Computer System Final Project
X-Part Toy-OS

Feng Yan* Zhao Xiaoran* Zhang Yunce*

*College of computer science and technology
Zhejiang University

June 2022

Table of Contents

1. Background
2. Buddy-system
3. System Call Table
4. File Loader
5. Shell
6. Conclusions and Thoughts

Table of Contents

1. Background
2. Buddy-system
3. System Call Table
4. File Loader
5. Shell
6. Conclusions and Thoughts

Background

Toy-OS

- Buddy-System
- Sys-call Table
- File Loader
- Shell

Table of Contents

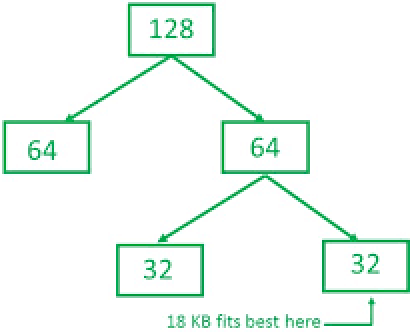
1. Background
2. Buddy-system
3. System Call Table
4. File Loader
5. Shell
6. Conclusions and Thoughts

Buddy-System

What's Buddy-system?

Buddy-System

Buddy-system



Buddy-System

How did I implement the buddy-system?

Buddy-System

How did I implement the buddy-system?

```c
#ifndef _BUDDY_H
#define _BUDDY_H

#include "types.h"
uint64 ROUNDUP(uint64 num);

struct buddy
{
  unsigned long size;
  unsigned *bitmap;
};

void init_buddy_system(void);
void *alloc_pages(int);
void free_pages(void *);

#endif
```

Buddy-System

Details

```c
void init_buddy_system(void)
{
    buddy_item.size = PHY_SIZE / PGSIZE;
    uint64 needPage = buddy_item.size * 2 * sizeof(unsigned int) / PGSIZE;
    unsigned int *begin = (unsigned int *)(U64ADDR(_end));

    buddy_item.bitmap = begin;
    begin[0] = buddy_item.size;

    for (int i = 1; i < 2 * buddy_item.size - 1; i++)
    {
        begin[i] = begin[(i - 1) / 2] / 2;
    }

    //申请需要的page
    needPage += (U64ADDR(_end) - U64ADDR(text_start)) / PGSIZE;
    alloc_pages(needPage);
}
```

Buddy-System

Details

```
假设我们系统一共有8个可分配的页面，可分配的页面数需保证是2^n
memory:
+--------+--------+--------+--------+--------+--------+--------+--------+
| page 0 | page 1 | page 2 | page 3 | page 4 | page 5 | page 6 | page 7 |
+--------+--------+--------+--------+--------+--------+--------+--------+
buddy system将页面整理为 2^n 2^(n-1) ... 2^1 2^0 不同大小的块:
8 +---> 4 +---> 2 +---> 1
  |       |       |
  |       |       +---> 1
  |       |
  |       +---> 2 +---> 1
  |               |
  |               +---> 1
  |
  +---> 4 +---> 2 +---> 1
          |       |
          |       +---> 1
          |
          +---> 2 +---> 1
                  |
                  +---> 1
```

Buddy-System

Details

```
#define P_PAGE 0x1000
#define parent(x) (((x + 1) >> 1) - 1)
#define lson(x) (((x + 1) << 1) - 1)
#define rson(x) ((x + 1) << 1)

struct buddy buddy;
unsigned ins_bitmap[2 * P_PAGE - 1];
unsigned long page_offset = 0x80000000ul;

extern unsigned long *_end;
```

Buddy-System

alloc_pages

```
void *alloc_pages(int num)
{
    int need = ROUNDUP(num);
    if (buddy_item.bitmap[0] < need)
    {
        for (int i = 0; i < 100; i++)
        {
            puti(buddy_item.bitmap[i]);
            puts(" ");
        }
        puts("What fuck ?\n");
        return '\0';
    }
    int index = 0;
    while (1)
    {
        if (index + 1 >= buddy_item.size)
        {
            break;
        }
        if (buddy_item.bitmap[index * 2 + 1] >= need)
        {
            index = index * 2 + 1;
        }
        else if (buddy_item.bitmap[index * 2 + 2] >= need)
        {
            index = index * 2 + 2;
        }
        else
        {
            break;
        }
    }
    uint64 *va = (uint64 *)(VM_START + PGSIZE * (buddy_item.bitmap[index] * (index + 1) - buddy_item.size));
    buddy_item.bitmap[index] = 0;
    while (index)
    {
        index = (index - 1) / 2;
        buddy_item.bitmap[index] = max(buddy_item.bitmap[index * 2 + 1], buddy_item.bitmap[index * 2 + 2]);
    }

    return va;
}
```

Buddy-System

```c
void *alloc_pages(int num)
{
    int need = ROUNDUP(num);
    if (buddy_item.bitmap[0] < need)
    {
        for (int i = 0; i < 100; i++)
        {
            puti(buddy_item.bitmap[i]);
            puts(" ");
        }
        puts("What fuck ?\n");
        return '\0';
    }
    int index = 0;
    while (1)
    {
        if (index + 1 >= buddy_item.size)
        {
            break;
        }
        if (buddy_item.bitmap[index * 2 + 1] >= need)
        {
            index = index * 2 + 1;
        }
        else if (buddy_item.bitmap[index * 2 + 2] >= need)
        {
            index = index * 2 + 2;
        }
        else
        {
            break;
        }
    }
    uint64 *va = (uint64 *)(VM_START + PGSIZE * (buddy_item.bitmap[index] * (index + 1) - buddy_item.size));
    buddy_item.bitmap[index] = 0;
    while (index)
    {
        index = (index - 1) / 2;
        buddy_item.bitmap[index] = max(buddy_item.bitmap[index * 2 + 1], buddy_item.bitmap[index * 2 + 2]);
    }

    return va;
}
```

Buddy-System

free_pages

```
void free_pages(void *ptr)
{
    int index = ((uint64)ptr - VM_START) / PGSIZE;
    int tmp = index;
    int nowIndex = index + buddy_item.size - 1;
    int nowSize = 1;

    while (1)
    {
        if (!buddy_item.bitmap[nowIndex])
        {
            buddy_item.bitmap[nowIndex] = nowSize;
            break;
        }
        nowIndex = (nowIndex - 1) / 2;
        nowSize *= 2;
    }

    while (nowIndex)
    {
        nowIndex = (nowIndex - 1) / 2;
        if (buddy_item.bitmap[nowIndex * 2 + 1] == nowSize && buddy_item.bitmap[nowIndex * 2 + 2] == nowSize)
        {
            buddy_item.bitmap[nowIndex] = 2 * nowSize;
        }
        else
        {
            buddy_item.bitmap[nowIndex] = max(buddy_item.bitmap[nowIndex * 2 + 1], buddy_item.bitmap[nowIndex * 2 + 2]);
        }
        nowSize *= 2;
    }
    return;
}
```

Buddy-System

```c
void free_pages(void *ptr)
{
    int index = ((uint64)ptr - VM_START) / PGSIZE;
    int tmp = index;
    int nowIndex = index + buddy_item.size - 1;
    int nowSize = 1;

    while (1)
    {
        if (!buddy_item.bitmap[nowIndex])
        {
            buddy_item.bitmap[nowIndex] = nowSize;
            break;
        }
        nowIndex = (nowIndex - 1) / 2;
        nowSize *= 2;
    }

    while (nowIndex)
    {
        nowIndex = (nowIndex - 1) / 2;
        if (buddy_item.bitmap[nowIndex * 2 + 1] == nowSize && buddy_item.bitmap[nowIndex * 2 + 2] == nowSize)
        {
            buddy_item.bitmap[nowIndex] = 2 * nowSize;
        }
        else
        {
            buddy_item.bitmap[nowIndex] = max(buddy_item.bitmap[nowIndex * 2 + 1], buddy_item.bitmap[nowIndex * 2 + 2]);
        }
        nowSize *= 2;
    }
    return;
}
```

Buddy-System

void *kmalloc （size_t size)

```c
    // size 若在 kmem_cache_objsize 所提供的范围之内，则使用 slub allocator 来分配内存
    for (objindex = 0; objindex < NR_PARTIAL; objindex++)
    {
        // YOUR CODE HERE
        if (size <= kmem_cache_objsize[objindex])
        {
            p = kmem_cache_alloc(slub_allocator[objindex]);
            return p;
        }
    }

    // size 若不在 kmem_cache_objsize 范围之内，则使用 buddy system 来分配内存
    if (objindex >= NR_PARTIAL)
    {
        // YOUR CODE HERE
        p = alloc_pages((size - 1) / PAGE_SIZE + 1);
        set_page_attr(p, (size - 1) / PAGE_SIZE + 1, PAGE_BUDDY);
    }

    return p;
}
```

Buddy-System

void kfree(void *addr)

```c
void kfree(const void *addr)
{
    struct page *page;

    if (addr == NULL)
        return;

    // 获得地址所在页的属性
    // YOUR CODE HERE
    page = ADDR_TO_PAGE(addr);
    // 判断当前页面属性
    if (page->flags == PAGE_BUDDY)
    {
        // YOUR CODE HERE
        free_pages(page->header);
        clear_page_attr(ADDR_TO_PAGE(addr)->header);
    }
    else if (page->flags == PAGE_SLUB)
    {
        // YOUR CODE HERE
        kmem_cache_free( obj: addr);
    }

    return;
}
```

Buddy-System

get_unmapped_area

```
unsigned long get_unmapped_area(size_t length)
{
    unsigned long begin = 0;
    struct vm_area_struct *tmp = current->mm.vm_area;
    while (1)
    {
        int conflict = 0;
        while (tmp != NULL)
        {
            if (begin < tmp->vm_end && tmp->vm_start < begin + length)
            {
                conflict = 1;
                begin = tmp->vm_end;
                break;
            }
            if (tmp->vm_start >= begin + length)
            {
                break;
            }
            tmp = tmp->vm_next;
        }
        if (!conflict)
        {
            break;
        }
    }

    return begin;
}
```

Buddy-System

Do_mmap

```
if (begin->vm_start >= t->vm_end)
{
    begin->vm_prev = t;
    t->vm_next = begin;
    mm->vm_area = t;
    return (void *)t->vm_start;
}
int confilct = 0;
while (begin != NULL)
{
    if (!confilct)
    {
        if (t->vm_start < begin->vm_end && begin->vm_start < t->vm_end)
        {
            confilct = 1;
        }
    }
    prev = begin;
    begin = begin->vm_next;
}
```

Buddy-System

Do_mmap

```
    if (confilct)
    {
        t->vm_start = get_unmapped_area(length);
        t->vm_end = t->vm_start + length;
    }
    //寻找插入的位置
    begin = mm->vm_area;
    while (begin != NULL)
    {
        if (begin->vm_start >= t->vm_end)
        {
            prev->vm_next = t;
            t->vm_next = begin;
            t->vm_prev = prev;
            begin->vm_prev = t;
            return (void *)t->vm_start;
        }
        prev = begin;
        begin = begin->vm_next;
    }
    prev->vm_next = t;
    t->vm_prev = prev;
    printk("[S] New vm_area_struct: start %lx, end %lx, prot [r:%d,w:%d,x:%d]\n",
           t->vm_start, t->vm_end, t->vm_flags & VM_READ, t->vm_flags & VM_WRITE, t->vm_flags & VM_EXEC);
    return (void *)t->vm_start;
}
```

Buddy-System

mprotect

```c
void mprotect_do(uint64 pagetable, uint64 va, size_t __len, int prot)
{
    uint64 stop = va + __len;
    struct PAGE *now = (struct PAGE *)pagetable;
    while (va < stop)
    {
        int index2 = VPN2(va), index1 = VPN1(va), index0 = VPN0(va);
        uint64 pte2, pte1, pte0;

        pte2 = now->entrys[index2];
        if (!IsValid(pte2))
        {
            va += PGSIZE;
            continue;
        }

        now = (struct PAGE *)((pte2 >> 10) << 12);

        pte1 = now->entrys[index1];
        if (!IsValid(pte1))
        {
            va += PGSIZE;
            continue;
        }

        now = (struct PAGE *)((pte1 >> 10) << 12);
        pte0 = now->entrys[index0];
        if (IsValid(pte0))
        {
            now->entrys[index0] = ((pte0 >> 10) << 10) | (1 + (prot << 1));
        }
        va += PGSIZE;
    }
    return;
}
```

Buddy-System

Some conclusion

- -fno-builtin
- -gdwarf-2
- -O2

Buddy-System

Some conclusion

- -fno-builtin
- -gdwarf-2
- -O2

Table of Contents

1. Background
2. Buddy-system
3. System Call Table
4. File Loader
5. Shell
6. Conclusions and Thoughts

System Call Table

sys-call Implemented in finished course

- sys-call 64: SYS_WRITE (Lab4)
  - sys_write(unsigned int fd, const char* buf, size_t count)
  - printout the string from user, is call by user-function printf

System Call Table

sys-call Implemented in finished course

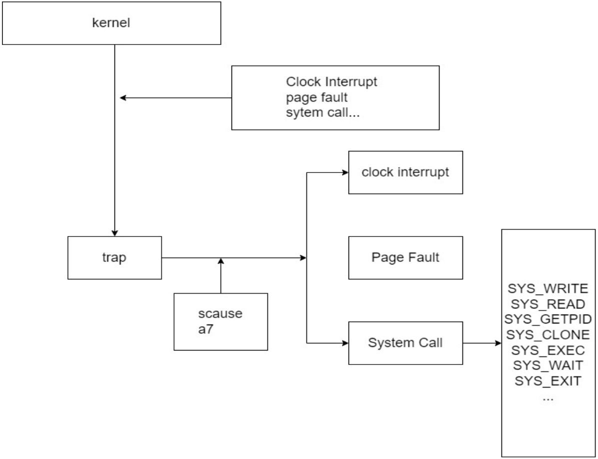
- sys-call 174: SYS_GETPID (Lab4)
    - sys_getpid
    - pick the pid of current process, pass it to user program with register a0

System Call Table

sys-call Implemented in finished course

- sys-call 220: SYS_CLONE (Lab5)
  - sys_clone(struct pt_regs *regs)
  - execute the fork process of current running process

System Call Table



System Call Table

sys-call Implemented in finished course

- The former implemented three sys-calls build the framework of sys-call schema of a kernel, we could extend the sys-call of the kernel with a unified schema

System Call Table

```
void trap_handler(unsigned long scause, unsigned long sepc, struct pt_regs *regs) {
    if(timer interrupt)
    {
        clock_set_next_event();
        do_timer();
    }
    else
    {
        if(Environment call from U-mode)
        {
            if(SYS_GETPID) {...}
            else if(SYS_WRITE) {...}
            else if(SYS_CLONE) {...}

            else if(SYS_READ) {...}
            else if(SYS_WAIT) {...}
            else if(SYS_EXIT) {...}
        }
        /*else if(Environment call from S-mode)
        {

        }*/
        else if(page fault handling)
        {
            do_page_fault(regs);
        }
    }
}
```

System Call Table

sys-call Implemented in X-Part

sys-call in buddy system

System Call Table

sys-call Implemented in X-Part

sys-call in buddy system

- sys_mprotect
- sys_munmap

System Call Table

sys-call Implemented in X-Part : In Buddy System (Referencing lab6)

- sys-call 226 : SYS_MPROTECT
  - change the memory access with specific content
  - int mprotect(void __addr, size_t _len, int __prot)

```
int mprotect(void *__addr, size_t __len, int __prot)
{
    // imnput prot
    mprotect_do(current->mm.pgtbl, (uint64)__addr, __len, __prot);
    return 0;
}
```

System Call Table

sys-call Implemented in X-Part : In Buddy System (Referencing lab6)

- sys-call 215 : SYS_MUNMAP
  - cancel virtual memory mapping in specific memory area, if success return 0, else return -1
  - int munmap (void *__addr, size_t __len)

```c
int munmap(void *start, size_t length)
{
    struct vm_area_struct *t = current->mm.vm_area;
    struct vm_area_struct *prev = NULL;
    while (t != NULL)
    {
        if (find memory area with input auguments)
        {
            free memory mapping;
            kfree(t);
            return 0;
        }
        continue searching in vma;
    }
    return -1;
}
```

System Call Table

sys-call Implemented in X-Part : In File system implementation and program loading (Referencing lab7)

System Call Table

sys-call Implemented in X-Part : In File system implementation and program loading (Referencing lab7)

- sys-call 63: SYS_READ
  - sys_read(struct pt_regs *regs)
  - read the string input from user program(shell), store it in buffer

```
int sys_read(struct pt_regs *regs){
    char* buf = regs->a1;
    int nbuf = regs->a2;

    for(i from 0 to nbuf){
        scaning input to buf;
    }
    return length of reading string;
}
```

System Call Table

sys-call Implemented in X-Part : In File system implementation and program loading (Referencing lab7)

- sys-call 260: SYS_WAIT
  - We implement two states of process in OS kernel : TASK_RUNNING and TASK_WAITING, serving for process switch for user programs
  - int sys_wait(struct pt_regs *regs)
  - set the current process(in running state) to pending state, the perform schedule(), switching to other program

```
int sys_wait(struct pt_regs *regs) {
    set current task as pending;
    schedule to switch to another task;
}
```

System Call Table

sys-call Implemented in X-Part : In File system implementation and program loading (Referencing lab7)

- sys-call 93: SYS_EXIT
  - void sys_exit()
  - Exit current children process, return to parent process
  - The difference from SYS_WAIT

```
void sys_exit() {
    while(1) {
        if( parent is pending )
        break;
        else
        schedule to give up cpu;
    }
    set parent as running state;
    schedule to give up cpu;
}
```

System Call Table

```
int proc_exec(struct task_struct *proc, const char *path) {
    memory preparing;

    for(for each segment in elf) {
        readi() read the program header(denoted as prog_header) of each segment;
        check whether prog_header.type == LOAD;
        parse_ph_flags() parse the prog_header.flags to permissions;
        uvmalloc() allocate user pages, and set proper permissions;
        loadseg() copy the content of this segment to allocated memory space;
    }

    allocate a page for user stack and update the newpgtbl;

    set proc's sstatus, sscratch, and sepc like task_init() in Lab5;
    set proc->pgtbl to newpgtbl;
}
```

System Call Table

Summary

- During these OS labs, we constructed a basic framework of systems call mechanism of a OS kernel.

System Call Table

Summary

- And implemented system call table with basic kernel function, which supports a toy-OS running
    - information input and output (SYS_READ, SYS_WRITE, SYS_GETPID)
    - basic memory management (SYS_MMAP, SYS_MPROTECT)
    - process fork (SYS_CLONE)
    - user program process management (SYS_WAIT, SYS_EXIT, SYS_EXECVE)

Table of Contents

1 Background

2 Buddy-system

3 System Call Table

4 File Loader

5 Shell

6 Conclusions and Thoughts

File Loader

File system integrating into kernel.

- We use a temporary file system: initramfs

File Loader

File system integrating into kernel.

- We use a temporary file system: initramfs
- The OS kernel will create a tmplfs file system to access the content

File Loader

We use a set of function serve as the file system interface of the Toy-OS: cpio, there are two major function in it

- struct inode *namei(char* path): receive the path of the file, return the loaction(node) in inittramfs
- int readi(struct inode *ip, int user_dst, void *dst, uint off, uint n);

```c
struct inode *namei(char *path) {//path文件名
    struct inode *n = kalloc();
    struct cpio_stat *stat = kalloc();
    n->i_private = stat;
    *stat = cpio_find_file(path);
    return n;
}

int readi(struct inode *ip, int user_dst, void *dst, uint64 off, uint64 n) {
    struct cpio_stat *stat = ip->i_private;
    void *base = stat->data;
    memmove(dst, base + off, n);
    return n;
}
```

File Loader

- In this section, we managed to load cpio into kernel with mapping it into memory, then the interatction between the Toy-OS and file system could be established
- After a large amount of trial and analysis, we adjusted part of the file construction, make cpio integrated in our Toy-OS

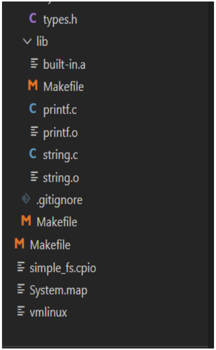
```
run: all
	@echo Launch the qemu ......
	@qemu-system-riscv64 -nographic -machine virt -kernel vmlinux -bios default -initrd simple_fs.cpio

debug: all
	@echo Launch the qemu for debug ......
	@qemu-system-riscv64 -nographic -machine virt -kernel vmlinux -bios default -initrd simple_fs.cpio -S -s
```

```
    types.h
  lib
    built-in.a
    Makefile
    printf.c
    printf.o
    string.c
    string.o
  .gitignore
  Makefile
Makefile
simple_fs.cpio
System.map
vmlinux
```

File Loader

- In this section, we managed to load cpio into kernel with mapping it into memory, then the interaction between the Toy-OS and file system could be established
- After a large amount of trial and analysis, we adjusted part of the file construction, make cpio integrated in our Toy-OS




```
run: all
	@echo Launch the qemu ......
	@qemu-system-riscv64 -nographic -machine virt -kernel vmlinux -bios default -initrd simple_fs.cpio

debug: all
	@echo Launch the qemu for debug ......
	@qemu-system-riscv64 -nographic -machine virt -kernel vmlinux -bios default -initrd simple_fs.cpio -S -s
```

File Loader

- Elf Loader: make program running on our kernel
- Overview of ELF header
  - main segment:
    - .text
    - .rodata
    - .data and .bss
  - elf header and program header store basic infomation about elf file
    - including the offset of program header table and section header table

File Loader

- Elf Loader

```
                  +--------------------+
                  |                    |
                  |     ELF Header     |
                  |                    |
                  +--------------------+
                  |                    |
        +---------+Program Header Table|
        |         |                    |
        |      +-----------------------+
        |      |  |                    |
        |      |  |       .text        |   <-----------+
        |      |  |                    |               |
        +--->  |  +--------------------+               |
        |      |  |                    |               |
        |      |  |      .rodata       |   <-----------+
        |      |  |                    |               |
        |      +-----------------------+               |
        |         |                    |               |
        |      +-----------------------+               |
        |      |  |                    |               |
        +--->  |  |       .data        |   <-----------+
               |  |                    |               |
               +-----------------------+               |
                  |                    |               |
                  |        .bss        |   <-----------+
                  |                    |               |
                  +--------------------+               |
                  |                    |               |
                  |Section Header Table+---------------+
                  |                    |
                  +--------------------+
```

File Loader

The Implemation: function loadseg() and function system call exec, for the exec() is described before, this time we focus on loadseg().

- int loadseg(page_table_t pagetable, uint64 va, struct inode *ip, uint offset, uint filesz);

- the function will walk through the pagetable, get the physical address of corresponding virtual address from given arguments
- when fetch the physical address, it will use readi() function, casting the content of the segment to memory physical address
- finally, turn to exec to execute the loaded program

File Loader

The Implemation: function loadseg() and function system call exec, for the exec() is described before, this time we focus on loadseg().

```c
    int ofs;
    for(i = 0, ofs = elf->phoff; i < elf->phnum; i++, ofs += sizeof(struct proghdr)) {
        if(readi(ip, 0, prog, ofs, sizeof(struct proghdr)) != sizeof(struct proghdr)) {
            printk("load program error!\n");
            return -1;
        }
        if(prog->type != ELF_PROG_LOAD
        || prog->memsz < prog->filesz
        || prog->vaddr + prog->memsz < prog->vaddr) {
            //printk("parse ph flags failed!\n");
            //return -1;
            continue;
        }
        void *segment=kalloc();
        create_mapping(pgd, prog->vaddr, segment-PA2VA_OFFSET,prog->filesz , (uint64)(prog->flags +8)); // PTE_U 和 PTE_X
        if(prog->vaddr % PGSIZE != 0) {
            printk("prog vadder is not aligned!\n");
            return -1;
        }
        //readi(ip,0,segment,prog->off,prog->filesz);
        if(loadseg(pgd, prog->vaddr, ip, prog->off, prog->filesz) == -1) {
            printk("loadseg failed!\n");
            return -1;
        }
    }
```

Table of Contents

1. Background
2. Buddy-system
3. System Call Table
4. File Loader
5. Shell
6. Conclusions and Thoughts

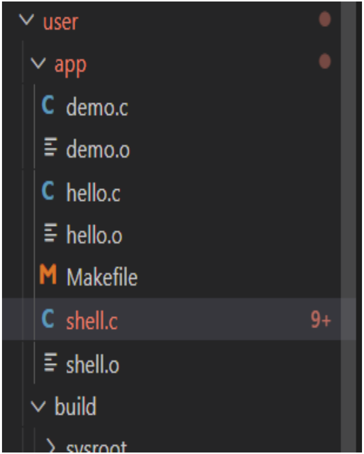
Shell

In lab7, the shell program is not provided, so wo implement a simple shell program to make user interact with the toy-os, to show the implementation of this program, we prepared two user programs: hello

Shell

In lab7, the shell program is not provided, so wo implement a simple shell program to make user interact with the toy-os.

To show the implementation of this program, we prepared a user programs: hello



Shell

Shell program introduction

- mainly a simple endless loop, waiting for user input

Shell

Shell program introduction

- mainly a simple endless loop, waiting for user input
- when receiving the user input, executing specific program
  - the kernel will fork a children process for the application to execute
  - the runcmd function will:
- load the specific program from file system to memory (unsing namei() and loadseg())
- then execute the program

Shell

Shell program introduction

```c
int main(void) {
	char buf[100];
	int fd;

	while(getcmd(buf,sizeof(buf)) >= 0) {
	  //printf("%s\n", buf);
		if (strcmp(buf, "hello") == 0
			|| strcmp(buf, "demo") == 0) {
			int pid = fork();
			printf("Get pid %d\n", pid);
			if(pid == 0)
				runcmd(buf);
			else if (pid > 0) {
				wait();
			}
			else
				printf("panic fork");
		}
		else {
			printf("Invalid command %s\n", buf);
		}
	}
	exit(0);
}

void runcmd(char* cmd) {
	exec(cmd);
	printf("exec %s failed\n", cmd);
	exit(0);
}
```

Shell

Shell program implementation

- The shell program is simple, but the procedure to load it is more complicated
  - Need to call function elf_loader to load shell program into memory
  - The function is called in task_init, after the idle process is inited

Shell

Shell program implementation

```c
void task_init(){
    /*
    allocate task struct
    elf_loader(task, "shell") // "shell" is the path of shell
    set task status to READY
    */
    // 1. 调用 kalloc() 为 idle 分配一个物理页
    idle = (struct task_struct*)kalloc();
    // 2. 设置 state 为 TASK_RUNNING;
    idle -> state = TASK_RUNNING;
    // 3. 由于 idle 不参与调度 可以将其 counter / priority 设置为 0
    idle -> counter = 0;
    idle -> priority = 0;
    // 4. 设置 idle 的 pid 为 0
    idle -> pid = 0;
    // 5. 将 current 和 taks[0] 指向 idle
    current = idle;
    task[0] = idle;
    int i,j;

    task[1]=(struct task_struct*)kalloc();

    task[1]->state = TASK_RUNNING;
    task[1]->counter=1;
    task[1]->priority=rand();
    task[1]->pid=1;
    task[1]->thread.ra=&__dummy;
    task[1]->thread.sp=(uint64)(task[1])+PGSIZE;
    task[1] -> thread_info=(struct thread_info *)kalloc();
    task[1] -> thread_info->kernel_sp = task[1] -> thread.sp - PA2VA_OFFSET;//PA

    elf_loader(task[1],"shell");

    printk("...proc_init done!\n");
}
```

Shell

Shell program execution

```
Build Finished OK
Launch the qemu ......

OpenSBI v0.9
   ____                    _____ ____ _____
  / __ \                  / ____|  _ \_   _|
 | |  | |_ __   ___ _ __ | (___ | |_) || |
 | |  | | '_ \ / _ \ '_ \ \___ \|  _ < | |
 | |__| | |_) |  __/ | | |____) | |_) || |_
  \____/| .__/ \___|_| |_|_____/|____/_____|
        | |
        |_|

Platform Name             : riscv-virtio,qemu
Platform Features         : timer,mfdeleg
Platform HART Count       : 1
...mm_init done!
Current is: .
Current is: demo
Current is: flag
Current is: hello
Current is: shell
...proc_init done!
[S-MODE] Hello RISC-V
[!] Switch from task 0[ffffffe007fbf000] to task 1[ffffffe007fbe000], prio: 1, counter: 1
$ hello
Get pid 2
[!] Switch from task 1[ffffffe007fbe000] to task 2[ffffffe007fb0000], prio: 10, counter: 10
Get pid 0
Current is: .
Current is: demo
Current is: flag
Current is: hello
[User] pid: 2, sp is 0000003fffffffe0
[User] pid: 2, sp is 0000003fffffffe0
```

Table of Contents

1 Background

2 Buddy-system

3 System Call Table

4 File Loader

5 Shell

6 Conclusions and Thoughts

Conclusions and Thoughts

Thanks!