Kernel threads

In this project you will build a kernel that supports cooperative kernel threads. Here are the kernel components that you will need to copy from xv6 (with modifications):

- Makefile and linker script ([kernel.ld])
- boot sequence: entry, start, main
- console, uart, and printf (only the synchronous parts)
- page allocator (kalloc)
- string library
- process code (kernel side only) including swtch and the scheduler

Delete anything related to the disk or file system, sleeping and waking up, interrupts, transitions to and from userspace, and virtual memory/paging.

Your goal is to able to run 10 threads that all run the following code:

```c
void thread_func(void) {
    for (;;) {
        int cid = cpuid();
        struct proc *proc = myproc();
        if (proc) {
            printf("Running proc %d on cpu %d\n", proc->pid, cid);
        } else {
            printf("Running no proc on cpu %d\n", cid);
        }
        yield();
    }
}
```

This function reports the current process/thread ID and the CPU number, then yields the CPU to another waiting process, then repeats. When running correctly, you should see the 10 threads all rapidly taking turns running. Each thread should eventually run on each CPU.

Here are a few important modifications to the basic xv6 kernel:

- When the process table is initialized, the kernel stack should be set up using its physical address, not a virtual address (since virtual memory is not enabled).
- Instead of a single userspace process, create 10 processes.
- Instead of having a process “return” to [forkret], have it go to another function that releases the process lock (see [forkret] for an example) and then calls [thread_func].
- Much of the process code will need to be deleted as it relates to userspace function or the creation and destruction of processes. This kernel will have 10 processes created manually and they will run forever.
- Every CPU should finish [main] by calling [scheduler], just as in the full version of xv6.