1. 介绍

`prctl` 是一个系统调用，用于在 Linux 系统上控制进程的行为和属性。它允许程序员通过设置不同的参数来修改进程的运行时行为。

`prctl` 可以用于以下目的：

1. **修改进程名称**：通过 `PR_SET_NAME` 参数，可以将进程的名称修改为指定的字符串。
2. **获取和修改进程的资源限制（RLIMIT）**：通过 `PR_GET_RESOURCE` 和 `PR_SET_RESOURCE` 参数，可以获取和修改进程的资源限制，例如最大打开文件数、CPU 时间限制等。
3. **获取和修改进程的信号处理方式**：通过 `PR_GET_PDEATHSIG` 和 `PR_SET_PDEATHSIG` 参数，可以获取和修改父进程终止时发送给子进程的信号。
4. **获取和修改进程的堆栈信息**：通过 `PR_GET_STACK` 和 `PR_SET_STACK` 参数，可以获取和修改进程的堆栈信息。
5. **获取和修改进程的状态标志**：通过 `PR_GET_DUMPABLE` 和 `PR_SET_DUMPABLE` 参数，可以获取和修改进程的状态标志，用于控制是否可以进行 core dump。
6. **获取和修改进程的辅助向量（Auxiliary Vector）**：通过 `PR_GET_AUXV` 参数，可以获取进程的辅助向量，这些向量包含了一些与 ELF 执行文件相关的信息

2.代码

原始代码（ulib/axstarry/src/syscall_task/imp/task.rs -- **syscall_prctl**）

```
#[cfg(target_arch = "x86_64")]
pub fn syscall_prctl(args: [usize; 6]) -> SyscallResult {
    use crate::{PrctlOption, PR_NAME_SIZE};

    let option = args[0];
    let arg2 = args[1] as *mut u8;
    match PrctlOption::try_from(option) {
        Ok(PrctlOption::PR_GET_NAME) => {
            // 获取进程名称。
            let mut process_name = current_task().name().to_string();
            process_name += "\0";
            // [syscall 定义](https://man7.org/linux/man-pages/man2/prctl.2.html)
要求 NAME 应该不超过 16 Byte
            process_name.truncate(PR_NAME_SIZE);
            // 把 arg2 转换成可写的 buffer
            if current_process()
                .manual_alloc_for_lazy((arg2 as usize).into())      //判断虚拟地址
是否在内存集中
                .is_ok()
            // 直接访问前需要确保地址已经被分配
            {
                unsafe {
                    let name = &mut *core::ptr::slice_from_raw_parts_mut(arg2,
PR_NAME_SIZE);

name[..process_name.len()].copy_from_slice(process_name.as_bytes());
                }
                Ok(0)
```

```
            } else {
                Err(SyscallError::EINVAL)
            }
        }
        Ok(PrctlOption::PR_SET_NAME) => Ok(0),
        _ => Ok(0),
    }

}
```

修改代码，添加PR_SET_NAME的具体实现

```
use alloc::{string::{String, ToString}, sync::Arc, vec::Vec};

Ok(PrctlOption::PR_SET_NAME) => {
            if current_process()
                .manual_alloc_for_lazy((arg2 as usize).into())
                .is_ok()
            {
                // Convert arg2 into a mutable reference to an array of u8
(process name buffer)
                unsafe {
                    let name = &mut *core::ptr::slice_from_raw_parts_mut(arg2,
PR_NAME_SIZE);
                    // Obtain the new process name from the buffer
                    let new_name_bytes = name.iter().take_while(|&&c| c !=
0).cloned().collect::<Vec<u8>>();
                    let new_name =
String::from_utf8(new_name_bytes).unwrap_or_default();
                    // Set the new process name
                    current_task().set_name(&new_name);
                    Ok(0)
                }
            } else {
                 Err(SyscallError::EINVAL)
            }
        }
```

3.验证

验证修改进程名称的功能，修改之后再获取该进程的名称，看是否与设定的一致

```
#include <stdio.h>
#include <sys/prctl.h>
#include <errno.h>
#include <string.h>

int main() {
    const char *new_name = "MyTestProcess";  // 新的进程名称

    // 设置进程名称
    if (prctl(PR_SET_NAME, new_name, 0, 0, 0) != 0) {
```

```
        fprintf(stderr, "Failed to set process name: %s\n", strerror(errno));
        return 1;
    }

    // 再次获取进程名称验证
    char process_name[17];  // 16字节 + 终止符

    if (prctl(PR_GET_NAME, process_name, 0, 0, 0) != 0) {
        fprintf(stderr, "Failed to get process name after setting: %s\n",
strerror(errno));
        return 1;
    }

    printf("Process name after setting: %s\n", process_name);

    return 0;

}
```

经验证，进程名称已修改成功

```
/ # ./test_setname
        7:     file=./test_setname [0];  generating link map
        7:       dynamic: 0x0000000000004da0  base: 0x0000000000001000   size: 0x0000000000004030
        7:         entry: 0x0000000000002100  phdr: 0x0000000000001040  phnum:                 13
        7:
        7:
        7:     file=libc.so.6 [0];  needed by ./test_setname [0]
        7:     file=libc.so.6 [0];  generating link map
        7:       dynamic: 0x000000000021fbc0  base: 0x0000000000006000   size: 0x0000000000228e50
        7:         entry: 0x000000000002ff50  phdr: 0x0000000000006040  phnum:                 14
        7:
        7:
        7:     calling init: /lib64/ld-linux-x86-64.so.2
        7:
        7:
        7:     calling init: /lib/libc.so.6
        7:
        7:
        7:     initialize program: ./test_setname
        7:
        7:
        7:     transferring control: ./test_setname
        7:
        7:
        7:     calling fini: ./test_setname [0]
        7:
Process name after setting: MyTestProcess
/ #
```

4.参考资料

linu中使用prctl函数为线程指定名字_prctl使用-CSDN博客

Linux—用prctl()给线程命名_linux prctl-CSDN博客