User Interrupts – A faster way to signal

Sohil Mehta
sohil.mehta@intel.com
Goals

• Motivation
• Use cases
• Software architecture
• Feedback / Opens
• Next steps
Motivation

• Deliver events directly to user space – bypass kernel
• Overcome limitations of existing IPC and I/O events
  • Synchronous, Asynchronous, Polling.
  • Signals, pipes, RPCs, hardware notifications, etc.
• Super fast and efficient alternative

First hardware implementation – x86
  • Intel processor code-named Sapphire Rapids
Sources of User Interrupts

- Receiver infrastructure is common
- Now - User to User IPC - SENDUIPI
  - RFC patches are out
- Soon - Kernel to User
  - Patches in development
- Later - Other sources
  - e.g. devices

https://lore.kernel.org/lkml/20210913200132.3396598-1-sohil.mehta@intel.com/
Use cases

- General purpose fast IPC
- User mode schedulers
- Event dispatch for I/O stacks
  - e.g. User space networking
- Abstractions
  - Libevent, Liburing, etc.
- Other suggestions?

Relative IPC Latency (normalized to User IPI)

9x performance improvement

*details in backup
Software design

- Opt in – via system calls
- Sender – notifies the event
  - User application, kernel, others
- Receiver – waiting for events
  - User application
- Connection setup
  - File descriptor based
  - Closest equivalent is eventfd()
Receiver APIs

- Register User Interrupt Handler
  ```c
  int uintr_register_handler(handler_func, flags);
  int uintr_unregister_handler(flags);
  ```

- Create an fd representing the vector - priority
  ```c
  uintr_fd = uintr_create_fd(vector, flags);
  ```

- Share Connection with sender
  - Any existing fd exchange mechanism via inheritance or sharing via socket domain
Sender APIs

- Sender Connection Management using FDs
- Receive FD via inheritance or UNIX domain sockets
  ```c
  uipi_handle = uintr_register_sender(uintr_fd, flags);
  int uintr_unregister_sender(uintr_fd, flags);
  ```
- x86 Instruction
  ```c
  void _senduipi(uipi_handle)
  ```
User IPI Sample

Receiver
void __attribute__ ((__interrupt__)) u_handler(struct __uintr_frame *frame, unsigned long vector) {
    write(STDOUT_FILENO, “User Interrupt!\n”, 16);
    uintr_received = 1;
}
int main(int argc, char *argv[]) {
    int vector = 0;
    pthread_t pt;

    ret = uintr_register_handler(u_handler, 0);
    uintr_fd = uintr_create_fd(vector, 0);
    _stui();
    printf("Receiver enabled interrupts\n");

    pthread_create(&pt, NULL, &sender_thread, NULL)
    /* Do some other work */
    while(!uintr_received);

    pthread_join(pt, NULL);
    close(uintr_fd);
    uintr_unregister_handler(0);
    exit(EXIT_SUCCESS);
}

Sender
void *sender_thread(void *arg) {
    int uipi_handle;

    uipi_handle = uintr_register_sender(uintr_fd, 0);

    printf("Sending IPI from sender thread\n");
    _senduipi(uipi_handle);

    uintr_unregister_sender(uintr_fd, 0);
    return NULL;
}

Compiler – GCC 11

gcc -muintr -mgeneral-reg-only uipi_sample.c -lpthread -o uipi_sample

Output
$ ./uipi_sample
Receiver enabled interrupts
Sending IPI from sender thread
User Interrupt!
Feedback

- Syscall and FD based approach
- Other architectures

Opens

- Interrupting blocking system calls like sleep(), read(), etc.
- Common `uipi_handle` across threads (and processes)
- Kernel page table isolation support
- Many more..
Next steps?

Any suggestion is appreciated!

Please review UINTR patches on LKML.
https://lore.kernel.org/lkml/20210913200132.3396598-1-sohil.mehta@intel.com/
THANK YOU

Special Thanks!
Ashok Raj, Dave Hansen, Jacob Pan Jun, Tony Luck.
Backup
HW arch key elements

- **UPID** - Posted Interrupt descriptor
  - Pending interrupt vectors
  - Notification state – ON, SN.
  - Routing information
- **UITT** - Interrupt Target Table
  - UPID pointer
  - Vector information
- **SENDUIPI <UITT index>**
- Interrupt posting
  - Senduipi updates UPID
  - Generate notification interrupt
- Notification processing
  - Recognize interrupt
  - Move to UIRR
- Interrupt delivery
  - Push vector, invoke handler
  - Return from handler - Uiret
x86 Instructions

- `senduipi <uipi_handle>` – send a user IPI to a target task based on the UITT index.
- `uiret` – Return from a User Interrupt handler
- `clui` – Mask user interrupts by clearing UIF (User Interrupt Flag).
- `stui` – Unmask user interrupts by setting UIF.
- `testui` – Test current value of UIF.
Kernel API for User Notification

- Exchange uintr_fd with kernel agent
- Post user interrupt from the kernel
  
  ```c
  int uintr_notify(int uintr_fd);
  ```
- Kernel identifies target, vector from uintr_fd
IPC Performance

9x performance improvement with User IPI (running)

Original workload: https://github.com/goldsborough/ipc-bench
Updated workload: https://github.com/intel/uintr-ipc-bench
Config: 1M ping-pong IPC notifications with message size=1

Results have been estimated based on internal tests with:
Kernel: Linux v5.14.0 + User IPI patches
System: Internal pre-production platform

*Performance varies by use, configuration and other factors.