# Hardware Breakpoint implementation in BCC


Manali Shukla <manashuk@cisco.com>

Aanandita Dhawan <aadhawan@cisco.com>

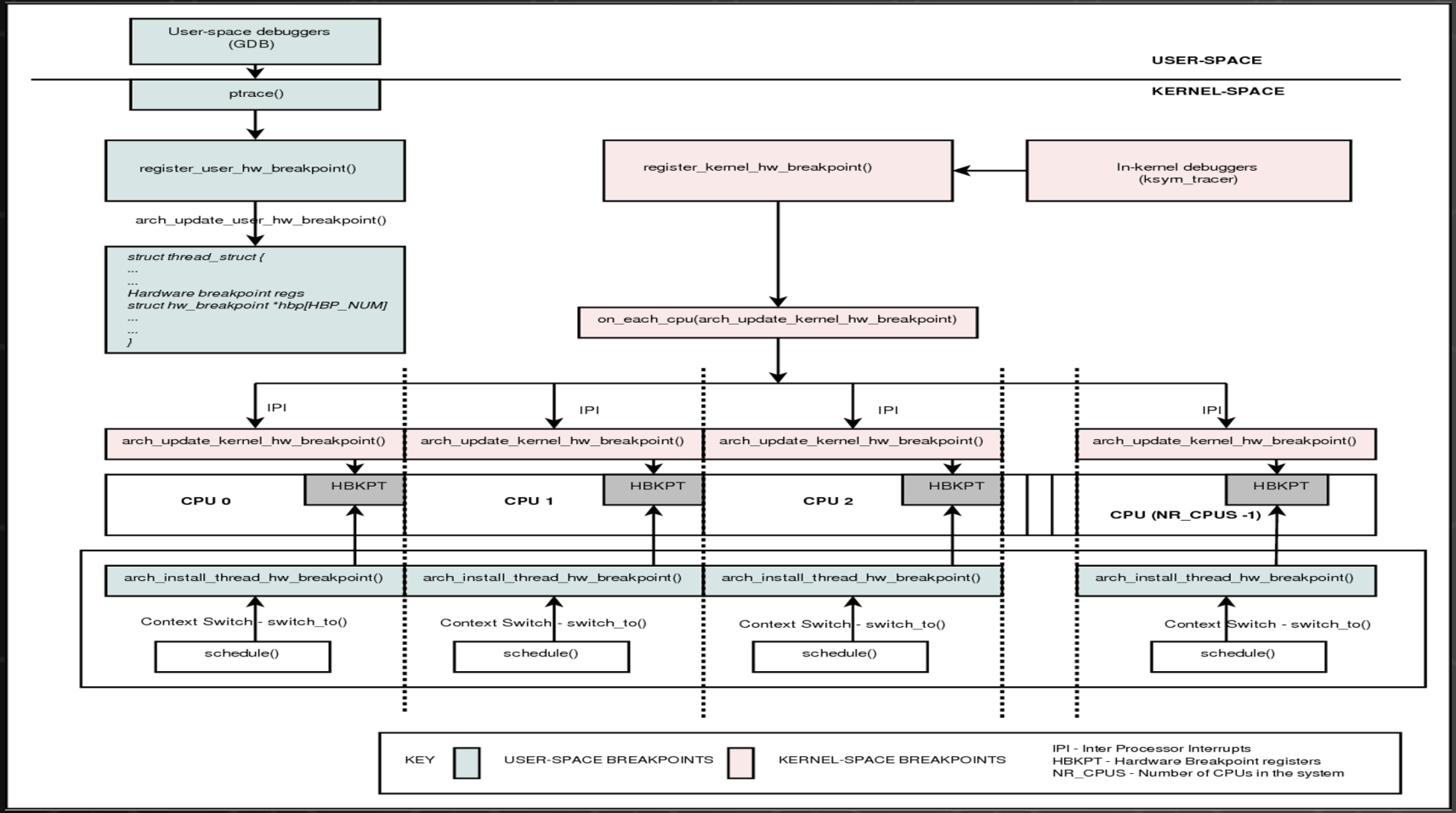
Maneesh Soni <manesoni@cisco.com>

October 28, 2020

01

# Hardware breakpoint

- Memory watchpoint
- Used in debuggers
- Elegant mechanism to monitor memory access
- Perf hardware breakpoint implementation:
  mem:<addr>[:access] [Hardware breakpoint]
  Example: perf stat -e mem:0xffffffffbb65f478:rw




https://www.kernel.org/doc/ols/2009/ols2009-pages-149-158.pdf

02

# Implementation

*libbpf.c*

```
void bpf_attach_breakpoint(uint64_t symbol_addr, int pid, int progfd, int bp_type) {
  struct perf_event_attr attr = {};

  memset(&attr, 0, sizeof(attr));
  attr.size = sizeof(attr);
  attr.type = PERF_TYPE_BREAKPOINT;
  attr.bp_len = HW_BREAKPOINT_LEN_4;
  attr.bp_addr = symbol_addr;
  attr.bp_type = (__u32)bp_type;
  attr.sample_period = 1;
  attr.precise_ip = 2; // request synchronous delivery
  attr.wakeup_events = 1;

  int i, nr_cpus = sysconf(_SC_NPROCESSORS_CONF);

  for (i=0; i<nr_cpus; i++) {
    check_on_each_cpu(i, &attr, progfd, pid);
  }
}

…
…
enum bpf_prog_type prog_type = BPF_PROG_TYPE_PERF_EVENT;
int efd = syscall(__NR_perf_event_open, attr, pid, cpu, -1, PERF_FLAG_FD_CLOEXEC);
if (efd < 0) {
  printf("event fd %d err %s\n", efd, strerror(errno));
  return;
}
```

# 03 Usage

***breakpoint.py***

```
bpf_text = """
#include <linux/sched.h>
#include <uapi/linux/ptrace.h>
struct stack_key_t {
  int pid;
  char name[16];
  int user_stack_id;
  int kernel_stack_id;
};
BPF_STACK_TRACE(stack_traces, 16384);
BPF_HASH(counts, struct stack_key_t, uint64_t);
```

```
int func(struct pt_regs *ctx) {
  struct stack_key_t key = {};
  key.pid = bpf_get_current_pid_tgid() >> 32;
  bpf_get_current_comm(&key.name, sizeof(key.name));
  key.kernel_stack_id = stack_traces.get_stackid(ctx, 0);
  key.user_stack_id = stack_traces.get_stackid(ctx,
                                               BPF_F_USER_STACK);
  u64 zero = 0, *val;
  val = counts.lookup_or_init(&key, &zero);
  (*val)++;
  bpf_trace_printk("Hello, World! Here I accessed am
address!\\n");
  return 0;
}
"""
b = BPF(text=bpf_text)
symbol_addr = input()
pid = input()
bp_type = input()

b.attach_breakpoint(symbol_addr, pid, "func", bp_type)
```

# 04 Output

```
#/home/manashuk/Documents# kernel.pid_max = 4194304

[1]+ Done sysctl kernel.pid_max
```

***Output***

```
root@ubuntu:/home# python
breakpoint.py
0xffffffffaa2623b0
-1
3
Kernel Stack :
scan_block
scan_gray_list
kmemleak_scan
kmemleak_scan_thread
kthread
ret_from_fork

User Stack :
- kmemleak (161)
```

05

# To-do list

- Incorporate comments
  - Add len as part of user parameter
  - Test `check_on_each_cpu()` is required or not ?
- symbols -> symbol address
  - manual right now
  - Can it be made as part of implementation?

Any Questions ???