UCORE Porting on MIPS32S CPU

Chen Yuheng
Tsinghua Unv.

November 26, 2013

1 Introduction

This report gives a brief introduction to our project – *UCORE* porting onto the MIPS32S platform, including the kernel, limited device drivers as well as the user-lib. *UCORE* is an experimental modern operating system that includes a basic memory manager, a limited process/ thread scheduler, linux-like VFS and incomplete POSIX-compatible syscall interface. Currently, the following platforms are supported:

- *Mipssim in Qemu*, which is simulated with qemu-system-mipsel. The hardware configuration is available in qemu's source code, several important components of which is summarized in Table. 1.

Component	Type	Base Address	IRQ
Uart	16450	0x1fd003f8	4
Timer	On Chip	–	7
PIC	On Chip	–	–

Table 1: Mipssim Hardware Configuration

- *Our FPGA MIPS32S Implementation* hardware platform.

Our code is available on Github [1] and is kept updating. Any bug reports are welcomed.

2 Architecture of the MIPS32S Platform

3 Development Environment

This section is a guide to setup and use the cross-compile development environment for Mips.

[1] https://github.com/chyh1990/ucore-thumips

3.1 Toolchain

We use the standard toolchain GCC for Mips to compile ucore, even on our MIPS32S CPU, in which only a subset of MIPS1 Instruction Set is implemented. The best place to get a free GCC-MIPS toolchain is on CodeSourcery [2], just make sure you download GCC 4.6. The package also includes gdb for MIPS.

Another way to get a toolchain is compiling it from source code. GCC-core 4.6.3+Binutils 2.22 is tested. Compiling GCC is tricky, just google it if you come into any issues. Here is my configuration for GCC on 64-bit Ubuntu 12.04:

```
../gcc-4.6.3/configure --prefix=/home/guest/cpu/build-gcc/mips_gcc
  --target=mips-sde-elf --disable-nls --enable-languages=c
  --disable-multilib --without-headers --disable-shared
  --without-newlib --disable-libgomp -disable-libssp
  --disable-threads --disable-libgcc
```

3.2 Simulator

You can run ucore-thumips with the standard Qemu in your Linux distribution:

```
qemu-system-mipsel -M mipssim -m 32M -serial stdio
  -kernel obj/ucore-kernel-initrd
```

But we recommend to use our modified Qemu to simulate our simplified MIPS Instruction Set and Flash support.

what's more, it will be necessary to setup the mips-sde-elf-gdb properly, refer to the following *.gdbinit* example:

```
set endian little
set mipsfpu none
target remote 127.0.0.1:1234
file obj/ucore-kernel-initrd
```

3.3 MIPS32S Programming Guide

MIPS32S supports a simpified MIPS32 Intruction Set, here is several important differences:

1. No *lh/sh*
2. No *divu*
3. No *add/sub*, only *addu/subu* is supported.

We use some tricks to solve these problems, see *kern/thumips.h*.

Another problem is that MIPS32S does NOT support delayed slot. Solving this problem by giving GCC options correctly:

```
CFLAGS :=  -EL -G0 -fno-delayed-branch -Wa,-O0
```

An example MIPS32S C project can be found in test1.tar.gz.

[2] https://sourcery.mentor.com/GNUToolchain/release2189

4 Source Code Organization

This section introduces the source code orginazation of ucore-thumips and explains several configuration options.

4.1 Source Tree

Since our work is based on LAB0-LAB8, important directories are listed in Table. 2.

Directory	Description
debug	debug console after a kernel panic
driver	device driver interface definition
include	useful macros for MIPS32S
fs	Filesystem
init	kernel entry point and initialization code
libs	utilities
mm	low-level memory management
process	context switch
sync	atomic operation
syscall	MIPS-specific syscall machanism
trap	exception handling

Table 2: ucore-thumpis directories

4.2 Makefile

UCore does not have a configuration system yet, all configuration is hand-coded in the Makefile.

1. *GCCPREFIX*, toolchain path;
2. *USER_APPLIST*, the applications to be included.

More detailed memory layout is defined in memlayout.h in the corresponding machine directory, see Section. 5.3.

5 Implementation Details

This section describes the implementations of some important mechanism in the MIPS32S CPU, which is similar to the standard MIPS32 CPU. Thus, I refer to Harvard's OS/161[1] project during writing the code.

5.1 Booting

The booting process is also simulated. Our Qemu needs the following files:
There is two ways to use our modified Qemu to boot ucoer for MIPS.
Then, the C enironment must be setup:

- *sp*, setup the stack

File	Type	Description
ucore-kernel-initrd	ELF	ucore kernel with ramdisk
flash.img	Binary	used for flash simulation, can be a link to the kernel
boot/loader.bin	Binary	BIOS, knows ELF and load kernel from Flash
thumips_insn.txt	Text	Instruction Set Descriptor

Table 3: Qemu Files

Offset	Size	Usage
0x00000000	0x80000000	Userspace, TLB mapped
0x80000000	0x20000000	Kernel Space, Direct mapping
0xA0000000	0x20000000	IO Space, Direct Mapping

Table 4: Bootable Kernel Memory Layout

- *gp*, set to _gp (defined in ldscript)
- zero .bss section

For convenience, root filesystem image(ramdisk) is linked with the kernel, appending at the end of the *.data* section. [3].

5.2 Exception Handling

The most important hardware support for exception/interrupt handling on MIPS32S is the *SR* register:

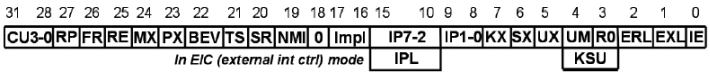

Figure 1: Status Register[2]

- ERL, MUST be clear by software after reset;
- EXL and KSU, MUST be set properly to handle nested exceptions.

The kernel use a trapframe structure to save registers when privilege mode transition occurs:

The trapframe is constructed by the assembly in trap/exception.S, be aware with the fact that registers *k0* and *k1* is reserved by the compiler so we can use them in the exception handler without saving them.

See *kern/trap/exception.S* for more details.

5.3 Memory Management

MIPS32S has no MMUs, but has a programmable TLB unit. So it is possible to emulate MMU utilizing the TLB miss exception. The software simulated

[3]see tools/kernel.ld

Algorithm 1 Trapframe

```
/* $1 - $30 */
struct pushregs {
 uint32_t reg_r[30];
};

/*
 * Structure describing what is saved on the stack during entry to
 * the exception handler.
 *
 * This must agree with the code in exception.S.
 */

struct trapframe {
  uint32_t tf_vaddr;  /* coprocessor 0 vaddr register */
  uint32_t tf_status; /* coprocessor 0 status register */
  uint32_t tf_cause;  /* coprocessor 0 cause register */
  uint32_t tf_lo;
  uint32_t tf_hi;
  uint32_t tf_ra; /* Saved register 31 */
  struct pushregs tf_regs;
  uint32_t tf_epc;  /* coprocessor 0 epc register */
};
```

MMU of MIPS32S is similar to X86's, Ucore uses the following configuration (which is similar to X86's VA layout) :

PDT Index	PTE Index	Offset
10	10	12

Table 5: Virtual Address in ucore

In our TLB miss handler, we first checkout whether it is just a TLB miss or a page table miss by looking the address up in our emulated X86 page table. If it is a TLB miss, check the access permission and fill it up (or raise a access violation). If it is a page miss, call the do_pgfault. See Alg. 2.

5.4 Context Switch

According to MIPS32 O32 ABI, we must save the following registers during the context switching:

Algorithm 3 Context

```
struct context {
  uint32_t sf_s0;
  uint32_t sf_s1;
  uint32_t sf_s2;
  uint32_t sf_s3;
  uint32_t sf_s4;
  uint32_t sf_s5;
  uint32_t sf_s6;
  uint32_t sf_s7;
  uint32_t sf_s8;
  uint32_t sf_gp;
  uint32_t sf_ra;
  uint32_t sf_sp;
};
```

According to O32 ABI, *s0-s8* must be reserved by the callee, *gp* is a global pointer, *ra* is the interrupted PC. In addition, we must switch the kernel stack, which is saved in *sf_sp*.

5.5 System Call

The syscall mechanism is borrowed from Linux2.6. In MIPS, a special instruction *syscall* handles user mode to supervisor mode transition.

In ucore, we employ the following system calling convention:

1. *a0* - *a3*, arguments(from left to right)
2. *v0*, syscall number
3. *syscall*
4. return value in *v0*

Syscall is wrapped in user mode library and can be called as a normal C function.

5.6 User Library and Application

The code in *user* should works without modification. However, since *libs-user-ucore/syscall.c* is not compatible with MIPS's system calling convetion, we use some macros from Linux to create our own system call entries. See Alg. 4.

Another modification is *user/libs/user.ld*. The .text section of user application is located at 0x10000000.

6 Conclusion

In our project, we work out a basically working version of ucore for MIPS32S. However, just as what items listed on the TODO list imply, much work remains for making ucore for MIPS32S pratically usable operating system.

References

[1] David A. Holland, Ada T. Lim, and Margo I. Seltzer. A new instructional operating system. *SIGCSE Bull.*, 34(1):111–115, February 2002.

[2] Dominic Sweetman. *See MIPS Run, Second Edition.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

Algorithm 2 User Mode System Calling Convetion

```
/* use software emulated X86 pgfault */
static void handle_tlbmiss(struct trapframe* tf, int write)
{
  int in_kernel = trap_in_kernel(tf);
  assert(current_pgdir != NULL);
  uint32_t badaddr = tf->tf_vaddr;
  int ret = 0;
  pte_t *pte = get_pte(current_pgdir, tf->tf_vaddr, 0);
  if(pte==NULL || ptep_invalid(pte)){   //PTE miss, pgfault
    //tlb will not be refill in do_pgfault,
    //so a vmm pgfault will trigger 2 exception
    //permission check in tlb miss
    ret = pgfault_handler(tf, badaddr, get_error_code(write, pte));
  }else{ //tlb miss only, reload it
    /* refill two slot */
    /* check permission */
    if(in_kernel){
      tlb_refill(badaddr, pte);
      return;
    }else{
      if(!ptep_u_read(pte)){
        ret = -1;
        goto exit;
      }
      if(write && !ptep_u_write(pte)){
        ret = -2;
        goto exit;
      }
      tlb_refill(badaddr, pte);
      return ;
    }
  }

exit:
  if(ret){
    print_trapframe(tf);
    if(in_kernel){
      panic("unhandled pgfault");
    }else{
      do_exit(-E_KILLED);
    }
  }
  return ;
}
```

Algorithm 4 User Mode System Calling Convetion

```
static inline int
syscall(int num, ...) {
    va_list ap;
    va_start(ap, num);
    uint32_t arg[MAX_ARGS];
    int i, ret;
    for (i = 0; i < MAX_ARGS; i ++) {
        arg[i] = va_arg(ap, uint32_t);
    }
    va_end(ap);

    num += SYSCALL_BASE;

    asm volatile(
      ".set noreorder;\n"
      "move $v0, %1;\n" /* syscall no. */
      "move $a0, %2;\n"
      "move $a1, %3;\n"
      "move $a2, %4;\n"
      "move $a3, %5;\n"
      "syscall;\n"
      "nop;\n"
      "move %0, $v0;\n"
      : "=r"(ret)
      : "r"(num), "r"(arg[0]), "r"(arg[1]), "r"(arg[2]), "r"(arg
          [3])
      : "a0", "a1", "a2", "a3", "v0"
    );
    return ret;
}
```