# Homework 4 : xv6 CPU alarm

## Part 1

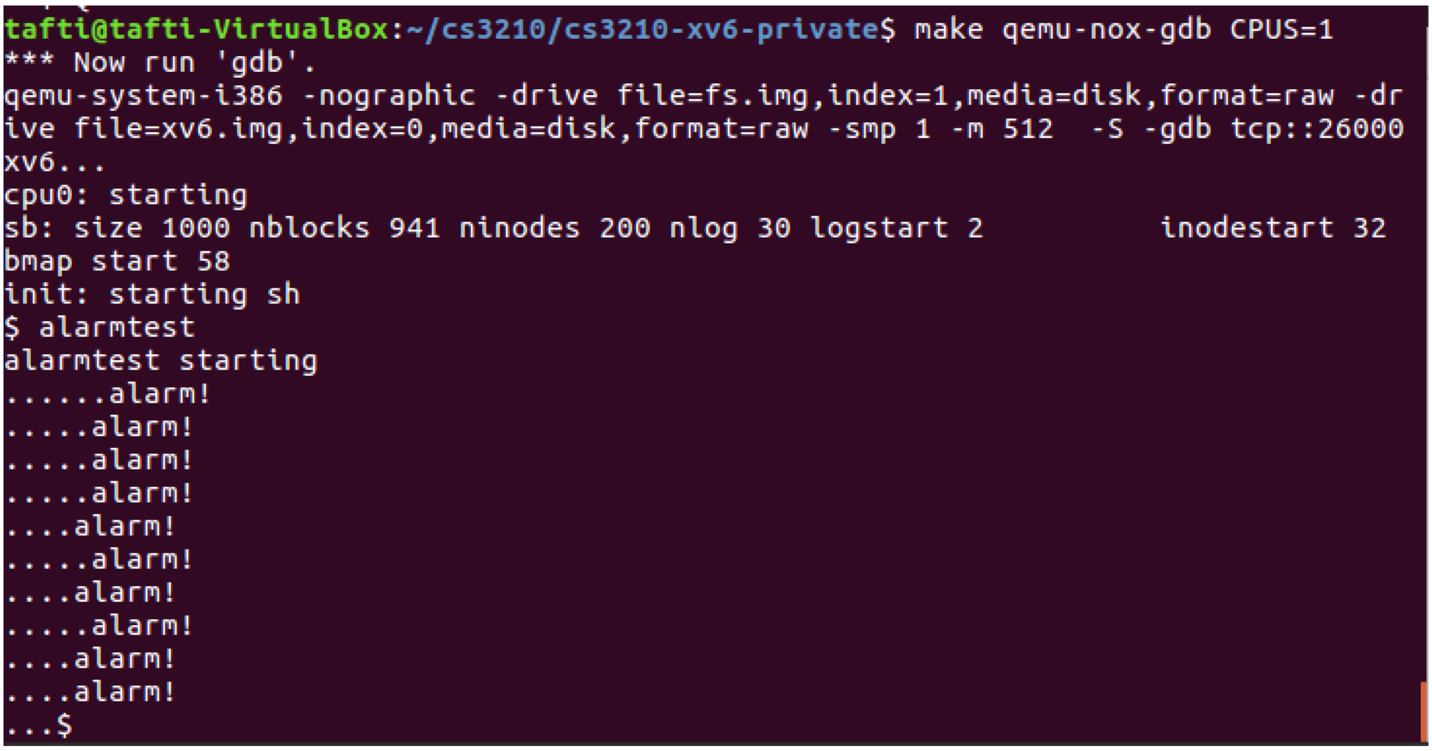
### Adding the syscall and test program

1. Created the file `alarmtest.c` in which I copied the given code
2. `Makefile` : added `_alarmtest` to `UPROGS`
3. `syscall.h` : added syscall number for the date syscall
4. `syscall.c` : declared and added a new `sys_alarm` function to the `syscalls` array.
5. `usys.S` : added `SYSCALL(alarm)`
6. `user.h` : added `alarm` to the list of syscalls.
7. `sysproc.c` : defined the `sys_alarm()` function.

### Implementing `sys_alarm()` , ...

1. `proc.h` :
   - added fields to `struct proc` :
     - `proc->ticks` : interval (in number of ticks) at which we want to call the handler function.
     - `proc->ticksrem` : the number of ticks remaining until the next call.
     - `proc->handler` : the handler function
   - defined a new type `Handler` (function type, I prefer using them that way)
2. `proc.c` : Initialized `ticks` field to `-1` and `handler` field to `(void*)0` *(or `NULL` )* in `allocproc()` .
3. `sysproc.c` : Implemented `sys_alarm()`
4. `trap.c` : Implemented interrupt handler for timer interrupt.

Output :



```
tafti@tafti-VirtualBox:~/cs3210/cs3210-xv6-private$ make qemu-nox-gdb CPUS=1
*** Now run 'gdb'.
qemu-system-i386 -nographic -drive file=fs.img,index=1,media=disk,format=raw -dr
ive file=xv6.img,index=0,media=disk,format=raw -smp 1 -m 512  -S -gdb tcp::26000
xv6...
cpu0: starting
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2         inodestart 32
bmap start 58
init: starting sh
$ alarmtest
alarmtest starting
......alarm!
.....alarm!
.....alarm!
.....alarm!
....alarm!
.....alarm!
....alarm!
.....alarm!
....alarm!
....alarm!
...$
```

## Part 2

### 1. Avoid clobbering registers

Caller-saved registers are `eax` , `ecx` , `edx` (cf. Post @230) on Piazza. We make space for them on the stack and push them.

The solution I implemented to restore them is not the cleanest, but I couldn't get any other way to work, even discussing this with others. It required changing the interface of the `alarm` syscall…

- `proc.h` : added a `handlerwr` field to `proc` to hold the wrapper of my handler. See `alarmtest.c` modif. below for a description of what the wrapper does.
- `proc.c` : initialize `proc->handlerwr` to `NULL`
- `user.h` : modified the `alarm` declaration to also pass the wrapper function pointer
- `alarmtest.c` : added the wrapper, which calls the handler and restores caller saved registers (using inline asm).
- `sysproc.c` : modified to also get the wrapper arguement and store in the dedicated `proc->handlerwr` field.
- `trap.c` : Modified to push the caller-saved registers on the user stack, push the handler function pointer (to be used by the wrapper function), then set `eip` to the wrapper function.

### 2. Avoid reentering the handler

- `types.h` : Added a global variable `int alarm_in_handler`
- `alarmtest.c` : Set / unset the variable
- `trap.c` : Added a check of the variable before handling the interrupt (& thus calling the handler)