Deadlocks and Hangs

Crash Dump Analysis 2014/2015
Overview

• **Deadlock**
  - “Cycle in resource waiting chain”
  - Coffman conditions
  - Various resources: mutexes, rwlocks, condition variables, implicit resources

• **Hang**
  - “No forward progress”
  - Using deadman timer
More formal definition

Configuration in which two or more *activities* uninterruptibly block waiting for *resources* held by the others in the blocking chain

- **Activities** can be processes, threads, interrupts
- **Resources** can be synchronization primitives, but also generic resources
Coffman conditions

Necessary conditions for deadlock

1) One **resource** can be owned by only one **activity** at a time
2) An **activity** can request additional **resources** even if it already owns some
3) A **resource** cannot be forcibly revoked from an **activity**
4) A cycle exists in the **activity-resource** waiting chain
Deadlock example

P1:
lock(A); ✔
lock(B); CreateTime

P2:
lock(B); ✔
lock(A); CreateTime
Protection against race conditions

- Usually figure as resources in deadlocks
- In Solaris
  - Mutexes
  - Readers-Writer Locks
  - Condition Variables
Mutual exclusion for critical sections

- Solaris kernel: kmutex_t
- ::mutex

```c
mutex_enter(&pidlock);
retval = p->p_pgrp;
mutex_exit(&pidlock);
```

```plaintext
> fffffff02e10356e0::mutex
    ADDR   TYPE         HELD MINSPL OLDSPL WAITERS
    fffffff02e10356e0 adapt fffffff02d5848980   -   -     no
```
Readers-Writer Lock

- Critical sections for multiple readers or one writer
  - Solaris kernel: krwlock_t
  - ::rwlock

```c
rw_enter(&nvf_list_lock, RW_READER);
rval = nvlist_lookup_nvlist(nvf_list, id, &list);
rw_exit(&nvf_list_lock);

rw_enter(&nvf_list_lock, RW_WRITER);
rval = nvlist_add_uint32(nvf_list, id, value);
rw_exit(&nvf_list_lock);
```

```c
> ffffff00e93ece80::rwlock
  ADDR   OWNER/COUNT FLAGS      WAITERS
  fff00e93ece80  ffffff00f1947b20 B100
    WRITE_LOCKED ------+
```
Condition Variables

- **Waiting for a condition to become true**
  - The condition is indicated by `cv_signal()` or `cv_broadcast()`
  - Condition tested and changed under the protection of a mutex
  - Solaris kernel: `kcondvar_t`
  - `::wchaninfo`

```c
mutex_enter(&as->a_contents);
while (AS_ISCLAIMGAP(as))
    cv_wait(&as->a_cv, &as->a_contents);
AS_SETCLAIMGAP(as);
mutex_exit(&as->a_contents);
```

> fffffff00e8cc6dfa::wchaninfo -v
 ADDR   TYPE   NWAITERS   THREAD     PROC
 fffffff00e8cc6dfa cond 1: fffffff00e91aa0a0 Xorg

9
What runs in the system?

- Crash dumps taken on a deadlocked or hung system may not exhibit the culprit directly
  - Need to look further and deeper
    - ::cpuinfo
    - ::threadlist
    - ::findstack
    - Find arguments on the stack or use WCHAN as shown by ::threadlist
```
> ::cpuinfo -v
ID ADDR             FLG NRUN BSPL PRI RNRN KNRNRN SWITCH THREAD           PROC
 0 ffffffffbc34aa0  1b    1   10  -1   no    no t-3     ffffff0002805c80 (idle)

RUNNING <---+    |    |    ---> PIL THREAD
READY         |          10 ffffff00028c5c80
EXISTS         |           5 ffffff00028bfc80
ENABLE         |

+--> PRI THREAD           PROC
 60 ffffff0002e30c80 sched

> ffffff00028c5c80::threadlist -v
ADDR             PROC              LWP CLS PRI            WCHAN
fffffffff00028c5c80 ffffffffbc29c30                0   0 109 ffffffffbc6340
PC: resume_from_intr+0xb4    THREAD: unix`thread_create_intr()
stack pointer for thread ffffffff00028c5c80: ffffffff00028c59a0
[ ffffffff00028c59a0 resume_from_intr+0xb4() ]
  swtch+0x90()
  turnstile_block+0x75b()
  mutex_vector_enter+0x261()
  clock+0x64f()
```
Interpretation of WCHAN

- Various means
  - Using ::whatis
  - Guessing the type from the stack trace
  - Need to investigate what is the holder doing

```
> ffffffffbbcd6340::whatis
ffffffffbcb6340 is tod_lock+0 in genunix's bss
> ffffffffbbcd6340::mutex
   ADDR  TYPE   HELD MINSPLO OLDSPL WAITERS
fffffffbbcd6340 adapt ffffffff0e91aa0a0 - - yes
```
Useful queries

- Is someone waiting on e.g. a rwlock?
  - ::threadlist -v ! less
  - /rw_enter

- ::findlocks
  - Can detect wait cycles
  - Needs ::typegraph
  - “nota bene: locks may be held”
Deadlock appearance

- A deadlocked system will either

  (a) Crash because the kernel detects the cycle in the waiting chain

  (b) Appear hung and unresponsive
      - Eventually crash due to deadman timer, if the lbolt variable does not change

  (c) Appear working, if the resources involved in the deadlock are not vital
Dealing with hangs

- **Goal: Force the system to crash**
  - In order to find the culprit in the crash dump
  - It may be illustrative to explore the hung system using kmdb before forcing the crash dump
    - Using breakpoints and binary search to find the top-level function which loops (if any)
      - The only option if the hang occurs too early before a dump can be generated
**Binary search on a stack trace**

1. Break into kmdb
2. $C$
3. Pick the return address in the middle of the stack trace
4. :c

- If the breakpoint **was hit**, clear all breakpoints (:z) and repeat the search on the lower half of the stack trace
- If the breakpoint **was not hit**, clear all breakpoints (:z) and repeat the search on the upper half of the stack trace

★ It is possible that the stack trace starts with the top-level function. In that case, try to put a breakpoint to a function called from it and see if it gets called
Enforcing crash dump

- If you can still use the system shell
  - `hald -d`
  - `reboot -d`
  - `uadmin 5 1`

- If kmdb is loaded and you can break into it (F1+A, Stop+A, Ctrl+] se)
  - `$<systemdump`

- If you can break into OBP prompt on SPARC (Stop+A, Ctrl+] se)
  - `sync`
Enforcing crash dump (2)

- **Using a button**
  - NMI/XIR buttons on server machines
  - Three times the power button

- **Deadman timer**
Deadman timer

- Periodic activity
  - Wakes up each second and monitors the system lbolt variable
  - Needs to be enabled in /etc/system
    - set snooping=1
  - If lbolt does not change for a pre-configured amount of time (default is 50 s), the system dump is generated