OS Lab – 4

Madhav Bhardwaj (CS23B035)

September 2025

Explanation

- First of all as mentioned, make file is edited to compile alarmtest.c
- As every system call, we need to first declare them in user.h for making it user accessible, then add a line in usys.pl script to create a stub. Add syscall identifiers in syscall.h and syscall.c
- **Process structure:** In `proc.h`, we added new fields to track alarm state for each process:

```
int ticks_left;
int handling_alarm;
int alarm_interval;
uint64 alarm_handler;
struct trapframe alarm_tf;
```

  These allow us to remember how often to trigger the alarm, the remaining ticks, the user-space handler function, and a saved trapframe to restore state once the handler completes. The flag `handling_alarm` prevents alarms that occur while handler is executing.

- **in `usertrap()`:** In `trap.c`, inside `usertrap()`, we check for timer interrupts:

```
if (which_dev == 2) {
    if (p->handling_alarm == 0 && p->alarm_interval > 0) {
        p->ticks_left--;
        if (p->ticks_left == 0) {
            p->alarm_tf = *(p->trapframe);
            p->trapframe->epc = p->alarm_handler;
            p->handling_alarm = 1;
            p->ticks_left = p->alarm_interval;
        }
    }
    yield();
}
```

Here, the tick counter is decremented every clock tick. When it reaches zero, the current trapframe is saved, the program counter is redirected to the handler, and the counter is reset. Thus the next time the process returns to user space, it begins executing the handler.

- **sys_sigalarm():** The system call in `sysproc.c` initializes alarm parameters:

```
uint64 sys_sigalarm(void) {
  int ticks; uint64 handler;
  argint(0, &ticks);
  argaddr(1, &handler);
  struct proc* p = myproc();
  p->alarm_interval = ticks;
  p->ticks_left = ticks;
  p->alarm_handler = handler;
  return 0;
}
```

  This sets up how frequently the alarm fires and which function should be invoked.

- **sys_sigreturn():** The handler must call `sigreturn()` when finished, which restores the saved context:

```
uint64 sys_sigreturn(void) {
  struct proc *p = myproc();
  *(p->trapframe) = p->alarm_tf;
  p->handling_alarm = 0;
  return 0;
}
```

  This ensures execution continues exactly from the instruction that was interrupted, with all registers (except `a0`) restored. to ensure that a0 is also restored it must not be overwritten in syscall.c where syscall function is present.

```
p->trapframe->a0 = syscalls[num]();
if (num == SYS_sigreturn) {
  p->trapframe->a0 = p->alarm_tf.a0;
}
```