OSLab5 文档

5140379064 陈俊

一、Pgfault 相关

在 lab 的第一部分，我们在 env_create 函数中添加了对于 ENV_TYPE_FS 的支持，当 type 为 ENV_TYPE_FS 时，我将 eflags 中的 FL_IOPL_MASK 置上，这样这个文件系统 env 就可获得 I/O 的权限。

之后，就是实现 pgfault 的部分。当触发 page fault 时，用 page_alloc 创建一个页，绑定到触发 page fault 的地址（ROUNDDOWN 后）上。再调用 ide_read，将对应的内容读到该页对应的虚拟地址上。

flush_block 实现了将内存中的数据刷回到磁盘的逻辑。当虚拟地址对应页存在且 dirty 位被置上时，通过调用 ide_write，就可以把更新的数据写回磁盘，更新文件系统。之后，也需要重新调用 sys_page_map 修改 perm，将 dirty 位去除。

二、FS block 相关

这一部分主要实现了三个函数：alloc_block、file_block_walk 以及 file_get_block。根据 free_block 的逻辑，在 bitmap 中，空闲块置为 1，而使用的块置为 0。在 alloc_block，只需从 bitmap 后面的块开始进行遍历，通过 block_is_free 函数判断块是否空闲，若空闲，则将改为置为 0，同时 flush_block，更新 bitmap。

File_block_walk 类似于 pgdir_walk 的 file 版本，返回对应 block num 的指针。如果为 direct block 或者 indirect block 存在，只需返回 filebno 对应位置存有 block number 的指针即可。若是 indirect block 不存在，若 alloc 为 1，则调用 alloc_block，再用 memset 将内容置为 0，再返回对应位置的 block num 的指针。

File_get_block 通过 filebno 获得对应 block 在内存中的虚拟地址。通过调用 file_block_walk，如果返回值为 0 且*pdiskbno 不为 0，说明已经该 filebno 已经对应到了一个块，只需通过 diskaddr 获得该块的虚拟地址即可。若*pdiskbno 为 0，说明块并未被分配，因此需要 alloc_block，并将该 block 清零，将 block num 写回到文件系统中该文件的 block num 中，再将该块的虚拟地址返回。

三、RPC 相关

这部分类似于 CSE 课程，写了 rpc 相关的代码。根据图示，serve.c 为服务器端的代码，通过获得 request 的内容，调用文件系统的相关函数，并将结果放在 ret 中。File.c 为客户端的代码，将参数放进 request 中，通过调用 fsipc 函数，将 request 发送给服务器端，通过进程间通信，最终获得返回值。

Read 的逻辑是在 devfile_read 中，通过 fd 获得文件 id，并且将 id 和 n 放入 request 中，调用 fsipc 函数，将 reqeust 发送给服务器端（或者说是文件系统的进程）。在服务器端，调用 file_read 函数，将读到的数据等放在 Fsipc 的 Fsret_read 中，同时更新文件的 offset 等信息。并通过 fsipc 将数据返回给客户端。

Write 的逻辑与 read 如出一辙，在此不再赘述。

客户端的 open 函数也类似，但需要先通过 fd_alloc 函数新建一个 fd，实际上是获得一个 fd 号码。然后通过 fsipc 的 open 远程调用，持有该远程文件的 fd。

四、Spawn 相关

本来我以为这部分需要我们实现一个 spawn 函数，但事实上只需我们添加一个系统调用，允许用户修改进程的 tf。这里，只需在 syscall.c 中通过 envid 获得 env，修改 env 的 tf，之后将其改为用户态和打开系统中断（在 eflags 中置上相应的位即可），就可以实现这一功能。整个 spawn 函数需要在父进程中，将文件的各部分内容映射到子进程的地

址空间中，同时 init_stack，再修改子进程的 tf 和 status。Exec 的实现要麻烦一点，因为是在当前进程调用，冒失地修改地址空间，会影响当前代码的运行。这部分 challenge 的实现我写在了 answers-lab5.txt 中，在此不作赘述。测试文件为 user/exec.c。