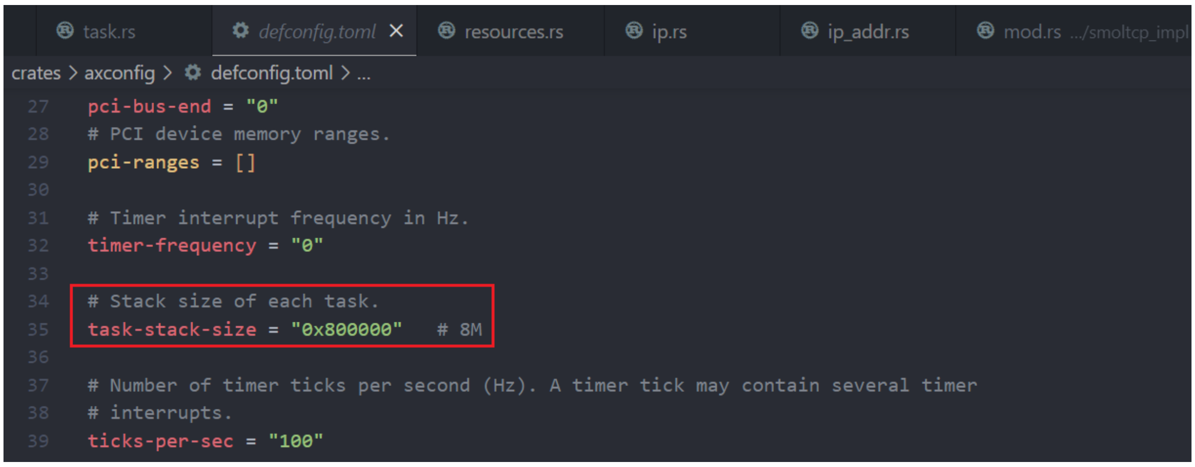
任务介绍

13. prlimit 需要支持设置栈大小 （2周，8.9）
    - 目前的 `syscall_prlimit64` 仅支持修改最大文件数，用户栈大小固定是 256K，但 ffmpeg 的调用是 `prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0` 直接要了 8M 的栈大小。这可能需要动一下 TCB 的结构，把常量改成变量。可以参考 `curr_process.fd_manager.set_limit(new_limit)` 设置文件数，写一个类似 `curr_process.set_stack_limit` 的东西。

源代码

```
pub fn syscall_prlimit64(args: [usize; 6]) -> SyscallResult {
    let pid = args[0];
    let resource = args[1] as i32;
    let new_limit = args[2] as *const RLimit;
    let old_limit = args[3] as *mut RLimit;
    // 当pid不为0，其实没有权利去修改其他的进程的资源限制
    let curr_process = current_process();
    error!("TASK_STACK_SIZE is {}", TASK_STACK_SIZE);
    if pid == 0 || pid == curr_process.pid() as usize {
        match resource {
            RLIMIT_STACK => {
                if old_limit as usize != 0 {
                    unsafe {
                        *old_limit = RLimit {
                            rlim_cur: TASK_STACK_SIZE as u64,
                            rlim_max: TASK_STACK_SIZE as u64,
                        };
                    }
                }
            }
```

TASK_STACK_SIZE是个常量，代表栈大小。可以在/crate/axconfig/defconfig.toml中修改



```
pci-bus-end = "0"
# PCI device memory ranges.
pci-ranges = []

# Timer interrupt frequency in Hz.
timer-frequency = "0"

# Stack size of each task.
task-stack-size = "0x800000"   # 8M

# Number of timer ticks per second (Hz). A timer tick may contain several timer
# interrupts.
ticks-per-sec = "100"
```

由于只在ffmpeg中用到这个特定大小栈，不必将其改为变量，可以在/crates/linux_syscall_api/src/ctypes.rs再添加两个常量来实现定义该栈的大小

修改过后的代码

/crate/linux_syscall_api/src/syscall_task/imp/task.rs

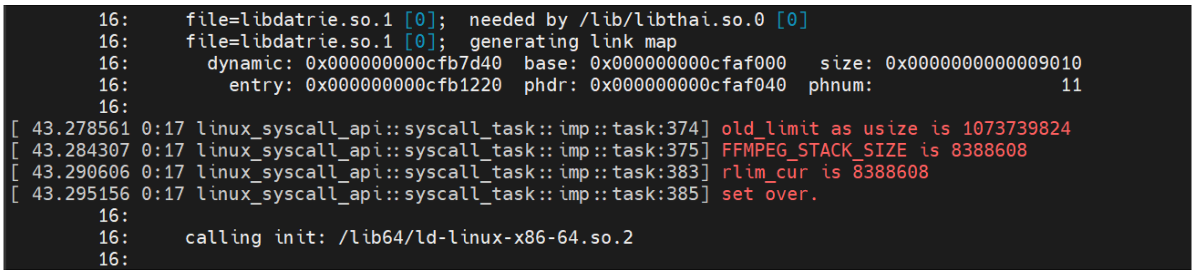
```
pub fn syscall_prlimit64(args: [usize; 6]) -> SyscallResult {
    let pid = args[0];
    let resource = args[1] as i32;
    let new_limit = args[2] as *const RLimit;
    let old_limit = args[3] as *mut RLimit;
    // 当pid不为0，其实没有权利去修改其他的进程的资源限制
    let curr_process = current_process();

if pid == 0 || pid == curr_process.pid() as usize {
    match resource {
        RLIMIT_STACK => {
            if old_limit as usize != 0 {
                if old_limit as usize == APPLY_STACK_SIZE {
                    unsafe {
                        *old_limit = RLimit {
                            rlim_cur: FFMPEG_STACK_SIZE as u64,
                            rlim_max: FFMPEG_STACK_SIZE as u64,
                        };
                    }
                } else {
                    unsafe {
                        *old_limit = RLimit {
                            rlim_cur: TASK_STACK_SIZE as u64,
                            rlim_max: TASK_STACK_SIZE as u64,
                        };
                    }
                }
            }
        }
```

/crates/linux_syscall_api/src/ctypes.rs

```
/// 申请使用的栈大小
pub const APPLY_STACK_SIZE: usize = 0x3ffff830;
/// ffmpeg 使用的栈大小
pub const FFMPEG_STACK_SIZE: usize = 0x800000;
```

实现效果



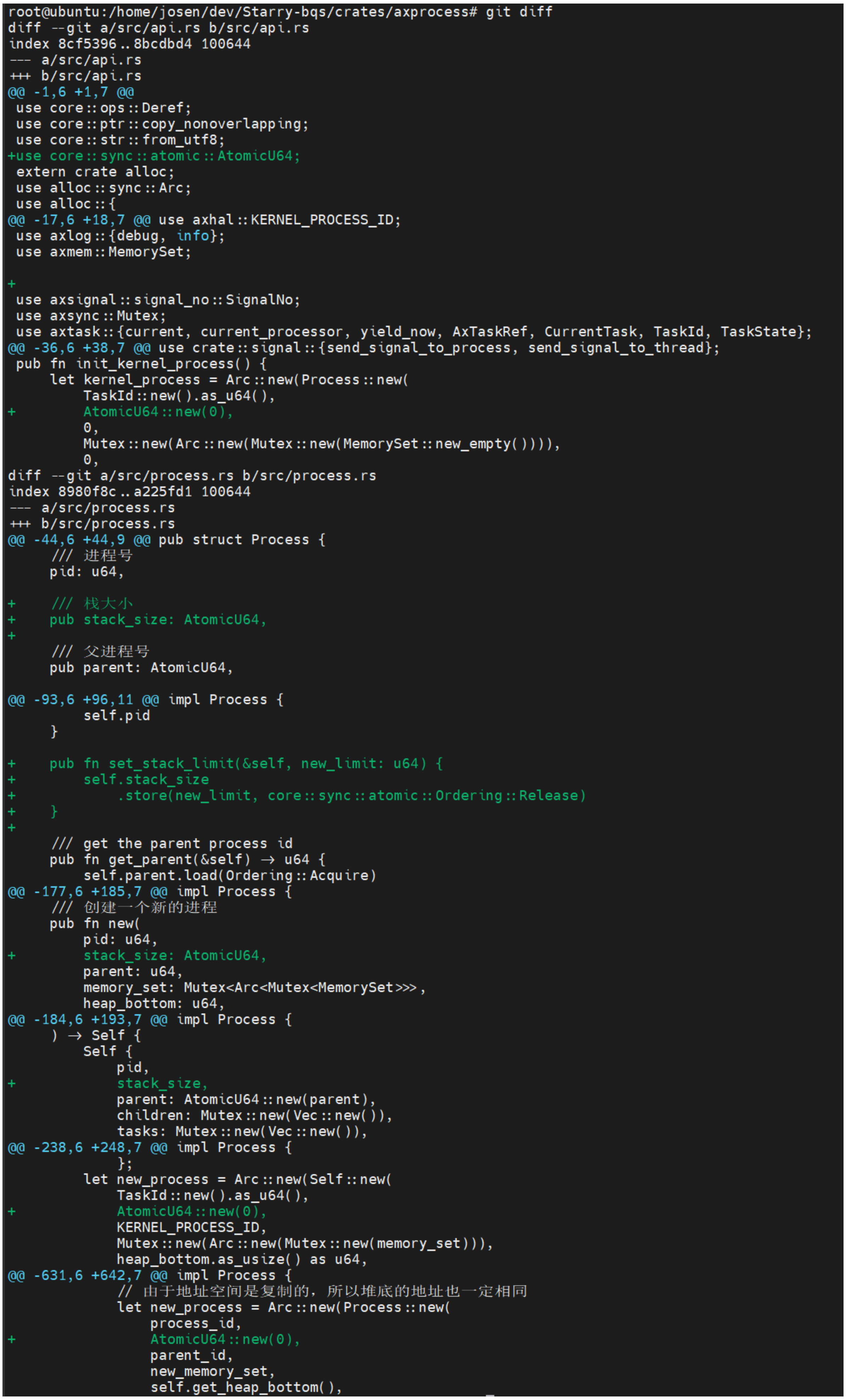
```
        16:     file=libdatrie.so.1 [0];  needed by /lib/libthai.so.0 [0]
        16:     file=libdatrie.so.1 [0];  generating link map
        16:       dynamic: 0x000000000cfb7d40  base: 0x000000000cfaf000   size: 0x0000000000009010
        16:         entry: 0x000000000cfb1220  phdr: 0x000000000cfaf040  phnum:                  11
        16:
[ 43.278561 0:17 linux_syscall_api::syscall_task::imp::task:374] old_limit as usize is 1073739824
[ 43.284307 0:17 linux_syscall_api::syscall_task::imp::task:375] FFMPEG_STACK_SIZE is 8388608
[ 43.290606 0:17 linux_syscall_api::syscall_task::imp::task:383] rlim_cur is 8388608
[ 43.295156 0:17 linux_syscall_api::syscall_task::imp::task:385] set over.
        16:
        16:     calling init: /lib64/ld-linux-x86-64.so.2
        16:
```

commit修改后

/crate/linux_syscall_api/src/syscall_task/imp/task.rs

```
pub fn syscall_prlimit64(args: [usize; 6]) -> SyscallResult {
    let pid = args[0];
    let resource = args[1] as i32;
    let new_limit = args[2] as *const RLimit;
    let old_limit = args[3] as *mut RLimit;
    // 当pid不为0，其实没有权利去修改其他的进程的资源限制
    let curr_process = current_process();
    if pid == 0 || pid == curr_process.pid() as usize {
        match resource {
            // TODO: 改变了新创建的任务栈大小，但未实现当前任务的栈扩展
            RLIMIT_STACK => {
                let mut stack_limit: u64 = curr_process.get_stack_limit();
                if old_limit as usize != 0 {
                    unsafe {
                        *old_limit = RLimit {
                            rlim_cur: stack_limit,
                            rlim_max: stack_limit,
                        };
                    }
                }
                if new_limit as usize != 0 {
                    let new_size = unsafe { (*new_limit).rlim_cur };
                    if new_size > axconfig::TASK_STACK_SIZE as u64 {
                        stack_limit = new_size;
                        curr_process.set_stack_limit(stack_limit);
                    }
                }
            }
```

crates/axprocess/



```
root@ubuntu:/home/josen/dev/Starry-bqs/crates/axprocess# git diff
diff --git a/src/api.rs b/src/api.rs
index 8cf5396..8bcdbd4 100644
--- a/src/api.rs
+++ b/src/api.rs
@@ -1,6 +1,7 @@
 use core::ops::Deref;
 use core::ptr::copy_nonoverlapping;
 use core::str::from_utf8;
+use core::sync::atomic::AtomicU64;
 extern crate alloc;
 use alloc::sync::Arc;
 use alloc::{
@@ -17,6 +18,7 @@ use axhal::KERNEL_PROCESS_ID;
 use axlog::{debug, info};
 use axmem::MemorySet;

+
 use axsignal::signal_no::SignalNo;
 use axsync::Mutex;
 use axtask::{current, current_processor, yield_now, AxTaskRef, CurrentTask, TaskId, TaskState};
@@ -36,6 +38,7 @@ use crate::signal::{send_signal_to_process, send_signal_to_thread};
 pub fn init_kernel_process() {
     let kernel_process = Arc::new(Process::new(
         TaskId::new().as_u64(),
+        AtomicU64::new(0),
         0,
         Mutex::new(Arc::new(Mutex::new(MemorySet::new_empty()))),
         0,
diff --git a/src/process.rs b/src/process.rs
index 8980f8c..a225fd1 100644
--- a/src/process.rs
+++ b/src/process.rs
@@ -44,6 +44,9 @@ pub struct Process {
     /// 进程号
     pid: u64,

+    /// 栈大小
+    pub stack_size: AtomicU64,
+
     /// 父进程号
     pub parent: AtomicU64,

@@ -93,6 +96,11 @@ impl Process {
         self.pid
     }

+    pub fn set_stack_limit(&self, new_limit: u64) {
+        self.stack_size
+            .store(new_limit, core::sync::atomic::Ordering::Release)
+    }
+
     /// get the parent process id
     pub fn get_parent(&self) -> u64 {
         self.parent.load(Ordering::Acquire)
@@ -177,6 +185,7 @@ impl Process {
     /// 创建一个新的进程
     pub fn new(
         pid: u64,
+        stack_size: AtomicU64,
         parent: u64,
         memory_set: Mutex<Arc<Mutex<MemorySet>>>,
         heap_bottom: u64,
@@ -184,6 +193,7 @@ impl Process {
     ) -> Self {
         Self {
             pid,
+            stack_size,
             parent: AtomicU64::new(parent),
             children: Mutex::new(Vec::new()),
             tasks: Mutex::new(Vec::new()),
@@ -238,6 +248,7 @@ impl Process {
             };
         let new_process = Arc::new(Self::new(
             TaskId::new().as_u64(),
+            AtomicU64::new(0),
             KERNEL_PROCESS_ID,
             Mutex::new(Arc::new(Mutex::new(memory_set))),
             heap_bottom.as_usize() as u64,
@@ -631,6 +642,7 @@ impl Process {
             // 由于地址空间是复制的，所以堆底的地址也一定相同
             let new_process = Arc::new(Process::new(
                 process_id,
+                AtomicU64::new(0),
                 parent_id,
                 new_memory_set,
                 self.get_heap_bottom(),
```

具体代码查看

[prlimit set stack size for ffmpeg by Josen-B · Pull Request #3 · Starry-OS/axprocess (github.com)] support ZLMediaKit by Josen-B · Pull Request #7 · Starry-OS/linux_syscall_api (github.com)