MP3

r09922136 廖婕吟

1. Implementations of 3 syscalls

1) `thrdstop()`

We initialize the process's context by the function's parameters at first, including `thrdstop_interval` and `thrdstop_handler_pointer` and set `thrdstop_ticks` to zero since we need to countdown the ticks from zero. Then we update the `thrdstop_context_used` array to keep track of whether we have store the context in the relative index in the `thrdstop_context` array after we find the index we desire to store the context if the parameter `thrdstop_context_id` is -1 or store the context to the index equals to the parameter `thrdstop_context_id` if it is a value between 0 to `MAX_THRD_NUM`. Value 1 in `thrdstop_context_used` array means the relative index in `thrdstop_context` array is used and 0 is unused.

We countdown the ticks and saves the context before jump to handler in the `usertrap()` and `kerneltrap()`. Everytime when we get into the trap after the timer interrupt, we will check the `thrdstop_interval` to know if we need to countdown, if the `thrdstop_interval` is -1, then yield, if it is not -1, then we can add the `thrdstop_ticks` till it achieve the interval value then save the thread's context from the trapframe before jump to handler by `thrdstop_handler_pointer`.

Eventually the `thrdstop()` funciton will return the index of the `thrdstop_context` array where we stored the context before context switch, and if the array is full, it will return -1.

2) `thrdresume()`

We have two parameters in this function which are `thrdstop_context_id` and `is_exit`.

If `is_exit` is zero, we reload the context stored in `thrdstop_context[thrdstop_context_id]` to the trapframe and continue to execute that context.

If `is_exit` isn't zero, we set the `thrdstop_context_used[thrdstop_context_id]` to zero means the `thrdstop_context[thrdstop_context_id]` is empty and can be used to save the new context, Then we cancel the previous `thrdstop()` by setting the `thrdstop_interval` to -1.

3) `cancelthrdstop()`

This function cancels the `thrdstop()`. It save the current thread context into `thrdstop_context[thrdstop_context_id]` if it is valid value between 0 to `MAX_THRD_NUM`, and no need to store if `thrdstop_context_id` is -1. At last, it return the `thrdstop_ticks`.

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

We store the program counter and the CPU registers (e.g. `ra`, `sp`, `gp`, `tp`, saved registers, temp registers, arguments registers). It is **not** redundant to store all callee and caller registers. Since we can't predict user threads' behavior and the user threads may be caller or callee at the occurrence of time interruptions while executing, we can't determine which CPU register we don't need to save. As a result, it is necessary to save all registers if we don't want to lose information and gain the wrong execution result.

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?

`Swtch` don't have to save caller resgisters since the caller-saved registers are already saved on the kernel stack (if needed) by the calling C code before calling `Swtch`.

Because `Swtch` (callee) will ovewrite return address (`ra`: to jump to the instruction from which the new process previously called `Swtch`) and all callee-saved registers with another new process's context while context switching, we need to save the current proc's `ra` and all callee-saved registers.