Solaris kernel thread stack usage

Context: this document proposes to add a new `kmem_stackinfo` variable to Solaris kernel for a new feature that gives informations about the kernel thread stack usage through `mdb`/`kmdb`. This new variable and a new kmdb/mdb command (`::stackinfo`) allow a user to see the current and maximum usage (a high water mark) of a kernel thread stack using a display with percentage and an history of the same data for dead threads (a circular log of threads that have exercice their kernel stack the most).

About kernel threads: a Solaris kernel thread stack size is provided at kernel thread creation time (as an argument to Solaris kernel `thread_create()` function, a default size is provided if no specific size is asked). Note that kernel stacks of dead kernel threads (zombies threads are created by `thread_exit()`) are cached for potential reuse and finally freed later by the `thread_reaper` kernel thread. Kernel thread stacks are managed by `segkp` segment driver, they can be swapped out when memory goes very low. Note that a kernel thread stack is created for each process LWP.

As a kernel thread stack size is fixed at a kernel thread creation time, this proposed new feature helps developers/testers/system administrators to answer the following questions:

- How much of a kernel thread stack is really used during a kernel thread life ?
- How close is a kernel thread (or 'was' for a dead thread) from a kernel stack overflow panic ?

The mechanism to show how much kernel thread stack is really used by a kernel thread is simple: at thread creation time, the kernel thread stack is filled with a specific pattern (`0xbadcbadcbadcbadc`, instead of zero filled) if the `kmem_stackinfo` variable is set to a value different than 0. During the kernel thread execution, the kernel thread stack pattern is progressively overwritten. This feature takes in account hardware architecture (stack grows either to lower or larger addresses). A simple count from stack top until pattern is not found (or stack botom is reached) gives a high water mark value, the maximum kernel stack space used by a kernel thread. This allows:

- To compute the percentage of kernel thread stack really used (a high water mark) for current kernel threads in the system
- When a kernel thread ends, we can log to a small circular memory buffer the last kernel threads that have used the most their kernel thread stacks before dying

Proposed is a new Solaris mdb/kmdb command: ::stackinfo

```
> ::help stackinfo

NAME
  stackinfo - display kthread_t stack usage

SYNOPSIS
  [ addr ] ::stackinfo

DESCRIPTION
     -a show also TS_FREE kthreads
     -h print history (dead kthreads)

ATTRIBUTES

  Target: kvm
  Module: genunix
  Interface Stability: Unstable

>
```

For example, when a Solaris system (in examples, its a Solaris 10 sparcv9 system) has been booted with `set kmem_stackinfo = 0x1` in `/etc/system`, the mdb/kmdb `::stackinfo -h` (history) command shows the last 20 dead kernel threads that have used their kernel thread stack the most:

```
> ::stackinfo -h
Freed threads stack usage history:
          THREAD            STACK   SIZE  MAX CMD/LWPID or STARTPC
0000030002f2d900 000002a10317c000   5ae0  49% sched/1
...
000002a100413cc0 000002a10040e000   5ad0  40% 10c6718 (mt_config_thread)
>
```

A kernel thread named `mt_config_thread` at kernel address `0x2a100413cc0` had a kernel thread stack allocated at kernel address `0x2a10040e000` of size `0x5ad0` and had used a maximum of 40% of its kernel thread stack when this kernel thread ended. The process `sched` LWP `1` has used its kernel thread stack at maximum of 49% when when this kernel thread ended. Note that the kernel thread adresses shown by the `::stackinfo -h` command must not be used in any other command, as these kernels threads do no longer exists in the system.

SIZE is not shown, for example, as `0x6000`, but `0x5ad0`. This is mainly because `kthread_t` data itself is also 'on the stack', SIZE is the effective size available when a thread starts (see `usr/src/uts/common/disp/thread.c`, function `thread_create()`). SIZE is computed as `(t_stk - t_stkbase)` or `(t_stkbase - t_stk)` depending the architecture, because of the following code in `thread_create()`:

```
#ifdef STACK_GROWTH_DOWN
                t->t_stk = stk + stksize;
                t->t_stkbase = stk;
#else
                t->t_stk = stk;                 /* 3b2-like */
                t->t_stkbase = stk + stksize;
#endif /* STACK_GROWTH_DOWN */
```

Example:

```
> ffffff00065f9c80::stackinfo
          THREAD            STACK   SIZE  CUR  MAX CMD/LWPID
ffffff00065f9c80 ffffff00065f5000   4c80   2%  16% taskq_d_thread()
> ffffff00065f9c80::print kthread_t t_stk
t_stk = 0xffffff00065f9c80
> ffffff00065f9c80::print kthread_t t_stkbase
t_stkbase = 0xffffff00065f5000
>
```

0xffffff00065f9c80 - 0xffffff00065f5000 = 0x4c80

To examine current kernel threads stack usage:

```
> ::stackinfo
          THREAD            STACK   SIZE  CUR  MAX CMD/LWPID
000000000180e000 0000000001800000   bae0   1%  n/a sched/1
000002a10001fcc0 000002a10001a000   5ad0   1%   5% idle()
000002a100017cc0 000002a100012000   5ad0   2%  14% taskq_thread()          T
000002a10015fcc0 000002a10015a000   5ad0   2%  15% streams_bufcall_service()
0000030264cbe9e0 000002a100b56000   5ad0   4%  44% java/1
000003002becba20 000002a1007f8000   5ad0   4%  47% init/1
000002a100667cc0 000002a100662000   5ad0   2%  15% pageout_scanner()
...
```

The `pageout_scanner` kernel thread (`0x2a100667cc0`) is currently at 2% of its kernel thread stack but has used up to 15% of its kernel thread stack since creation (note that the `::stackinfo` commands are `STACK_BIAS` aware).

```
000002a100667cc0::findstack
stack pointer for thread 2a100667cc0: 2a100667121
[ 00000  cv_wait+0x38() ]
  000002a1006671d1 pageout_scanner+0x1a4()
  000002a1006672d1 thread_start+4()
```

(add `7ff` (`STACK_BIAS`) for real stack addresses shown on Solaris >= 10)

```
thread_start+4(): 2a1006672d1 +  7ff = 2a100667ad0
STACK + SIZE    : 2a100662000 + 5ad0 = 2a100667ad0
cv_wait+0x38()  : 2a100667121 +  7ff = 2a100667920
(2a100667ad0 - 2a100667920) / 5ad0 = 2%
```

If we dump `2a100662000` to `2a100667ad0`, we found continuously the pattern `0xbadcbadcbadcbad` up to `2a100666c80`, so MAX is

```
100 * (2a100667ad0 - 2a100666c80) /  5ad0)) = 15 %
```

Note that `MAX` can be shown as `n/a` for the first kernel thread as the first Solaris kernel thread stack is not allocated by Solaris kernel `thread_create()` function.

To examine a specific kernel thread stack usage:

```
> 30000ea1c40::stackinfo
          THREAD            STACK   SIZE  CUR  MAX CMD/LWPID
0000030000ea1c40 000002a1007f8000   5ae0   4%  32% init/1
>
```

It is also possible to show kernel threads stack usage for threads even if they are in `TS_FREE` state using the `-a` option (for 'all'), note that the MAX value can than be shown as `n/a` for some kernel threads that have never been executed, for example the kernel threads used for interrupts are in `TS_FREE` state while not operating:

```
> ::stackinfo -a
...
000002a10000fcc0 000002a10000a000   5ad0   1%  n/a thread_create_intr()
000002a100007cc0 000002a100002000   5ad0   1%  n/a thread_create_intr()
000002a10003fcc0 000002a10003a000   5ad0   1%  n/a thread_create_intr()
000002a100037cc0 000002a100032000   5ad0   1%  n/a thread_create_intr()
000002a10002fcc0 000002a10002a000   5ad0   1%  n/a thread_create_intr()
000002a100027cc0 000002a100022000   5ad0   1%  n/a thread_create_intr()
000002a10005fcc0 000002a10005a000   5ad0   1%  n/a thread_create_intr()
000002a100057cc0 000002a100052000   5ad0   1%  n/a thread_create_intr()
000002a10004fcc0 000002a10004a000   5ad0   1%  11% thread_create_intr()
000002a100047cc0 000002a100042000   5ad0   1%  28% thread_create_intr()
...
```

(this system has 1 CPU, so 10 interrupt threads, only 2 interrupts threads have been in use).

There is also a std (Statically Defined Tracing) kernel Dtrace probe, that can be used to have more details about kernel stack usage of 'dead' threads using for example, the following anonymous script called 'stackinfo.d':

```
#!/usr/sbin/dtrace -s
#pragma D option quiet

kthread_t *t;

sdt:genunix:thread_log_stk_usage_free:stack-usage
/ ((kthread_t *)arg0)->t_tid != 0/
{
        t = (kthread_t *)arg0;
        @a1[((t->t_procp)->p_user).u_comm] = lquantize(arg2, 0, 100, 10);
}

sdt:genunix:thread_log_stk_usage_free:stack-usage
/ ((kthread_t *)arg0)->t_tid == 0/
{
        t = (kthread_t *)arg0;
        @a2[t->t_startpc] = lquantize(arg2, 0, 100, 10);
}

END
{
        printa("%a %@8u\n", @a2);
}
```

With `kmem_stackinfo` not zero in `/etc/system`):

```
# dtrace -AFs stackinfo.d;reboot
```

On reboot

```
# dtrace -ae
```

shows:

```
...
genunix`mt_config_thread
           value  ------------- Distribution ------------- count
             < 0 |                                         0
               0 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 11
              10 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 5
              20 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 11
              30 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 10
              40 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 1
              50 |                                         0
...
```

```
$ mkd -k
::stackinfo -h
...
000002a100227ca0 000002a100222000   5a90  46% 113a3a4 (mt_config_thread)
...
```

or for example:

```
#!/usr/sbin/dtrace -s
#pragma D option quiet

kthread_t *t;

sdt:genunix:thread_log_stk_usage_free:stack-usage
{
        t = (kthread_t *)arg0;
        @a["all"] = lquantize(arg2, 0, 100, 10);
}
```

that shows:

```
-  all
           value  --------             < 0 |
    0

               0 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@          11705
              10 |@@@@@@                                    2255
              20 |@@                                        718
              30 |@                                         261
              40 |                                          40
              50 |                                          1
              60 |                                          0
```

NOTES:

- Stack SIZE is shown as kthread_t (`t_stk - t_stkbase`) or (`t_stkbase - t_stk`) depending the architecture.
- When `kmem_stackinfo` variable is 0 (the default value) `::stackinfo -h` command is not available, and the `::stackinfo [-a]` shows `n/a` for the MAX value.
- If `kmem_stackinfo` variable is set/unset dynamically, the MAX value may be shown as `100%`, depending on the value of the bit when thread was created. This feature is consistent when `kmem_stackinfo` is set at boot time and not changed dynamicaly.
- Setting `kmem_stackinfo` as an effect on performances. With a value different than 0, kernel threads creation and kernel threads destruction are slower. In average, the overhead to create a kernel thread is 3 times slower, the overhead to handle the end of a kernel thread is two times slower.
- This proposed feature is not useless because of Dtrace (an anonymous dtrace script tracking kernel stack usage will less precise because forcibly using a profile probe) and this new feature can be ported to Solaris 8/9 where there is no dtrace.
- Don't use this feature to make default stack size smaller than Solaris default one

This feature has been tested against Solaris 10u4 (sparc, x86) and SunCluster 3.2 (that creates thousand of kernel threads). It can be easily backported to older Solaris releases.

ANNEXE: Implementation

We show here the simple changes done for this new feature in the `genunix` module (we do not describe the `mdb`/`kmdb` changes). Two files are modified for this feature in the genunix module:

In `usr/src/uts/common/sys/thread.h`, we add

```
/*
 * mdb stackinfo feature, high water mark for stack usage.
 */
#define       KMEM_LOG_STK_USAGE_PATTERN      0xbadcbadcbadcbadcULL
#define       KMEM_LOG_STK_USAGE_SIZE         20
#define       KMEM_LOG_STK_STR_SIZE           64

typedef struct kmem_log_stk_usage {
       caddr_t kthread;
       caddr_t t_startpc;
       caddr_t start;
       size_t  stksz;
       size_t  percent;
       id_t    t_tid;
       char    cmd[KMEM_LOG_STK_STR_SIZE];
} kmem_log_stk_usage_t;
```

In `usr/src/uts/common/disp/thread.c`, we add

```
/*
 * mdb stackinfo feature, high water mark for stack usage
 */
unsigned int            kmem_stackinfo;         /* stackinfo feature on-off */
kmem_log_stk_usage_t    *log_stk_usage_buf;     /* stackinfo circular log */
kmutex_t                log_stk_lock;           /* protects log_stk_usage_buf */

/*
 * forward declarations for mdb stackinfo feature
 */
static void thread_log_stk_usage_create(kthread_t *);
static void thread_log_stk_usage_free(kthread_t *);
static size_t stk_compute_percent(caddr_t, caddr_t, caddr_t);
```

and `thread_create` function has 3 lines added (in bold, the code added)

```
kthread_t *
thread_create(...
...
        kthread_t *t;
...
#endif /* STACK_GROWTH_DOWN */
        }

        if (kmem_stackinfo != 0) {
                thread_log_stk_usage_create(t);
        }
        /* set default stack flag */
        if (stksize == lwp_default_stksize)
                t->t_flag |= T_DFLTSTK;
...
```

and `thread_exit()` function has 3 lines added (in bold, the code added)

```
void
thread_exit()
{
      ...
      if (kmem_stackinfo != 0) {
```

```
            thread_log_stk_usage_free(t);
        }

        t->t_state = TS_ZOMB;         /* set zombie thread */
```

and following new functions are added:

```
static void
thread_log_stk_usage_create(kthread_t *t)
{
        caddr_t        start;  /* stack start */
        caddr_t        end;    /* stack end   */

        /*
         * Stack grows up or down, see thread_create(),
         * compute stack memory area start and end.
         */
        if (t->t_stk > t->t_stkbase) {
                start = t->t_stkbase;
                end = t->t_stk;
        } else {
                start = t->t_stk;
                end = t->t_stkbase;
        }

        /* alignement check */
        if (((uintptr_t)start & 0x7) != 0 || ((uintptr_t)end & 0x7) != 0) {
                return;
        }
        /* if the stack size is larger than a meg, assume that it's bogus */
        if ((end - start) > (1024 * 1024)) {
                return;
        }

        /* fill up stack with a pattern (instead of zeros) */
        while (start < end) {
                *(uint64_t *)start = KMEM_LOG_STK_USAGE_PATTERN;
                start += sizeof (uint64_t);
        }
}


static void
thread_log_stk_usage_free(kthread_t *t)
{
        caddr_t start;
        caddr_t end;
        caddr_t ptr;
        size_t stksz;
        size_t smallest = 0;
        size_t percent = 0;
        uint_t index = 0;
        uint_t i;
        static size_t smallest_percent = (size_t)-1;
        static uint_t full = 0;

         mutex_enter(&log_stk_lock);
         if (log_stk_usage_buf == NULL) {
                 log_stk_usage_buf = (kmem_log_stk_usage_t *)
                     kmem_zalloc(KMEM_LOG_STK_USAGE_SIZE *
                     (sizeof (kmem_log_stk_usage_t)), KM_NOSLEEP);
                 if (log_stk_usage_buf == NULL) {
                         mutex_exit(&log_stk_lock);
                         return;
                 }
         }
         mutex_exit(&log_stk_lock);

        /*
         * Stack grows up or down, see thread_create(),
         * compute stack memory aera start and end.
         */
        if (t->t_stk > t->t_stkbase) {
                start = t->t_stkbase;
                end = t->t_stk;
        } else {
                start = t->t_stk;
                end = t->t_stkbase;
        }

        /* stack size */
        stksz = end - start;
```

```
/* if the stack size is larger than a meg, assume that it's bogus */
if (stksz > (1024 * 1024)) {
        return;
}

/* search until no pattern in the stack */
if (t->t_stk > t->t_stkbase) {
        ptr = start;
        while (ptr < end) {
                if (*(uint64_t *)ptr != KMEM_LOG_STK_USAGE_PATTERN) {
                        percent = stk_compute_percent(end, start, ptr);
                        break;
                }
                ptr += sizeof (uint64_t);
        }
} else {
        ptr = end - sizeof (uint64_t);
        while (ptr > start) {
                if (*(uint64_t *)ptr != KMEM_LOG_STK_USAGE_PATTERN) {
                        percent = stk_compute_percent(start, end, ptr);
                        break;
                }
                ptr -= sizeof (uint64_t);
        }
}

 DTRACE_PROBE3(stack__usage, kthread_t *, t,
     size_t, stksz, size_t, percent);

 if (percent == 0) {
         return;
 }

mutex_enter(&log_stk_lock);

if (full == KMEM_LOG_STK_USAGE_SIZE && percent < smallest_percent) {
        /*
         * The log is full and already contains the highest values
         */
        mutex_exit(&log_stk_lock);
        return;
}

/* keep a log of the highest used stack */
for (i = 0; i < KMEM_LOG_STK_USAGE_SIZE; i++) {
        if (log_stk_usage_buf[i].percent == 0) {
                index = i;
                full++;
                break;
        }
        if (smallest == 0) {
                smallest = log_stk_usage_buf[i].percent;
                index = i;
                continue;
        }
        if (log_stk_usage_buf[i].percent < smallest) {
                smallest = log_stk_usage_buf[i].percent;
                index = i;
        }
}

if (percent >= log_stk_usage_buf[index].percent) {
        log_stk_usage_buf[index].kthread = (caddr_t)t;
        log_stk_usage_buf[index].t_startpc = (caddr_t)t->t_startpc;
        log_stk_usage_buf[index].start = start;
        log_stk_usage_buf[index].stksz = stksz;
        log_stk_usage_buf[index].percent = percent;
        log_stk_usage_buf[index].t_tid = t->t_tid;
        log_stk_usage_buf[index].cmd[0] = '\0';
        if (t->t_tid != 0) {
                stksz = strlen((t->t_procp)->p_user.u_comm);
                if (stksz >= KMEM_LOG_STK_STR_SIZE) {
                        stksz = KMEM_LOG_STK_STR_SIZE - 1;
                        log_stk_usage_buf[index].cmd[stksz] = '\0';
                } else {
                        stksz += 1;
                }
                (void) memcpy(log_stk_usage_buf[index].cmd,
                    (t->t_procp)->p_user.u_comm, stksz);
        }
        if (percent < smallest_percent) {
                smallest_percent = percent;
        }
}
mutex_exit(&log_stk_lock);
```

```
}


static size_t
stk_compute_percent(caddr_t t_stk, caddr_t t_stkbase, caddr_t sp)
{
        size_t percent;
        size_t s;

        if (t_stk > t_stkbase) {
                percent = t_stk - sp + 1;
                s = t_stk - t_stkbase + 1;
        } else {
                percent = sp - t_stk + 1;
                s = t_stkbase - t_stk + 1;
        }
        percent = ((100 * percent) / s) + 1;
        if (percent > 100) {
                percent = 100;
        }
        return (percent);
}
```