Linux Kernel Development 04

工海系 b06501018 朱紹勳

	Linux Kernel Dev 3rd	Understand Linux Kernel 3rd	
Bootstrap		System Bootstrap	Linux system management handbook
Interprocess Communication		Chapter 19	Advance Programming in Unix Enivronment
Timing	Chapter 11	Chapter 6	
Memory Addressing		Chapter 2	

Bootstrap

Bootstrap

- BIOS(Basic Input / Output System)
- protected mode / real mode(√)
- Procedure
    (1) Run a series of tests on hardware.
    (2) Initializes the hardware devices, start kernel thread & init.
    (3) Searches OS to boot (access section in CD-ROM)
    (4) Copy the contents of to RAM, starting from physical address 0x00007c00,
    (5) Jump to that address and executes code.

Continue…

- Read the first 512 bytes of storage → Master Boot Record(MBR)
- MBR will tell which partition has the boot loader
- Boot loader store OS info, memory address
  - GNU GRUB:
    - deal with multiple os in same computer (dual boot selection)
- Run shell script
- Boot loader will help we load into linux kernel

- start_kernel():

(1) sched_init()
(2) build_all_zonelists() /* initialize memory zone */
(3) page_alloc_init() and mem_init() /* Buddy system allocators */
(4) trap_init() and init_IRQ /* initialize IDT */
(5) softirq_init() /* initialize TASKLET_SOFTIRQ */
(6) time_init() /* initialize time system*/
(7) kmem_cache_init() /* initial slab (memory management) */
(8) calibrate_delay() /* check CPU clock speed */
(9) process 1 is created by invoking the kernel_thread() /* init */

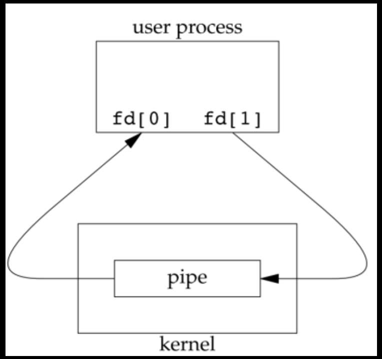
- **BIOS → MBR → boot loader → kernel → init process → login**
- **Root File system**

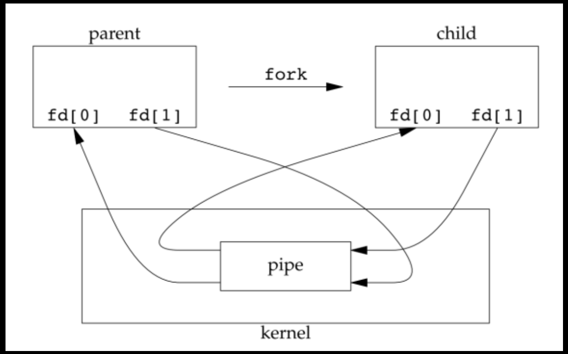
Interprocess Communication

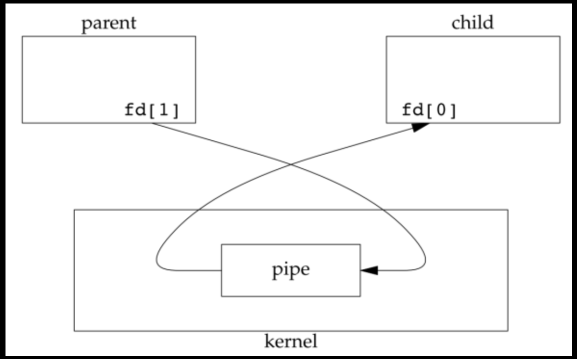
Interprocess Communication

- Mechanisms
  (1) Pipe
  (2) Semaphores
  (3) Messages
  (4) Shared memory regions
  (5) Socket

Pipe / FIFO







- `$ ls | more`
- one pipe has one direction flow.
- `pipe(), popen(), pclose()`

```
#include "apue.h"
int main(void){
	int n;
	int  fd[2];
	pid_t pid;
	char line[MAXLINE];
	if (pipe(fd) < 0) err_sys("pipe error");
	if ((pid = fork()) < 0) {err_sys("fork error");}
	else if (pid > 0) { /* parent */
		close(fd[0]);
		write(fd[1], "hello world\n", 12);
	} else { /* child */
		close(fd[1]);
		n = read(fd[0], line, MAXLINE);
		write(STDOUT_FILENO, line, n);
	}
	exit(0);
}
```

```
#include <stdio.h>
…
FILE *fp;
int status;
char path[PATH_MAX];
if(fp == NULL)
	/* Handle error*/
while (fgets(path, PATH_MAX, fp) != NULL)
	printf(“%s”, path);
status = pclose(fp);
if (status == -1){
	/* Error reported by pclose()*/
}else{
	/* Use macros described under wait() to inspect
“status” in orderto determine success/failure of
command executed by popen() */
}
```

System V IPC

- Linux system programming support both System V & POSIX
- System V is originated from AT&T
- Allow User Mode processes:
  - Synchronize processes by semaphores
  - send & recv message
  - Share a memory area

IPC Semaphore

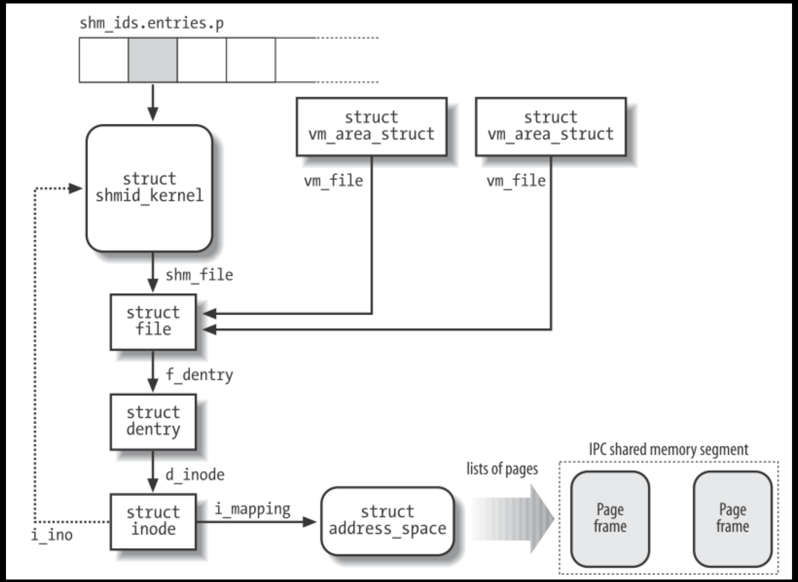
- Just like the semaphore mentioned in process chapter, is a counter
- semaphore control a shared resource, and limit the process to access
- a struct called `semid_ds`

```
int semget(key_t key, int nsems, int flag);
      // create &obtain a semaphore ID

int semctl(int semid, int semnum, int cmd, ... /* union semun arg */ );
      // semaphore initialization

int semop(int semid, struct sembuf semoparray[], size_t nops);
      // Related operation
```

IPC Shared Memory



- Mentioned in OS course, the fastest IPC ways

  Related to virtual file system

```
int shmget(key_t key, size_t size, int flag);
        // obtain a shared memory identifier
int shmctl(int shmid, int cmd, struct shmid_ds *buf );
        // various shared memory operations
void *shmat(int shmid, const void *addr, int flag);
        // process attaches it to its address space
int shmdt(const void *addr);
        // done with a shared memory segment, we call shmdt to detach it
void *mmap(void *addr, size_t len, int prot, int flag, int fd, off_t off );
        // map a context to the memory
```

POSIX Message Queue

- Message Queue is a linked list store in kernel
- POSIX interface seems simpler, better than old System V
- Other interface, like XSI

Function names	Description
`mq_open()`	Open (optionally creating) a POSIX message queue
`mq_close()`	Close a POSIX message queue (without destroying it)
`mq_unlink()`	Destroy a POSIX message queue
`mq_send()`, `mq_timedsend()`	Send a message to a POSIX message queue; the latter function defines a time limit for the operation
`mq_receive()`, `mq_timedreceive()`	Fetch a message from a POSIX message queue; the latter function defines a time limit for the operation
`mq_notify()`	Establish an asynchronous notification mechanism for the arrival of messages in an empty POSIX message queue
`mq_getattr()`, `mq_setattr()`	Respectively get and set attributes of a POSIX message queue (essentially, whether the send and receive operations should be blocking or nonblocking)

Timer & Time Management

- Purpose of timer:
  - interrupt, time slice
  - update run time & real time
  - Resource statistic

- Clock type:
  - Real Time Clock (always powered) : initialize xtime
  - System Timer - Programmable Interval Timer (PIT) : interrupt mechanism

- tick & HZ:
  - different default value in architecture, but programmable in <asm/param.h>.
  - 1000 HZ may be a good option

jiffies

- A counter variable, recording the total tick after starting. 0xfffb6c20
- defined in <linux/jiffies.h>
- overflow… `#define time_after, time_before, time_after_eq, timer_before_eq`
- Application:

  converts from jiffies to seconds:

```
(jiffies / HZ)
```

```
unsigned long timeout = jiffies + 5*HZ; /*time out in five seconds */
if (time_before(jiffies, timeout)) {
        /* not yet timeout */
}else{
        /* timeout */
}
```

Timer Interrupt : Every 1/1000 sec

- Related to architecture
  - xtime_lock, response & set sys clock, update real time clock, tick_periodic()
- Not related to architecture
  - resource statistic, scheduler_tick()…

```
static void tick_periodic(int cpu)
{
        if (tick_do_timer_cpu == cpu) {
                write_seqlock(&xtime_lock);

                /* Keep track of the next tick event */
                tick_next_period = ktime_add(tick_next_period, tick_period);

                do_timer(1);
                write_sequnlock(&xtime_lock);
        }

        update_process_times(user_mode(get_irq_regs()));
        profile_tick(CPU_PROFILING);
}
```

```
void do_timer(unsigned long ticks)
{
        jiffies_64 += ticks;
        update_wall_time();
        calc_global_load();
}
```

```
void update_process_times(int user_tick)
{
        struct task_struct *p = current;
        int cpu = smp_processor_id();

        /* Note: this timer irq context must be accounted for as well. */
        account_process_tick(p, user_tick);
        run_local_timers();
        rcu_check_callbacks(cpu, user_tick);
        printk_tick();
        scheduler_tick();
        run_posix_cpu_timers(p);
}
```

Real Time

- `<time/timekeeping.h>`
- `xtime`, 1970/1/1
- need seqlock for r&w
- `read_seqbegin()`
  `read_seqretry()`
- `update_times()`

---user mode---

- `gettimeofday()`
  - `do_gettimeofday()` – `sys`
- `settimeofday()`

Timer (dynamic, kernel)

- defined in <linux/timer.h>, manage kernel time, `struct timer_list`
- executes timers in bottom-half context
- Application:

```
void run_local_timers(void)
{
        hrtimer_run_queues();
        raise_softirq(TIMER_SOFTIRQ);   /* raise the timer softirq */
        softlockup_tick();
}
```

```
struct timer_list my_timer;
init_timer(&my_timer);
my_timer.expires = jiffies + delay;     /* timer expires in delay ticks */
my_timer.data = 0;                      /* zero is passed to the timer handler */
my_timer.function = my_function;        /* function to run when timer expires */
void my_timer_function(unsigned long data);
add_timer(&my_timer);
del_timer(&my_timer);
```

- Delaying Execution

  - Busy Looping

```
unsigned long timeout = jiffies + 2*HZ;        /* 2 sec */
while (time_before(jiffies, timeout)) cond_resched();
```

  - Small Delays

```
udelay(150);      /* delay for 150 µs */
ndelay(2);        /* delay for 2 ns */
```

  - `schedule_timeout()`

    put task to sleep until delay time has elapsed, then wakeup.

```
set_current_state(TASK_INTERRUPTIBLE);
schedule_timeout(s * HZ);
```

Memory Addressing

- Three types of memory address
  - (1) Logical Address

    used in machine code, segment + offset
  - (2) Linear(Virtual) Address

    32 bits (4GB) 0x00000000 – 0xffffffff

    PAE : 36bits

    64bit : 128TB…
  - (3) Physical Address

    RAM address
- Memory Translation Mode

  (1) Real Mode (2) Protected Mode

CPU Mode

(1) Real Mode :

Addresses in real mode always correspond to real locations in memory

In 8086 arch, people think 1MB RAM is large

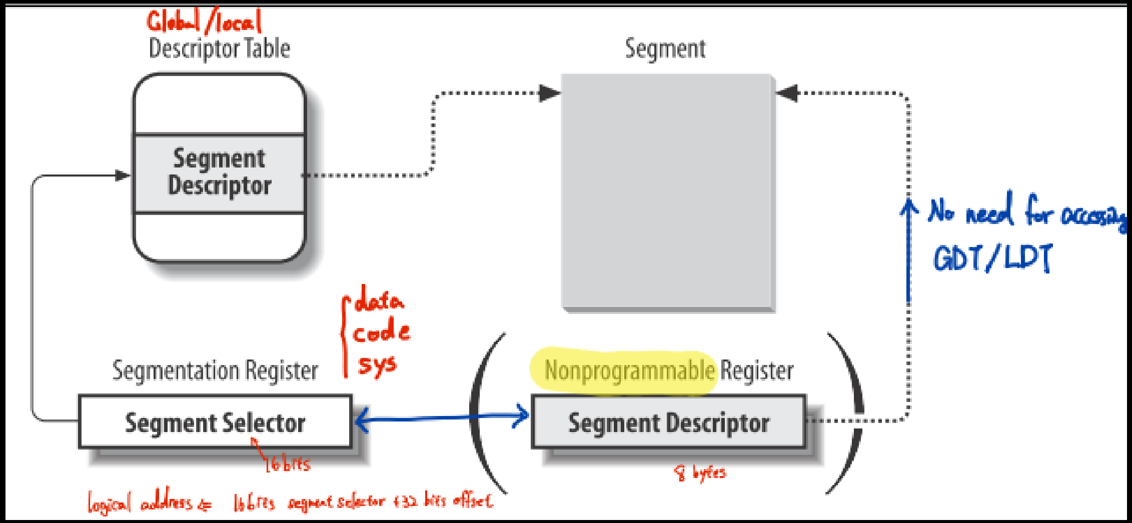
(2) Protected Mode

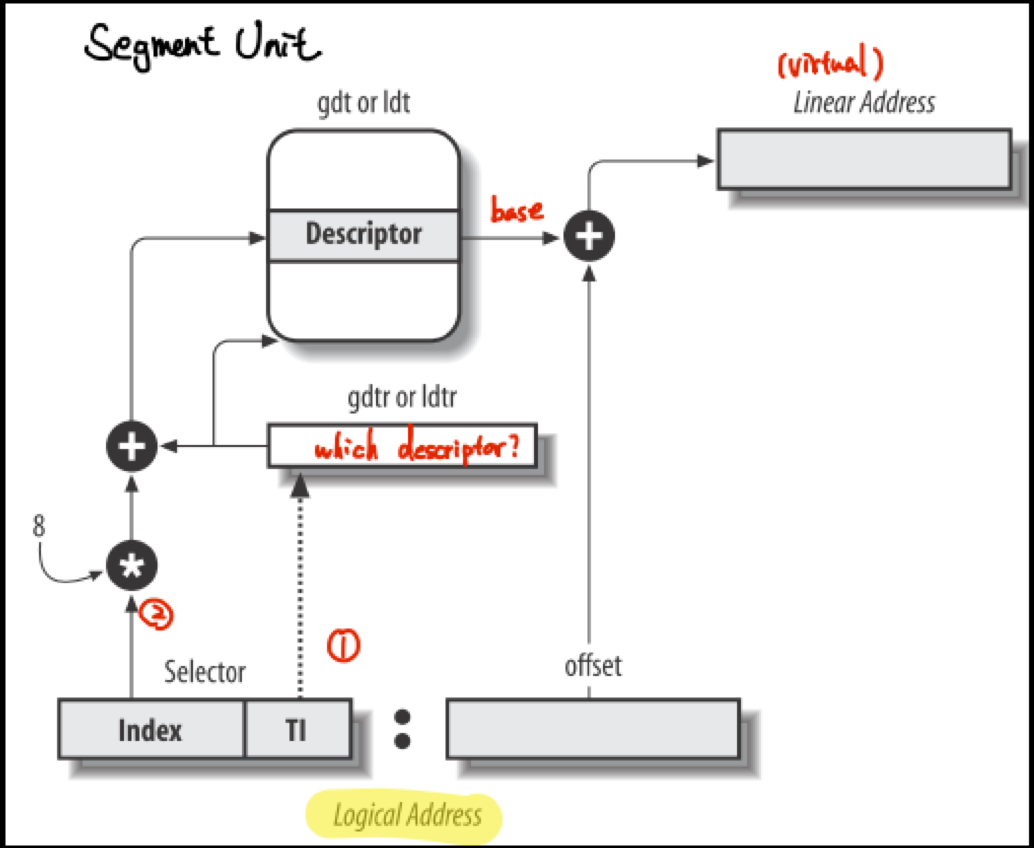
allows system software to use features such as virtual memory, paging and safe multi-tasking

(3) Virtual -8086 Mode

(4) System Management Mode

Hardware Segmentation

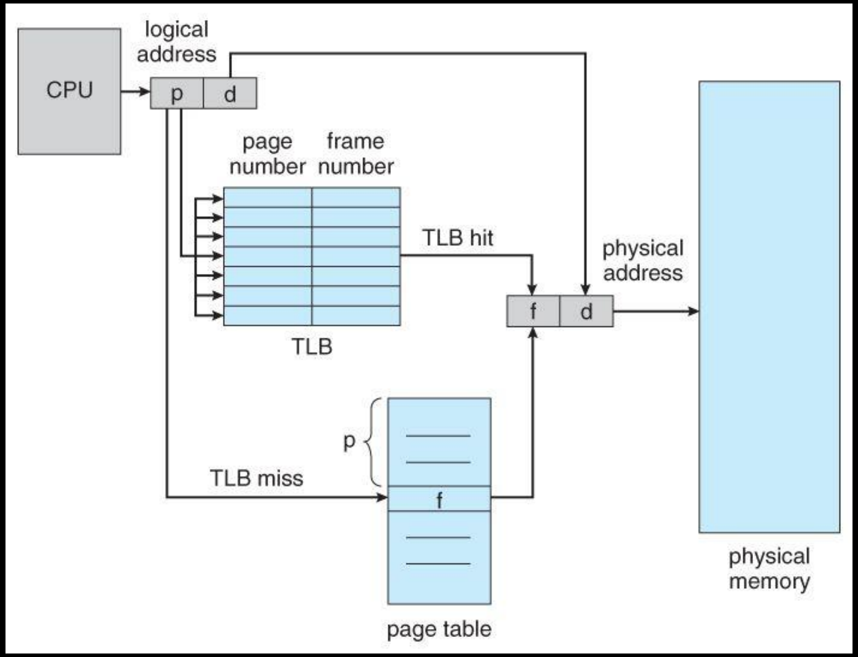




Hardware Paging

- Map virtual mem (page) to physical mem (page frame) Fail : Page Fault Exception
- For 32 bit : Directory(10) + Offset(22, 4MB) Physical address Extension
- For 64 bit : use 3 – 4 level paging

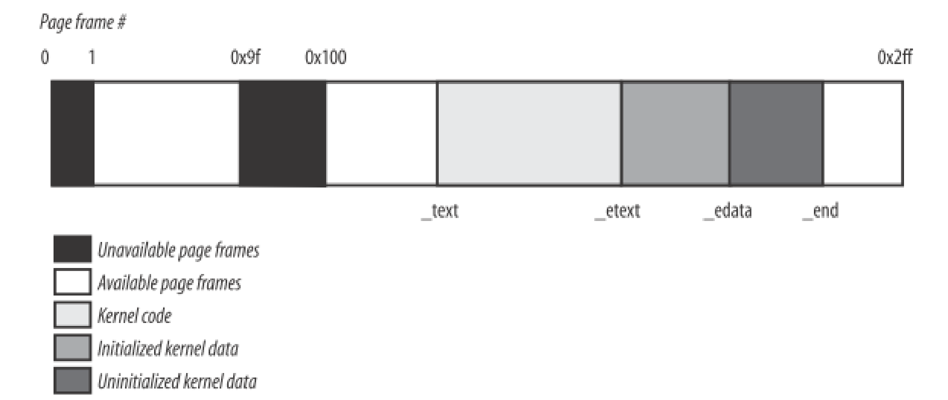
- CPU clock (3GHz) vs RAM (2666MHz) : cache(s)
- Translation Lookaside Buffer (TLB)
  - Speedup the translation of virtual mem
  - Store physical mem in TLB after first time access
  - Each CPU has its own TLB



Linux Paging

- Adapt 4 level Paging (ver2.6) : global, upper, middle, table
- defined Macros for page table handling
- Page frame in physical memory :
  - Every process has individual physical memory section
  - Page frame 0 is used for BIOS (0x000a0000 ~ 0x000fffff)
  - Linux kernel store start from 0x00100000(2nd RAM)

Variable name	Description
num_physpages	Page frame number of the highest usable page frame
totalram_pages	Total number of usable page frames
min_low_pfn	Page frame number of the first usable page frame after the kernel image in RAM
max_pfn	Page frame number of the last usable page frame
max_low_pfn	Page frame number of the last page frame directly mapped by the kernel (low memory)
totalhigh_pages	Total number of page frames not directly mapped by the kernel (high memory)
highstart_pfn	Page frame number of the first page frame not directly mapped by the kernel
highend_pfn	Page frame number of the last page frame not directly mapped by the kernel



Process Page Table

- Process virtual memory
  - 0x00000000 – 0xbfffffff
    addressing in user & kernel mode
  - 0xc0000000 – 0xffffffff
    addressing only in kernel mode
  - PAGE_OFFSET 0xc0000000

Kernel Page Table

- Maintain a page table in Master kernel Page Global Directory
- Initialize :
  - kernel create a limited address space
  - kernel use remaining space to create page table
- Provisional kernel Page Tables
  - Map virtual to physical (8MB)
  - When RAM > 4096MB

Handling Hardware Cache & TLB

- Hardware caches are addressed by cache lines
- Mapping of virtual to phys is determined by kernel, not hardware
- The timing of flushing TLB

  ex: context switch
  ex: LRU…

Method name	Description	Typically used when
flush_tlb_all	Flushes all TLB entries (including those that refer to global pages, that is, pages whose Global flag is set)	Changing the kernel page table entries
flush_tlb_kernel_range	Flushes all TLB entries in a given range of linear addresses (including those that refer to global pages)	Changing a range of kernel page table entries
flush_tlb	Flushes all TLB entries of the non-global pages owned by the current process	Performing a process switch
flush_tlb_mm	Flushes all TLB entries of the non-global pages owned by a given process	Forking a new process
flush_tlb_range	Flushes the TLB entries corresponding to a linear address interval of a given process	Releasing a linear address interval of a process
flush_tlb_pgtables	Flushes the TLB entries of a given contiguous subset of page tables of a given process	Releasing some page tables of a process
flush_tlb_page	Flushes the TLB of a single Page Table entry of a given process	Processing a Page Fault

TLB-invalidating methods of linux
Intel provide easier method

- For mult-processor, we use interprocessor interrupt to execute TLB-invalidating function
- Lazy – TLB
  the CPU remembers that its current process is running on a set of page tables whose TLB entries for the User Mode addresses are invalid.