Challenge 完成的是 PartB 中的 sfork

首先回顾一下 PartB 的内容

PART B Copy-on-Write Fork

这个 Part 的核心就是完成一个 copy-on-write 的 fork，即写时拷贝的 fork

起因是因为 fork 出的子进程绝大多数都会直接 excv 别的进程，和父进程执行相同代码的子进程毕竟是少数

如果是完整的 fork 把父进程的资源全部复制一遍的话，那么会造成大量的资源浪费（资源复制时间的浪费，复制出来的资源没有被使用），造成 kerenl 的低效率

鉴于这个原因，需要实现一个 copy-on-write 的 fork（写时拷贝）

copy-on-write 的 fork 的核心思想是 child 除了 UXSTACK 之外，child 只把 parent 的内存映射到自己的内存上，并设为 COW（即可读但是写时拷贝），但是不设置自己的内存。只有当试图写这些内存中的某个 page 时，才会触发 page_fault 并调用注册的 page_fault_handler 去重新拷贝一份 page

整个执行过程：

1.用户调用 fork

2. 利 用 set_pagefault_handler 设 置 pfentry.S 中 的 _pagefault_handler 入 口 为 pgfault ， 并 用 sys_env_set_pgfault_upcall 设 置 env_pagefault_upcall 为 _pagefault_upcall（ _pagefault_upcall 中调用_pagefault_handler）

3. fork 中用之前的 sys_exofork 创建子进程

4.用 duppage 映射内存，设置子进程的 env_pagefualt_upcall（同样为_pagefault_upcall）和 UXSTACK

5.把子进程的状态设为 ENV_RUNNABLE 并返回

6.在发生 SYS_PGFLT 时，被 trap 截住进入 trapdispatch 再进入 page_fault_handler，设置处理的 UXSTACK，把 eip 指向之前设置的 env_pgfault_upcall（即_pgfault_upcall），_pgfault_upcall 再调用 pgfalut 处理之后返回

sfork 与 fork 的区别：

fork 是子进程与父进程共享内存是可读而不可写的 ,当试图 write 时会触发 pgfault 重新拷贝一份 page

而 sfork 中子进程与父进程除了 ustack 与 uxstack 外 ,共享的内存是即可读又可写的，也就是完全共用一块内存。

在使用 sfork 时需要注意的事情：

既然 sfork 是除了 stack 完全共用一块内存，那么就是说，只有 stack 上的局部变量是不共享的，全局变量是完全共享的，我们需要调整所有用到 thisenv（在 lib/libmain.c 中声明的全局变量）为 envs+ENVX(sys_getenvid())（当然实际上只有 ipc_recv 用到）

Challenge! Implement a shared-memory `fork()` called `sfork()`. This version should have the parent and child *share* all their memory pages (so writes in one environment appear in the other) except for pages in the stack area, which should be treated in the usual copy-on-write manner. Modify `user/forktree.c` to use `sfork()` instead of regular`fork()`. Also, once you have finished implementing IPC in part C, use your `sfork()` to run `user/pingpongs`. You will have to find a new way to provide the functionality of the global `thisenv` pointer.

```
static int sduppage(envid_t envid, unsigned pn, int is_stack )
{
    int r = 0;
    void* temp = (void*) (pn*PGSIZE);

     if ( is_stack || (vpt[pn] & PTE_COW) )    //SAME AS duppage, use copy on right
    {
        if ((r = sys_page_map(sys_getenvid(), temp, envid, temp, PTE_P | PTE_U | PTE_COW )) < 0) return r;
        if ((r = sys_page_map(sys_getenvid(), temp, sys_getenvid(), temp, PTE_P | PTE_U | PTE_COW )) < 0) return r;
    }
```

```
	else if ( (vpt[pn] & PTE_W) )    //different
	{
		if ((r = sys_page_map(sys_getenvid(), temp, envid, temp, PTE_P |
PTE_U | PTE_W)) < 0) return r;
		if ((r = sys_page_map(sys_getenvid(), temp, sys_getenvid(), temp,
PTE_P | PTE_U | PTE_W )) < 0) return r;
	}
	else    //SAME AS duppage
	{
		if ((r = sys_page_map(sys_getenvid(), temp, envid, temp, PTE_U |
PTE_P)) < 0) return r;
	}
	return 0;
}

int sfork(void)
{
	extern void _pgfault_upcall (void);
	envid_t new_id;
	uintptr_t addr;
	int r;
	extern unsigned char end[];
```

```
    //1. init pgfault
    set_pgfault_handler(pgfault);

    //2. create a child environment.
    new_id = sys_exofork();
    if (new_id < 0) panic("sys_exofork: %e", new_id);
    if (new_id == 0)
    {
        // can not be changed for thisenv is a globle variable
        // in sfork, we should not use it anymore
        //thisenv = &envs[ENVX(sys_getenvid())];
        return 0;
    }

// 3. map
    int is_stack = 1;
    for (addr = UXSTACKTOP - PGSIZE - PGSIZE - PGSIZE; addr >= UTEXT
- PGSIZE; addr -= PGSIZE )
        if ( (vpd[PDX(addr)] & PTE_P)>0 && (vpt[PGNUM(addr)] &
PTE_P) >0 && (vpt[PGNUM(addr)] & PTE_U) >0  )
        {
```

```
            if ((r = sduppage(new_id, PGNUM(addr),is_stack)) < 0)
                return r;
        }
        else is_stack = 0;

    // allocate exception stack
    if ((r = sys_page_alloc(new_id, (void *) (UXSTACKTOP - PGSIZE), PTE_P
| PTE_U | PTE_W)) < 0) return r;

    //4. set a entrypoint for child
    if ((r = sys_env_set_pgfault_upcall(new_id, _pgfault_upcall)) < 0)
return r;

    //5. mark child to runnable
    if ((r = sys_env_set_status(new_id, ENV_RUNNABLE)) < 0) return r;

    return new_id;
}
```

代码实现：

和 fork 类似，但是区别在于 stack 处是 PTE_COW 的，其他如果本身是 PTE_W 的话，也设为 PTE_W，所以设置一个 sduppage 替代 duppage

另外所有涉及到全局变量 thisenv 的地方全部弃用（包括 ipc_recv）