Jos Lab 4: Preemptive Multitasking

Li Xinyu 515030910292

Part A: Multiprocessor Support and Cooperative Multitasking

Application Processor Bootstrap

> **Exercise 1**
> 修改 `page_init()` 的代码，将起始地为 `MPENTRY_PADDR` 的页排除在 `page_free_list` 之外，只要判断语句改为 `(i < npages_basemem && i != MPENTRY_PADDR/PGSIZE)` 即可。

Per-CPU State and Initialization

> **Exercise 2**
> 修改函数 `mem_init_mp()`，为每个CPU独自的栈空间做地址映射，每个栈的大小是 `KSTKSIZE`，但是每个栈中间还有大小为 `KSTKGAP` 的溢出保护区域。因此计算每个栈的栈底位置为 `uintptr_t kstackbtm_i = KSTACKTOP - i * (KSTKSIZE + KSTKGAP) - KSTKSIZE`，再调用 `boot_map_region(kern_pgdir, kstackbtm_i, KSTKSIZE, PADDR(percpu_kstacks[i]), PTE_W)` 完成映射。
> 但是由于原先在 `mem_init()` 中 `boot_map_region_large(kern_pgdir, KERNBASE, ksize, 0, PTE_W)` 使用大页映射了 `KERNBASE` 以上的地址区域，导致函数 `mem_init_mp()` 中 `boot_map_region(kern_pgdir, IOMEMBASE, -IOMEMBASE, IOMEM_PADDR, PTE_W)` 在 `page_walk` 的时候就会把 `pde` 当成 `pte` 导致出错，因此将 `boot_map_region_large` 改回成 `boot_map_region`。

-

> **Exercise 3**
> 修改函数 `trap_init_percpu()`，将 `ts` 改成 `thiscpu->cpu_ts`，将 `gdt[GD_TSS0 >> 3]` 改为 `gdt[(GD_TSS0 >> 3) + i]`，并将 `thiscpu->cpu_ts.ts_esp0` 改为Exercise2中设置的栈即 `KSTACKTOP - thiscpu->cpu_id * (KSTKSIZE + KSTKGAP)`。

Locking

>

Exercise 4
在函数 `i386_init()` 中 `boot_aps()` 之前调用 `lock_kernel()`；
在函数 `mp_main()` 的最后调用 `lock_kernel()` 然后调用 `sched_yield()`；
在函数 `trap()` 中 `assert(curenv)` 之前调用 `lock_kernel()`；
在函数 `env_run` 中 `env_pop_tf()` 之前调用 `unlock_kernel()`；

-

Exercise 4.1
实现 `spinlock.c` 中的ticket spinlock：
在函数 `holding()` 中使用 `return lock->cpu == thiscpu;` 判断当前CPU是否持有锁；
在函数 `__spin_initlock` 将 `lk->own` 和 `lk->next` 都初始化为0；
在函数 `spin_lock()` 中首先调用原子操作 `atomic_return_and_add` 将 `lk->next` 加一都到当前的票数，然后判断其是否与 `lk->own` 相等，否则等待 `asm volatile ("pause")`；
在读取 `lk->own` 的时候也要使用 `atomic_return_and_add` 这一原子操作(只不过加0)，否则就会卡住。
在函数 `spin_unlock()` 中释放锁，调用 `atomic_return_and_add` 将 `lk->own` 加一就好。
最后在 `spinlock.h` 中添加宏定义 `#USE_TICKET_SPIN_LOCK` 开启ticket lock。

Round-Robin Scheduling

Exercise 5
修改函数 `sched_yield()`，如果 `cur_env` 不为NULL，获得其在 `envs` 中的index，然后从index的后一个开始循环遍历 `envs` 直至回到index，看是否存在一个 `env_type != ENV_TYPE_IDLE` 且 `envs[i].env_status == ENV_RUNNABLE` 的environment，如果存在则调用 `env_run` 运行，否则如果 `curenv->env_status == ENV_RUNNING` 就运行当前的environment。
然后在 `kern/syscall.c` 中的switch条件中添加一条 `case SYS_sbrk: ret = sys_sbrk(a1);`。
另外在进syscall的时候要保存Trapframe，还要拿锁和放锁确保只有一个CPU处在内核态，因此添加了一个包装函数 `syscall_trap`。由于用户也可以由int 0x80进入内核，因此还需要注册 `T_SYSCALL` 的trap_handler并在 `trap_dispatch()` 中进行处理。另外还要修改 `trapentry.S` 中的相关代码。

System Calls for Environment Creation

Exercise 6
修改 `sys_exofork()`，调用 `env_alloc()` 创建新的enviroment，将状态设置为 `ENV_NOT_RUNNABLE`，然后复制当前enviroment的寄存器信息，然后返回其 `env_id`。
修改 `sys_env_set_status()`，调用 `envid2env()` 获取 `env_id` 对应的 `struct Env*`，出错则返回。如果其状态不是 `ENV_NOT_RUNNABLE` 或者 `ENV_RUNNABLE` 则返回 `-E_INVAL`，否则设置状态为传入的参数 `status`。
修改 `sys_alloc()`，首先检查 `va` 的地址是否合法，是否对齐以及相关权限是否正确，否

则返回 `-E_INVAL`。调用 `envid2env()` 获取 `env_id` 对应的 `struct Env*`，出错则返回，调用 `page_alloc()` 分配一个页，若返回的页为NULL则返回 `-E_NO_MEM`，最后调用 `page_insert()` 完成映射。
修改 `sys_page_map()`，首先检查地址是否合法、对齐情况以及权限是否正确，然后调用 `envid2env()` 获取 `srcenvid` 和 `dstenvid` 对应的环境 `srcenv` 和 `dstenv`，使用 `page_lookup()` 找到 `srcenv` 中 `srcva` 所对应的page，然后检查相关权限，最后调用 `page_insert()` 映射到 `dstenv` 中 `dstva` 地址处。
修改 `sys_page_unmap()`，首先检查地址是否合法以及对齐情况，然后调用 `envid2env()` 获取 `env_id` 对应的环境 `e`，然后调用 `page_remove` 取消 `e` 中 `va` 地址处的物理页映射。
最后为以上系统调用在 `syscall()` 的switch中添加各自的case语句。

Part B: Copy-on-Write Fork

Setting the Page Fault Handler

Exercise 7
修改函数 `sys_env_set_pgfault_upcall()`，调用 `envid2env()` 获取 `env_id` 对应的 `struct Env*`，出错则返回，然后对应environment 的 `env_pgfault_upcall` 设置为传入的函数指针 `func`。
然后在 `syscall()` 的switch中添加 `sys_env_set_pgfault_upcall` 的case语句。

Invoking the User Page Fault Handler

Exercise 8
修改 `page_fault_handler()`，如果 `curenv->env_pgfault_upcall != NULL`，根据 `tf_esp` 的地址设置UTrapframe如下：

```
if (tf->tf_esp >= UXSTACKTOP - PGSIZE && tf->tf_esp < UXSTACKTOP)
  utf = (struct UTrapframe *)(tf->tf_esp - utfsize - 4);
else
  utf = (struct UTrapframe *)(UXSTACKTOP - utfsize);
```

然后根据传入的 `tf` 设置UTrapframe的相关参数，最后将 `tf->tf_eip` 设置为 `curenv->env_pgfault_upcall`，`tf->tf_esp` 设置为 `utf`，
然后调用 `env_run(curenv)` 进入 `env_pgfault_upcall` 处理page fault。

User-mode Page Fault Entrypoint

Exercise 9
将trap-time的%eip保存到trap-time的栈上，根据UTrapframe的结构，`utf_eip` 在结构体第40字节处即 `0x28(%esp)`，相应地 `utf_esp` 即为 `0x30(%esp)`。

如下：

```
movl 0x30(%esp), %eax
movl 0x28(%esp), %ecx
subl $0x4, %eax
movl %ecx, (%eax)
movl %eax, 0x30(%esp)
```

然后将恢复trap-time寄存器，`struct PushRegs utf_regs`在结构体第8字节处，如下：

```
addl $0x8, %esp
popal
```

然后从栈上恢复elfags如下：

```
addl $0x4, %esp
popfl
```

最后调用`popl %esp`和`ret`回到page fault的指令处重新执行。

-

Exercise 10
修改`set_pgfault_handler()`，如果`_pgfault_handler == 0`,则注册一个page fault handler，首先调用`sys_page_alloc()`分配一个exception stack，如果返回值小于0则panic，然后调用`sys_env_set_pgfault_upcall`设置pgfault_upcall，同样如果返回值小于0则panic。

Implementing Copy-on-Write Fork

Exercise 11
修改`pgfault()`，首先判断faulting access是不是一个COW页以及是不是写操作，否则panic。然后调用`sys_page_alloc`分配一个页映射到暂时的地址`PFTEMP`，然后把地址`va`所在的页的内容拷贝到这个新页中，最后调用`sys_page_map`把新页映射回原地址`va`。
修改`duppage()`，首先判断page `pn`的地址`addr`是否小于UTOP以及是否在内存中(PTE_P)，否则直接返回。如果这个页是一个可写的或是COW，调用`sys_page_map`将这个页映射到目标environment的`addr`处设置权限为`PTE_U|PTE_P|PTE_COW`，并将这页重新映射修改其权限为`PTE_U|PTE_P|PTE_COW`；否则只调用`sys_page_map`一次将这个页映射到目标environment的`addr`处设置权限为`PTE_U|PTE_P`。
修改`fork()`，首先调用`set_pgfault_handler()`设置page fault handler，然后设置`sys_exofork()` fork出一个子环境，若返回值小于0则panic；若等于0(说明此时运行的是子环境)，则设置`thisenv`为`envs[ENVX(sys_getenvid())]`并返回0。否则由父环境对子环境的地址空间进行设置：遍历[0, UTOP/PGSIZE)的区域，如果不等于`(UXSTACKTOP-PGSIZE) / PGSIZE`且页位于内存中且为用户可见，则调用`duppage`进行映射；然后调用`sys_page_alloc()`为子环境设置exception stack，调用`sys_env_set_pgfault_upcall()`设置page fault handler，调用`sys_env_set_status`设置状态为`child_envid,ENV_RUNNABLE`,最后返回child_envid。

Part C: Preemptive Multitasking and Inter-Process communication (IPC)

Interrupt discipline

> **Exercise 12**
> 在 `trapentry.S` 中添加为每个irq_i添加一个 `TRAPHANDLER_NOEC(th_irq_i, IRQ_OFFSET+i)`，
> 然后在 `trap_init` 中，使用global的vector table调用 `SETGATE(idt[IRQ_OFFSET+i], 0, GD_KT, vectors[i+21], 0);` 进行注册。
> 在 `env_alloc()` 中添加一行 `e->env_tf.tf_eflags |= FL_IF` 开启用户态的中断。

Handling Clock Interrupts

> **Exercise 13**
> 在 `trap_dispatch()` 的switch中添加一条 `case IRQ_OFFSET+IRQ_TIMER:`，调用 `lapic_eoi(); sched_yield();` 进行处理。

Inter-Process communication (IPC)

> **Exercise 14**
> 修改 `sys_ipc_recv()`，如果 `dstva` 地址低于UTOP但是不是页对齐，则返回 `-E_INVAL`，否则设置 `env_ipc_recving` 为1， `env_ipc_dstva` 为 `dstva`， `env_status` 为 `ENV_NOT_RUNNABLE`，然后调用 `sched_yeild()`。
> 修改 `sys_ipc_try_send()`，首先使用 `envid2env` 获取 `envid` 对应的 `struct Env *` 结构体指针 `dstenv`，如果 `dstenv` 的 `env_ipc_recving` 不为1，说明目标环境没有被阻塞，返回 `-E_IPC_NOT_RECV`。如果 `srcva` 低于UTOP，检查权限以及页对齐情况，然后调用 `page_lookup` 得到 `srcva` 所映射的页 `pg` 并检查相关权限，然后如果 `dstenv->env_ipc_dstva` 也低于UTOP，则使用 `page_insert` 将 `pg` 映射到 `dstenv->env_ipc_dstva`。最后设置 `dstenv` 中与ipc相关的参数以及状态等信息。
> 最后在系统调用在 `syscall()` 的switch中添加 `case SYS_ipc_recv:` 和 `case SYS_ipc_try_send:`。
> 修改 `ipc_recv()`，如果 `pg` 为NULL，将其设置为UTOP(即意为no page)。然后调用 `sys_ipc_recv(pg)`，如果返回值小于0(说明失败)，则 `from_env_store` 和 `perm_store` 设置为0(如果不为NULL)；否则将 `from_env_store` 设置为 `thisenv->env_ipc_from`，`perm_store` 设置为 `thisenv->env_ipc_perm` 并返回 `thisenv->env_ipc_value`。
> 修改 `ipc_send()`，如果 `pg` 为NULL，将其设置为UTOP(即意为no page)。然后调用 `sys_ipc_try_send(to_env,val,pg,perm)` 直至返回值为0(发送成功)。如果返回值不为 `-E_IPC_NOT_RECV` 则panic；否则重试，重试之前调用 `sys_yield()` 交出CPU使用权(CPU-friendly)。

Challenge

实现 `sbrk()`：
在 `inc/lib.h` 中定义宏 `#define CHALLENGE`
修改 `fork.c` 中的函数 `sbrk()`，与 `brk()` 类似，只不过在进行页映射时，调用 `sys_page_map()` 而非 `duppage()` 实现内存的共享，但是要避免栈空间的共享，将索引为 `USTACKTOP/PGSIZE-1` 的页通过 `duppage()` 映射为COW，其余代码与 `brk()` 一致。
另外，由于 `thisenv` 是在 `libmain.c` 中申明的全局变量，而 `sbrk()` 使得父环境和子环境共享堆空间造成 `thisenv` 在子环境中也指向父环境的 `struct Env` 而造成IEC失效，因此需要将其显示地设置为 `&envs[ENVX(sys_getenvid())]`。
具体地，在 `ipc_recv()` 中添加一行

```
#ifdef CHALLENGE
  thisenv = &envs[ENVX(sys_getenvid())];
#endif
```

修改 `pingpongs.c` 中的函数 `umain()`，在 `sfork()` 之后如果回到父环境，则首先使得 `thisenv = &envs[ENVX(sys_getenvid())]`，此外在发送IEC的循环中的也首先加上 `thisenv = &envs[ENVX(sys_getenvid())]` 一句使得 `thisenv` 指向正确的 `struct Env`。
另外修改 `forktree.c` 中的函数 `forkchild()` 如下：

```
#ifdef CHALLENGE
  if (sfork() == 0) {
#else
  if (fork() == 0) {
#endif
```