pid_list.next) -
$pid_off)
                while ($next_th != $next_t)
                        set $next_th=(struct task_struct *)$next_th
                        printf "\npid %d; comm %s:\n", $next_t.pid, $next_t.comm
                        printf "==================\n"
                        set var $stackp = $next_t.thread.esp
                        set var $stack_top = ($stackp & ~4095) + 4096
                        set var $stack_bot = ($stackp & ~4095)

                        set $stackp = *($stackp)
                        while (($stackp < $stack_top) && ($stackp > $stack_bot))
                                set var $addr = *($stackp + 4)
                                info symbol $addr
                                set $stackp = *($stackp)
                        end
                        set $next_th=(((char *)$next_th->pids[1].pid_list.next)
- $pid_off)
                end
                set $next_t=(char *)($next_t->tasks.next) - $tasks_off
        end
end
document btt
        dump all thread stack traces on a kernel compiled with
CONFIG_FRAME_POINTER
end

define btpid
        set var $pid = $arg0
        set $tasks_off=((size_t)&((struct task_struct *)0)->tasks)
        set $pid_off=((size_t)&((struct task_struct *)0)->pids[1].pid_list.next)
        set $init_t=&init_task
```

```
        set $next_t=(((char *)($init_t->tasks).next) - $tasks_off)
        set var $pid_task = 0

        while ($next_t != $init_t)
                set $next_t=(struct task_struct *)$next_t

                if ($next_t.pid == $pid)
                        set $pid_task = $next_t
                end

                set $next_th=(((char *)$next_t->pids[1].pid_list.next) -
$pid_off)
                while ($next_th != $next_t)
                        set $next_th=(struct task_struct *)$next_th
                        if ($next_th.pid == $pid)
                                set $pid_task = $next_th
                        end
                        set $next_th=(((char *)$next_th->pids[1].pid_list.next)
- $pid_off)
                end
                set $next_t=(char *)($next_t->tasks.next) - $tasks_off
        end

        printf "\npid %d; comm %s:\n", $pid_task.pid, $pid_task.comm
        printf "==================\n"
        set var $stackp = $pid_task.thread.esp
        set var $stack_top = ($stackp & ~4095) + 4096
        set var $stack_bot = ($stackp & ~4095)

        set $stackp = *($stackp)
        while (($stackp < $stack_top) && ($stackp > $stack_bot))
                set var $addr = *($stackp + 4)
                info symbol $addr
                set $stackp = *($stackp)
        end
end
document btpid
        backtrace of pid
end


define trapinfo
        set var $pid = $arg0
        set $tasks_off=((size_t)&((struct task_struct *)0)->tasks)
        set $pid_off=((size_t)&((struct task_struct *)0)->pids[1].pid_list.next)
        set $init_t=&init_task
        set $next_t=(((char *)($init_t->tasks).next) - $tasks_off)
        set var $pid_task = 0

        while ($next_t != $init_t)
                set $next_t=(struct task_struct *)$next_t

                if ($next_t.pid == $pid)
                        set $pid_task = $next_t
                end
```

```
                set $next_th=(((char *)$next_t->pids[1].pid_list.next) -
$pid_off)
                while ($next_th != $next_t)
                        set $next_th=(struct task_struct *)$next_th
                        if ($next_th.pid == $pid)
                                set $pid_task = $next_th
                        end
                        set $next_th=(((char *)$next_th->pids[1].pid_list.next)
- $pid_off)
                end
                set $next_t=(char *)($next_t->tasks.next) - $tasks_off
        end

        printf "Trapno %ld, cr2 0x%lx, error_code %ld\n",
$pid_task.thread.trap_no, \
                                $pid_task.thread.cr2,
$pid_task.thread.error_code

end
document trapinfo
        Run info threads and lookup pid of thread #1
        'trapinfo <pid>' will tell you by which trap & possibly
        address the kernel panicked.
end


define dmesg
        set $i = 0
        set $end_idx = (log_end - 1) & (log_buf_len - 1)

        while ($i < logged_chars)
                set $idx = (log_end - 1 - logged_chars + $i) & (log_buf_len - 1)

                if ($idx + 100 <= $end_idx) || \
                   ($end_idx <= $idx && $idx + 100 < log_buf_len)
                        printf "%.100s", &log_buf[$idx]
                        set $i = $i + 100
                else
                        printf "%c", log_buf[$idx]
                        set $i = $i + 1
                end
        end
end
document dmesg
        print the kernel ring buffer
end
```