mmap()

In this week's lab, we will manually access the Raspberry Pi's peripheral registers from within a Linux user process. Using the `mmap()` system call, particular parts of RAM can be mapped into a user process. Specifically, we'll map Linux's `/dev/mem` file into the address space of a user process. Once this file is memory mapped, you can access our old friends FSEL1 (0x20200004) and GPSET0 (0x2020001C) to turn on an LED connected to GPIO25.

Memory Mapping

Use the `mmap()` system call to map the 0x20200000 memory location into the process address space. This system call is explained in detail in Chapter 49 of TPI. **Fill in the last four arguments of mmap(). Then, use the gpio_base pointer to turn an LED on and off.**

```
volatile unsigned int *gpio_base, *fsel1, *gpset0;
int memfd = open("/dev/mem", O_RDWR | O_SYNC);
if (memfd < 0) {
     perror("unable to open /dev/mem\n");
     exit(-1);
}
gpio_base = (unsigned int *) mmap(0, 4096, , , ,);
close(memfd);
if (gpio_base == MAP_FAILED){
     perror("unable to mmap\n");
     exit(-1);
}
```

Varying the Brightness

Your program should continuously turn your led on (for time *t1* microseconds) and off (for time *t2* microseconds) with some period (*t1 + t2* microseconds). Your program should take two command line parameters. The first parameter determines how long to keep the LED on, and the second parameter determines the period of the loop.

```
./ledpulse ontime period
```

Deliverables

lastname_lab9/

ledpulse.c C program that turns on the LED with variable brightness.