

## 作用

`syscall_dup2`是一个系统调用，用于复制一个文件描述符到另一个文件描述符。该函数将打开的文件描述符`old_fd`复制到文件描述符`new_fd`上，并返回新的文件描述符`new_fd`。如果`new_fd`已经打开，则`dup2`会先关闭`new_fd`，然后将`old_fd`复制到`new_fd`上。

`dup2`的主要用途是在进程间传递文件描述符。通过复制文件描述符，一个进程可以将其打开的文件传递给另一个进程，从而实现进程间通信。

`syscall_dup3`多了一个`flags` 参数，用来表示新文件描述符的标志。如果 `flags` 中包含 `O_CLOEXEC` 标记，那么我们会在复制文件描述符之前关闭新文件描述符对应的文件

## 文件描述符

在Linux系统中一切皆文件。如果要对某个设备进行操作，就不得不打开此设备文件，只要你打开文件就会获得该文件的文件描述符`fd`(file discriptor)，这个文件描述符就是一个整数。每个进程在PCB（Process Control Block）中保存着一份文件描述符表，文件描述符就是这个表的索引，每个表项都有一个指向已打开文件的指针。

每个进程都会默认打开3个文件描述符，即0、1、2。其中0代表标准输入流、1代表标准输出流、2代表标准错误流。在编程中通常使用宏`STDIN_FILENO`、`STDOUT_FILENO`和`STDERR_FILENO`分别来代表0，1，2。

## 修改代码

`ulib/axstrarry/src/syscall_fs/imp/io.rs`

```
pub fn syscall_dup3(args: [usize; 6]) -> SyscallResult {
    let fd = args[0];
    let new_fd = args[1];
    let flags = args[2];
    let process = current_process();
    let mut fd_table = process.fd_manager.fd_table.lock();
    if fd >= fd_table.len() {
        debug!("fd {} is out of range", fd);
        return Err(SyscallError::EPERM);
    }
    if fd_table[fd].is_none() {
        debug!("fd {} is not opened", fd);
        return Err(SyscallError::EPERM);
    }
    if fd == new_fd {
        debug!("oldfd is equal to newfd");
        return Err(SyscallError::EINVAL);
    }
    if new_fd >= fd_table.len() {
        if new_fd >= (process.fd_manager.get_limit() as usize) {
            // 超出了资源限制
            return Err(SyscallError::EBADF);
        }
    }
    for _i in fd_table.len()..new_fd + 1 {
        fd_table.push(None);
    }
}
```

```

    }
}
info!("dup3 fd {} to new fd {} with flags {}", fd, new_fd, flags);
fd_table[new_fd] = fd_table[fd].clone();
if flags as u32 & ctypes::O_CLOEXEC != 0 {
    fd_table[new_fd].as_mut().unwrap().set_close_on_exec(true);
}
Ok(new_fd as isize)
}

```

## api/arceos\_posix\_api/src/imp/fd\_ops.rs

```

pub fn sys_dup3(old_fd: c_int, new_fd: c_int, flags: c_int) -> c_int {
    debug!("sys_dup3 <= old_fd: {}, new_fd: {}, flags: {}", old_fd, new_fd,
flags);
    syscall_body!(sys_dup3, {
        if old_fd == new_fd {
            let r = sys_fcntl(old_fd, ctypes::F_GETFD as _, 0);
            if r >= 0 {
                return Ok(old_fd);
            } else {
                return Ok(r);
            }
        }
        if new_fd as usize >= AX_FILE_LIMIT {
            return Err(LinuxError::EBADF);
        }

        let f = get_file_like(old_fd)?;
        FD_TABLE
            .write()
            .add_at(new_fd as usize, f)
            .ok_or(LinuxError::EMFILE)?;

        if (flags & (ctype::O_CLOEXEC as c_int)) != 0 {
            let res = sys_fcntl(new_fd, ctypes::F_SETFD as _, ctypes::FD_CLOEXEC
as _);
        }

        Ok(new_fd)
    })
}

```

具体更改进入下面链接查看

[add sys\\_dup2 and sys\\_dup3 · Arceos-monolithic/Starry@bce5fc0 \(github.com\)](https://github.com/Arceos-monolithic/Starry@bce5fc0)

## 测试

```

#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>

```

```

int main() {
    int src_fd = open("input.txt", O_RDONLY);
    if (src_fd == -1) {
        perror("open");
        return 1;
    }

    int dest_fd = open("output.txt", O_WRONLY | O_CREAT | O_TRUNC, 0666);
    if (dest_fd == -1) {
        perror("open");
        close(src_fd);
        return 1;
    }

    // 使用dup3系统调用将src_fd复制到 dest_fd, 并设置O_CLOEXEC标志
    if (dup3(src_fd, dest_fd, O_CLOEXEC) == -1) {
        perror("dup3");
        close(src_fd);
        close(dest_fd);
        return 1;
    }

    printf("dup3 with O_CLOEXEC succeeded\n");

    // 验证O_CLOEXEC的特性
    if (fcntl(src_fd, F_GETFD) & FD_CLOEXEC) {
        printf("Source file descriptor has O_CLOEXEC flag set\n");
    } else {
        printf("Source file descriptor does not have O_CLOEXEC flag set\n");
    }

    if (fcntl(dest_fd, F_GETFD) & FD_CLOEXEC) {
        printf("Destination file descriptor has O_CLOEXEC flag set\n");
    } else {
        printf("Destination file descriptor does not have O_CLOEXEC flag
set\n");
    }

    close(src_fd);
    close(dest_fd);

    return 0;
}

```

```

/ # ./testdup
9:   file=./testdup [0]; generating link map
9:     dynamic: 0x00000000000004da0  base: 0x00000000000001000  size: 0x00000000000004018
9:     entry: 0x00000000000002100  phdr: 0x00000000000001040  phnum: 13
9:
9:   file=libc.so.6 [0]; needed by ./testdup [0]
9:   file=libc.so.6 [0]; generating link map
9:     dynamic: 0x00000000000021fbc0  base: 0x00000000000006000  size: 0x000000000000228e50
9:     entry: 0x0000000000002ff50  phdr: 0x00000000000006040  phnum: 14
9:
9:   calling init: /lib64/ld-linux-x86-64.so.2
9:
9:   calling init: /lib/libc.so.6
9:
9:   initialize program: ./testdup
9:
9:   transferring control: ./testdup
9:
9:   calling fini: ./testdup [0]
9:
dup3 with O_CLOEXEC succeeded
Source file descriptor has O_CLOEXEC flag set
Destination file descriptor has O_CLOEXEC flag set

```

可以看到源文件和目的文件的O\_CLOEXEC标志都已经设置成功，说明Sys\_dup3有效。

## CI问题

ARRACH CI中DUP2和DUP3找不到对应函数

Jobs

- unit-test
- app-test-for-unikernel (ubuntu-la...
- app-test-for-unikernel (ubuntu-la...
- app-test-for-unikernel (ubuntu-la...
- app-test-for-monolithic (ubuntu-l...
- app-test-for-monolithic (ubuntu-l...
- app-test-for-monolithic (ubuntu-l...

Run details

Usage

Workflow file

```

Run app tests
281 |
282 | = note: `[warn(unused_imports)]` on by default
283 |
284 | error[E0425]: cannot find function `syscall_dup2` in this scope
285 | --> ulib/axstarry/src/syscall_fs/mod.rs:23:17
286 |
287 | 23 |         DUP2 => syscall_dup2(args),
288 |         |
289 |         | help: a function with a similar name exists: `syscall_dup`
290 |
291 | ::: ulib/axstarry/src/syscall_fs/imp/io.rs:272:1
292 | 272 | pub fn syscall_dup(args: [usize; 6]) -> SyscallResult {
293 |     |----- similarly named function `syscall_dup` defined here
294 |
295 | error[E0425]: cannot find function `syscall_dup3` in this scope
296 | --> ulib/axstarry/src/syscall_fs/mod.rs:24:17
297 |
298 | 24 |         DUP3 => syscall_dup3(args),
299 |         |
300 |         | help: a function with a similar name exists: `syscall_dup`
301 |
302 | ::: ulib/axstarry/src/syscall_fs/imp/io.rs:272:1
303 | 272 | pub fn syscall_dup(args: [usize; 6]) -> SyscallResult {
304 |     |----- similarly named function `syscall_dup` defined here
305 |
306 | For more information about this error, try `rustc --explain E0425`.
307 | warning: `axstarry` (lib) generated 1 warning
308 | error: could not compile `axstarry` (lib) due to 2 previous errors; 1 warning emitted
309 | make[1]: *** [scripts/make/build.mk:41: _cargo_build] Error 101
310 | make[1]: Leaving directory '/home/runner/work/Starry/Starry'
311 | test script exited with: 3
312 | make: *** [Makefile:209: test_monolithic] Error 3
313 | Error: Process completed with exit code 2.

```

因为是在ARRACH下跑的Ci，加了`#[cfg(target\_arch = "x86\_64")]`之后只能在x86中使用，在ARRACH里无法识别，因此把dup3的`#[cfg(target\_arch = "x86\_64")]`去掉，DUP2保留，去掉上面重复的

```

root@ubuntu:/home/josen/Starry# git diff
diff --git a/ulib/axstarry/src/syscall_fs/imp/io.rs b/ulib/axstarry/src/syscall_fs/imp/io.rs
index 5bba2934..dfcd4983 100644
--- a/ulib/axstarry/src/syscall_fs/imp/io.rs
+++ b/ulib/axstarry/src/syscall_fs/imp/io.rs
@@ -337,8 +337,7 @@ pub fn syscall_dup2(args: [usize; 6]) → SyscallResult {
    /// * new_fd: usize, 新的文件描述符
    /// * flags: usize, 文件描述符标志
    /// 返回值:成功执行,返回新的文件描述符。失败,返回-1。
-#[cfg(target_arch = "x86_64")]
- pub fn syscall_dup3(args: [usize; 6]) → SyscallResult {
+pub fn syscall_dup3(args: [usize; 6]) → SyscallResult {
    let fd = args[0];
    let new_fd = args[1];
    let flags = args[2];
diff --git a/ulib/axstarry/src/syscall_fs/mod.rs b/ulib/axstarry/src/syscall_fs/mod.rs
index 9a26913a..5aca1ebb 100644
--- a/ulib/axstarry/src/syscall_fs/mod.rs
+++ b/ulib/axstarry/src/syscall_fs/mod.rs
@@ -20,7 +20,6 @@ pub fn fs_syscall(syscall_id: fs_syscall_id::FsSyscallId, args: [usize; 6]) → S
    GETCWD ⇒ syscall_getcwd(args),
    PIPE2 ⇒ syscall_pipe2(args),
    DUP ⇒ syscall_dup(args),
-    DUP2 ⇒ syscall_dup2(args),
    DUP3 ⇒ syscall_dup3(args),
    MKDIRAT ⇒ syscall_mkdirat(args),
    CHDIR ⇒ syscall_chdir(args),
@@ -69,8 +68,6 @@ pub fn fs_syscall(syscall_id: fs_syscall_id::FsSyscallId, args: [usize; 6]) → S
    #[cfg(target_arch = "x86_64")]
    DUP2 ⇒ syscall_dup2(args),
    #[cfg(target_arch = "x86_64")]
-    DUP3 ⇒ syscall_dup3(args),
-    #[cfg(target_arch = "x86_64")]
    LSTAT ⇒ syscall_lstat(args),
    #[cfg(target_arch = "x86_64")]
    OPEN ⇒ syscall_open(args),

```

## 2. 修改格式问题

```

root@ubuntu:/home/josen/Starry# cargo fmt --all -- --check
Diff in /home/josen/Starry/api/arceos_posix_api/src/imp/fd_ops.rs at line 102:

    /// Duplicate a file descriptor.
    pub fn sys_dup3(old_fd: c_int, new_fd: c_int, flags: c_int) → c_int {
-    debug!("sys_dup3 ≤ old_fd: {}, new_fd: {}, flags: {}", old_fd, new_fd, flags);
+    debug!(
+        "sys_dup3 ≤ old_fd: {}, new_fd: {}, flags: {}",
+        old_fd, new_fd, flags
+    );
    syscall_body!(sys_dup3, {
        if old_fd == new_fd {
            let r = sys_fcntl(old_fd, ctypes::F_GETFD as _, 0);
Diff in /home/josen/Starry/ulib/axstarry/src/syscall_fs/imp/io.rs at line 2:
    extern crate alloc;
    use crate::syscall_net::Socket;
    use crate::{IoVec, SyscallError, SyscallResult};
-    use axlibc::ctypes;
    use alloc::string::ToString;
    use alloc::sync::Arc;
    use alloc::vec;
Diff in /home/josen/Starry/ulib/axstarry/src/syscall_fs/imp/io.rs at line 9:
    use axerrno::AxError;
    use axfs::api::{FileIOType, OpenFlags, SeekFrom};
+    use axlibc::ctypes;

    use axlog::{debug, info};
    use axprocess::current_process;
Diff in /home/josen/Starry/ulib/axstarry/src/syscall_fs/imp/io.rs at line 328:
    info!("dup2 fd {} to new fd {}", fd, new_fd);
    // 就算new_fd已经被打开了,也可以被重新替代掉
    fd_table[new_fd] = fd_table[fd].clone();
-    Ok(new_fd as isize)
+    Ok(new_fd as isize)
}

/// 功能:复制文件描述符,并指定了新的文件描述符;

```

