OS MP3 Report

B08902055 李英華

May 24, 2021

1. Explain how you implement 3 syscalls.

   (a) `thrdstop`:

   - Set `interval`, `handler_pointer`, `context_id`, initialize `ticks`, and set the corresponding used bit.
   - If `context_id` is -1, find an empty slot in `thrdstop_context[MAX_THRD_NUM]`.
   - In `usertrap()`, increase `ticks` by 1 each time entering the `if(which_dev == 2)` condition, and if `thrdstop_ticks >= thrdstop_interval && thrdstop_interval != -1`, store context in `thrdstop_context[thrdstop_context_id]`. The details of what context I store is discussed in question 2.

   (b) `thrdresume`:

   - If `is_exited` is 0, reload the context in `thrdstop_context[thrdstop_context_id]`.
   - If `is_exited` is not 0, clear the used bit, reset `ticks` and `interval`, and `memset()` to clear `thrdstop_context[thrdstop_context_id]`.

   (c) `cancelthrdstop`:

   - Store current `ticks` for return value, reset `ticks`, `interval` and `handler_pointer`.
   - If `thrdstop_context_id` is not -1, store the context in `thrdstop_context[thrdstop_context_id]`.

2. When you switch to the `thrdstop_handler`, what context do you store? Is it redundant to store all callee and caller registers? Explain your reason.

   - `current_thread` → timer interrupt → `usertrap()` → `usertrapret()` → `thrdstop_handler`.
   - `uservec` saves all the registers in `trapframe`, but the `trapframe` might be replaced by other threads, so we need to store caller-saved registers and other contents needed from `trapframe`.
   - For caller-saved registers, we need to make sure that once we resume the thread, it can correctly return at `ra`, and the `t0-t6` (temporaries), `a0-a7` (function arguments and return value) will be well-preserved.
   - For callee-saved, it is expected to be preserved across procedure calls. So we need to store `sp` and `s0-s11` before switching to handler and restore before return to this thread.

- To sum up, we store all callee and caller registers and other such as `gp` (global pointer) and `tp` (thread pointer). Due to the above reasons, it is not redundant to store all callee and caller registers.

3. Take a look at `struct context` in `/kernel/proc.h`. In context switching for processes, why does it only save callee registers and the ra register?

   - Context switch: user process → system call or interrupt → `usertrap()` → `yield()` → `sched()` → `scheduler()` → `sched()` → `usertrap()` → `usertrapret()` → user process.
   - `uservec` have saved all the registers of user process and jump to `usertrap()`. `pc` is also stored in `trapframe` in `usertrap()`.
   - `context` should stored the registers when actually calling `swtch`, and `swtch` is called in `sched()` and `scheduler()`.
   - Therefore , we only need to store the registers that will change or be needed between these 2 functions ⇒ callee-saved registers and `ra`.
     - Callee-saved registers is expected to be preserved across procedure calls.
     - `ra` holds the return address from which `swtch` was called. During context-switch, `yield() -> sched() -> swtch()`: `yield()` calls `sched()`, and `swtch(&p->context, &mycpu()->context)` will need to return to `scheduler()`, so we need to store `ra`.