项目文档_进程

1 概述

- 小组成员：12110529曹哲振，12110601秦禹洲，12110644周思呈。
- Github仓库链接 https://github.com/Haosonn/Linux0.11_to_ARM 。
- 项目目标为将Linux0.11系统从x86架构移植到ARM架构。选择QEMU模拟器中的virt机器和cotex-a7处理器作为移植目标硬件。主要完成中断、内存和进程模块的实现。
- 项目基于 https://gitee.com/yeyening/linux_0.11_arm_imx6ull 实现。由于本次移植难度较大，上述任务由三个小组合作完成。本小组主要完成了进程部分的实现，主要包括 `schedule(), fork(), exec()` ，此处重点介绍这部分工作。

2 主要工作

2.1 内核调度程序初始化

在x86架构下，内核调度程序的初始化需要在全局描述符表GDT中设置任务0的任务状态段TSS和 LDT，但这两种数据结构是x86硬件支持的，ARM架构下需要重新设计初始化操作。
新的初始化函数 `sched_init()` 主要完成以下操作：

1. 清空任务数组 `task` ；
2. 使能需要用到的时钟中断 `NonSecurePhyTimer_IRQn` ，并将其处理函数设置为 `do_timer()` ；
3. 将 `user_stack` 的值加载到栈指针寄存器 `sp` 中。

2.2 context switch

原先的操作系统中，进程调度工作主要通过 `schedule()` 函数完成。该函数选择下一进程的逻辑主要由c语言实现，只有最后的 `switch_to(next)` 也就是context switch部分是用汇编语言完成。x86架构下的context switch将当前进程的寄存器值存入 `task_struct` 中的tss结构体，再读取新进程的tss值，完成上下文转换。但tss结构体实际上是x86架构中硬件支持的Task State Segment数据结构，位于GDT（Global Descriptor Table）中的一个条目中，而ARM架构中并没有该结构体。因此，我们新建一个 `cpu_context_struct` 结构体用来储存每个进程的寄存器信息。与x86不同，除了r0到r12这些通用寄存器和pc，sp，lr之外，ARM架构还提供了cpsr寄存器用于储存当前进程状态，其中信息具体见图2.1。因此 `cpu-context_struct` 中还需要储存该寄存器的值。

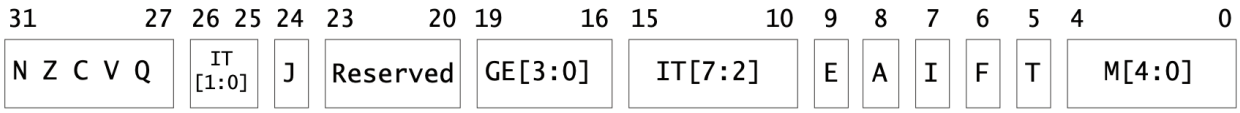

图2.1，cpsr寄存器。

2.3 时钟中断

进程需要每隔一段时间调度一次，该功能需要通过时钟中断完成。主要流程为：

1. 初始化GIC；
2. 初始化硬件时钟中断；
3. 硬件时钟每隔一段时间给CPU发送一个时钟中断，IRQ handler接收该时钟中断并调用 `do_timer()` 函数进行 `schedule()` 。

首先是在 `main` 函数中初始化GIC。GIC (Generic Interrupt Controller)是一种通用中断控制器，用于处理中断请求和向处理器核心分发中断。中断初始化函数中调用 `system_irqtable_init()` 先初始化中断服务函数表，之后每次需要处理IRQ中断时 `system_irqhandler()` 都会再该中断向量表中找到对应函数并跳转。原有仓库的实现中在core_a7.h头文件中通过 `GIC_Type` 结构体描述了GIC v2的各个寄存器位置，但在调试过程中与ARM Generic Interrupt Controller(ARM GIC控制器)V2.0[1]文档内容核对后发现寄存器信息有误，因此调整了其中部分寄存器位置。

然后通过 `enable_Timer()` 函数实现设置时钟。ARM架构下时钟中断主要由协处理器cp15控制，我们使用了其中的三个寄存器。

- CNTHPS_CTL(Counter-timer Secure Physical Timer Control Register)，用于控制物理时钟开关。bits[31:3] reserved保留。bit[2] 表示timer的状态，即是否满足定时器条件。bit[1] IMASK表示时钟中断是否被mask。bit[0] ENABLE表示时钟是否启用。[2] 将IMASK设置为0，ENABLE设置为1，表示启用时钟中断。
- CNTFRQ(Counter-timer Frequency Register)，用于设置系统计数器的频率。32位的寄存器均表示该频率大小。[3]
- CNTHPS_TVAL(Counter-timer Secure Physical Timer TimerValue Register)定时器值寄存器。CNTP_CVAL比较值寄存器 = CNTPCT物理定时器的计数值寄存器 + CNTP_TVAL，当系统计数器值到达比较值寄存器值后，会触发定时信号输出。[4]

初始化完成后通过 `cpsie if` 指令启用全局中断，之后硬件的时钟每隔一段时间会给CPU发送一个中断，通过IRQ handler接收。通过 `sched_init()` 中的设置，该时钟中断会由 `timer_irqhandler()` 通过调用 `do_timer()` 函数处理。`do_timer()` 首先重新设置计时器，然后调用 `schedule()` 函数完成调度。

2.4 fork创建新进程

通过 `fork()` 函数创建新进程。该函数首先通过一个系统调用宏 `_syscall0(type,name)` 触发对应swi系统调用。随后程序会进入 `sys_call_table` 中对应的 `sys_fork()` 函数进行真正的创建新进程操作。

`sys_fork()` 函数实现在kernel目录下的fork.c文件中。首先找到一个空的任务指针，然后将当前进程的寄存器和内存页表复制到新进程的对应位置中。父进程中 `fork()` 函数返回值为子进程pid，子进程则得到0。至此， `task` 数组中多了一个新的进程任务，之后调用 `schedule()` 时就可以调用到新进程。

3 配置和运行

- 采用arm-linux-gnueabihf工具链作为交叉编译器。在ubuntu20.04系统中运行。
- 通过 `make qemu` 命令编译运行项目。
  `-machine virt,gic-version=2` 指定使用virt机器模型，并且GIC版本为2。
  `-m 3G` 设置虚拟机的内存大小为3GB。
  `-cpu cortex-a7` 指定使用Cortex-A7处理器模型。
  `-nographic` 以无图形界面模式运行虚拟机，所有的输入输出通过命令行窗口进行。
  `-kernel Image` 指定内核镜像文件的路径。
  `-serial mon:stdio` 将虚拟机的串行输出重定向到主机的标准输入/输出。
  `-append "console=ttyAMA1 root=/dev/vda rw"` 指定内核启动参数，告诉内核使用ttyAMA1作为控制台设备，将根文件系统挂载到/dev/vda，并以读写模式启动。

其他配置运行细节详见README.md。

---

1. https://developer.arm.com/documentation/ihi0048/latest/ ↩
2. ARM 架构开发者手册， https://developer.arm.com/documentation/ddi0601/2022-03/AArch32-Registers/CNTHPS-CTL--Counter-timer-Secure-Physical-Timer-Control-Register--EL2-?lang=en ↩
3. ARM架构开发者手册， https://developer.arm.com/documentation/ddi0601/2022-03/AArch32-Registers/CNTFRQ--Counter-timer-Frequency-register?lang=en#fieldset_0-31_0 ↩
4. ARM架构开发者手册， https://developer.arm.com/documentation/ddi0601/2022-03/AArch32-Registers/CNTHPS-TVAL--Counter-timer-Secure-Physical-Timer-TimerValue-Register--EL2-?lang=en ↩