Advanced Operating Systems

(263-3800-00L)

Timothy Roscoe
Pravin Shinde
Herbstsemester 2014

http://www.systems.ethz.ch/courses/fall2013/AOS
Week 4:
AOS Coding Tutorial
(263-3800-00L)

Timothy Roscoe
Pravin Shinde
Herbstsemester 2014

http://www.systems.ethz.ch/courses/fall2014/AOS
Goal of this session

- Help you with milestone-2, ...
- by introducing some key concepts!
  - Resource management, communication
- Lot of API and code examples.
  - Ask questions if you have any confusion!
  - I don't promise I can answer all questions, but I will try.

You have been warned!
Outline

- Milestone 2 Overview
- Capabilities
- Error handling
- How to get physical memory
- Threads
- Local message-passing
- Stack-ripped C and Asynchronous C
Milestone 2:

- New code base
  - Full Barrelfish CPU driver
- Implement self-paging system
  - Each program responsible for handling its own page faults
  - Design user-level representation of current state of HW page tables
  - Virtual address allocation
- Demonstrate a process taking and handling page faults
  - init_memtest program
Why should you care about capability system?

- It's a mechanism of “resource management” in Barrelfish
  - Memory
  - Communication
  - Scheduling

- It will be useful to know how they work!
- We focus on “How to use them?”
  - More details, theory and other capability flavours in next week's lecture
Capabilities in Barrelfish

- “keys” that allow domain to access resource
- Cannot be forged
- Have types (e.g. Frame, Page Table)
- Stored in hierarchical capability space (CSpace)
  - comprised of capabilities of type CNode.
- Defined in domain-specific language (DSL)
Capability invocations

- Each type supports a set of operations
  - Think OO classes and methods
  - Most “system calls” are actually capability operations
- Restrictions on operations enforce invariants
  - E.g. It should be impossible to construct a bad page table from user space
Capability invocation example

```c
static inline errval_t
invoke_frame_identify(struct capref frame,  
                              struct frame_identity *ret)
{
    struct sysret sysret = cap_invoke1(frame, FrameCmd_Identify);

    assert(ret != NULL);
    if (err_is_ok(sysret.error)) {
        ret->base = sysret.value & (~BASE_PAGE_MASK);
        ret->bits = sysret.value & BASE_PAGE_MASK;
        return sysret.error;
    }
    ret->base = 0;
    ret->bits = 0;
    return sysret.error;
}
```
Capability management interface

- Capability types are defined in capabilities/caps.hl
  - The C types are prefixed with `ObjType_`
  - e.g. type `Frame` in the definition will be `ObjType_Frame` in C
- User space operates on `struct caprefs`
- `slot_alloc()` – allocates a slot in its cspace
- `cap_copy()` – create a copy of a capability
- `cap_retype()` – change the type (and size) of capability
- `cap_destroy()` – delete copy of a capability and free the cspace slot it was in
- The full interface can be found in `include/barreelfish/capabilities.h`
Capability management example

er rval t frame alloc(struct capref *dest, size_t bytes, size_t *retbytes)
{
    errval_t err = slot alloc(dest);
    if (err is fail(err)) {
        return err push(err, LIB_ERR_SLOT_ALLOC);
    }
    assert(bytes > 0);
    uint8_t bits = log2ceil(bytes);
    assert((1UL << bits) >= bytes);
    if (bits < BASE_PAGE_BITS) {
        bits = BASE_PAGE_BITS;
    }
    struct capref ram;
    err = ram alloc(&ram, bits);
    if (err is fail(err)) { // Error handling omitted
        err = cap retype(*dest, ram, ObjectType_Frame, bits);
        if (err is fail(err)) { // Error handling omitted
            err = cap destroy(ram);
            if (err is fail(err)) { return err; }
            if (retbytes != NULL) { *retbytes = 1UL << bits; }
            return SYS_ERR_OK;
        }
    }
}
Error Handling

- DSL for errors
- Makes it easy to define errors without having to deal with error numbers directly
- Helper functions generated by DSL compiler
  - `err_is_ok()` – returns true for success error codes
  - `err_is_fail()` – returns true for failure error codes
  - `err_push()` – allows stacking of multiple error codes in single 32 bit error value
  - `err_no()` – get last pushed error code from error value
  - `err_getstring()` – returns textual representation of error code
Error definition code example

- in errors/errno.fugu

```c
// errors for AOS
errors aos AOS_ERR_ {
    success OK             "Successful",
    failure OUT_OF_MEMORY  "Could not get memory while handling page fault",
    failure INVALID_ADDR   "Fault Address outside valid range",
    failure SLOT_ALLOC     "Error in slot_alloc",
};
```
void my_slot_alloc(struct capref *s) {
    errval_t err;
    err = slot_alloc(s);

    if (err_is_ok(err)) {
        debug_printf("got a slot\n");
        return AOS_ERR_OK; // success error code
    }

    if (err_is_fail(err)) {
        debug_printf("error in slot_alloc: %s\n", err_getstring(err));
        return err_push(err, AOS_ERR_SLOT_ALLOC);
    }
}
Memory allocation

- Physical memory is represented by Frame and DevFrame capabilities
- Physical memory (and page tables) can be mapped by invoking VNode_Map on a higher-level page table
- Each page table level has its own capability type
  - ex: VNode_ARM_l1 and Vnode_ARM_l2
Memory allocation interface

- `ram_alloc()` – allocates a RAM capability in the given slot
- `frame_alloc()` – allocates a Frame capability
- `arml2_alloc()` – allocates a ARM L2 page table
- `vnode_create()` – creates a VNode in a given slot
- `cnode_create()` – creates a CNode
- `vnode_map()` – the invocation for modifying entries in the hardware page tables
Pagetable allocation code example

- This code is included in the handout

```c
static errval_t arml2_alloc(struct capref *ret) {
    errval_t err;
    err = slot_alloc(ret);
    if (err_is_fail(err)) {
        debug_printf("slot_alloc failed: %s\n",
                      err_getstring(err));
        return err;
    }
    err = vnode_create(*ret, ObjType_VNode_ARM_l2);
    if (err_is_fail(err)) {
        debug_printf("vnode_create failed: %s\n",
                      err_getstring(err));
        return err;
    }
    return SYS_ERR_OK;
}
```

This function calls slot_alloc(), cap_retype() with the given capability type and cap_destroy()
Memory allocation code example

- This will be the core of what you're asked to do in the next milestone
- Error and consistency checking omitted for brevity

```c
#define FLAGS (KPI_PAGING_FLAGS_READ | KPI_PAGING_FLAGS_WRITE)
void handle_fault(genvaddr_t vaddr)
{
    int l1_index = ARM_L1_USER_OFFSET(vaddr);
    int l2_index = ARM_L2_USER_OFFSET(vaddr);
    struct capref l1_table = (struct capref) {
        .cnode = cnode_page,
        .slot  = 0,
    };
    // would need checks if ptables exist already
    struct capref l2_table;
    arml2_alloc(&l2_table);
    vnode_map(l1_table, l2_table, l1_index, FLAGS, 0, 1);
    struct capref frame;
    size_t bytes = 4096u;
    frame_alloc(&frame, bytes, &bytes)
    vnode_map(l2_table, frame, l2_index, FLAGS, 0, 1);
}
```
vnode_map() restrictions

- Each copy of a Frame (and DevFrame) capability can only be mapped once
  - i.e. 1 call to vnode_map()
  - Kernel uses capabilities to keep track of memory mappings
- If you want to map different parts of a large Frame capability: create a copy each time.
Capability management: Code example 2

• Error handling omitted
• You might want to use this in your project

static struct capref current_frame;
errval_t get_frame(size_t bytes)
{
    struct capref ram;
    size_t alloc_bits;
    alloc_bits = log2floor(bytes);
    ram_alloc(&ram, alloc_bits);
    cslot_t slots_needed = bytes / BASE_PAGE_SIZE;
    cslot_t slots;
    if (slots_needed > 1) {
        /* get CNode and retype into it */
        struct capref nextcncap; struct cnoderef nextcn;
        cnode_create(&nextcncap, &nextcn, slots_needed, &slots);
        current_frame = (struct capref) { .cnode = nextcn, .slot = 0 };
    } else {
        slot_alloc(&current_frame);
    }
    cap_retype(current_frame, ram, ObjType_Frame, BASE_PAGE_BITS);
    cap_destroy(ram);
    return SYS_ERR_OK;
}
Threads

- Barrelfish has user-level threads
- You will need them in a few weeks time
- Each thread has its own exception handler to handle exceptions that happen while it's running
- **Keep this in mind when designing your self-paging module**
Threads interface

- thread_create() - create a new thread
- thread_join() - wait for a thread to finish executing
- In general, a “standard” threading interface
- thread_set_exception_handler() - set general purpose exception handler for the currently executing thread
Threads code example: Exception handler setup

```c
#define S_SIZE 8192
static char e_stack[S_SIZE];
static char *e_stack_top = e_stack + S_SIZE;

static void handler_func(enum exception_type type, int subtype,
                          void *addr,
                          arch_registers_state_t *regs,
                          arch_registers_fpu_state_t *fpuregs)
{
    // handle exceptions
}

void some_func(void) {
    thread_set_exception_handler(handler_func,
                                  NULL /* output param for old handler func */,
                                  e_stack, e_stack_top,
                                  NULL, NULL /* output params for old e-stack */);
}
```

You need to worry about stack for exception handling!
Threads code example: Worker threads

```c
struct worker_state { char c; size_t bufsize; }
static int worker_func(void *arg)
{
    struct worker_state *st = arg;
    char *buf = malloc(st->bufsize);
    for (int j = 0; j < st->bufsize; j++) {
        buf[j] = j % 256;
        if (j % (bufsize>>7) == 0) { putchar(st->c); }
    }
}

int main(void) {
    struct worker_state st1 =
        (struct worker_state) { .c = '.', .bufsize = 1ul*1024*1024 };
    struct worker_state st2 =
        (struct worker_state) { .c = '*', .bufsize = 8ul*1024*1024 };
    struct thread *t1 = thread_create(worker_func, st1);
    struct thread *t2 = thread_create(worker_func, st2);
    thread_join(t1, NULL); thread_join(t2, NULL);
    return 0;
}
```
Local message passing (later milestones)

- We use message passing to communicate between programs

- Passing a message on the same core is done by invoking a special **EndPoint capability** of the other program

- Usually done using a wrapper layer
  - Imp channels

- Possible to transmit one capability per message

- Messages get dispatched on **waitsets**
  - Each domain has a default waitset.
Local messaging interface

- `Imp_sendX()` – Send a message with X words of payload
- `Imp_chan_recv()` – receive a message, mostly used in callbacks
- `Imp_chan_register_recv()` – register a callback that's executed when the channel has new message(s) available
- `get_default_waitset()` – returns default waitset of domain
Local messaging: Send

- Error checking omitted

```c
errval_t send_message(struct lmp_chan *c, uintptr_t *msgbuf,
                      size_t msg_words, size_t *words_sent)
{
    uintptr_t buf[LMP_MSG_LENGTH];
    if (msg_words > LMP_MSG_LENGTH) {
        msg_words = LMP_MSG_LENGTH;
    }
    memcpy(buf, msgbuf, msg_words*sizeof(uintptr_t));
    memset(buf+msg_words, 0, (LMP_MSG_LENGTH-msg_words));
    if (words_sent) { *words_sent = msg_words; }
    return lmp_chan_send9(c, LMP_SEND_FLAGS_DEFAULT, NULL_CAP,
                          buf[0], buf[1], buf[2], buf[3], buf[4],
                          buf[5], buf[6], buf[7], buf[8]);
}
```

Quiz: why this?
Local messaging: Receive

- Error checking omitted
- Receive handler gets de-registered when its called, need to re-register after handling each message

```c
errval_t recv_handler(void *arg)
{
    struct lmp_chan *lc = arg;
    struct lmp_recv_msg msg = LMP_RECV_MSG_INIT;
    struct capref cap;
    err = lmp_chan_recv(lc, &msg, &cap);
    if (err_is_fail(err) && lmp_err_is_transient(err)) {
        // reregister
        lmp_chan_register(lc, get_default_waitset(),
                          MKCLOSURE(recv_handler, arg));
    }

    debug_printf("msg buflen %zu\n", msg.buf.msglen);
    debug_printf("msg->words[0] = 0x%lx\n", msg.words[0]);
    lmp_chan_register(lc, get_default_waitset(),
                      MKCLOSURE(recv_handler, arg));
}
```

```c
void some_func(void) {
    // assumption: we have channel here
    lmp_chan_register_recv(chan, get_default_waitset(),
                           MKCLOSURE(recv_handler, chan));
}
```
Local messaging: Dispatch

- This is a piece of code that's present in most Barrelfish applications

```c
int main_loop(struct waitset *ws)
{
    // go into messaging main loop
    while (true) {
        err = event_dispatch(ws);
        if (err_is_fail(err)) {
            DEBUG_ERR(err, "in main event_dispatch loop");
            return EXIT_FAILURE;
        }
    }
    return EXIT_SUCCESS;
}

int main(void)
{
    // do initialization
    // ...
    // run messaging loop on default waitset
    return main_loop(get_default_waitset());
}
```
Stack-ripped C

- Consequence of message passing IDC
- chains of callbacks that contain core logic of application
- unreadable if core logic is sufficiently complex
- commonly called “stack-ripping”
void Lookup(NSChannel_t *c, char *name) {
    OnRecvLookupResponse(c, &ResponseHandler);
    // Store state needed by send handler
    c->st = name;
    OnSend(c, &SendHandler);
}

void ResponseHandler(NSChannel_t *c, int addr) {
    printf("Got response %d\n", addr);
}

void SendHandler(NSChannel_t *c) {
    if (OnSendLookupRequest(c, (char *)(c->st)) == BUSY) {
        Re-set SendHandler if we couldn't send after all
        OnSend(c, &SendHandler);
    }
}

• We can do better!
AC: Composable Asynchronous IO for Native Languages

T. Harris, M. Abadi, R. Isaacs, R. McIlroy; OOPSLA 2011

• AC stands for “Asynchronous C”

• The idea in this paper is to extend C to make stack-ripped code look “sequential”

• new language features: async, do{}finish, cancel

• Goal: make low-level message-passing code scalable

• No multi-threading, the extensions just identify opportunities where multiple messages can be issued concurrently
Lookup Example: AC version

- from the paper

```c
// Caution: functions ending in AC may block
void LookupAC(NSChannel_t *c, char *name) {
    int addr;
    SendLookupRequestAC(c, name);
   RecvLookupResponseAC(c, &addr);
    printf("Got response %d\n", addr);
}

void TwinLookupAC(NSChannel_t *c1, NSChannel_t *c2, char *name) {
    do {
        async LookupAC(c1, name); // S1
        async LookupAC(c2, name); // S2
    } finish;
    printf("Got both responses\n"); // S3
}
```