Raymond You

Pre-Lab Assignment 1

Professor Kimani

**a)**

**Initialize() –**

-Initializes general-purpose I/O.

-open is a system call that is used to open a new file and obtain its file descriptor.

-/dev/mem provides access to the system's physical memory and is the path to the file that is opened.

-O\_RDWR: open for reading and writing

-nmap() creates a new mapping in the virtual address space of the calling process

-**\*mmap(void \*addr, size\_t length, int prot, int flags, int fd, off\_t offset)**

-addr: the starting address for the new mapping. If it is null, the kernel chooses the address at which to create the mapping

-length: the length of the mapping

-prot: describes the desired memory protection of the mapping.

-flags: determines whether updates to the mapping are visible to other processes mapping the same region, and whether updates are carried through to the underlying file

-fd: the file descriptor

-offset: the contents of a file mapping are initialized using length bytes starting at offset

**Finalize() –**

-Close general-purpose I/O

-munmap: map or unmap files or devices into memory

-**int munmap(void \*addr, size\_t len)**

-addr and len: moves any mappings for those entire pages containing any part of the address space of

the process starting at addr and continuing for len bytes

**RegisterRead() –**

-Read a 4-byte value from the specified general-purpose I/O location

-(int \*) casts a variable number to a pointer to an int, then the \* in front deference it

to an int. Pretty much returns pBase + offset

**RegisterWrite() –**

-Write a 4-byte value at the specified general-purpose I/O location.

-Write sizeof(int) bytes into memory starting at pBase + offset

**b)**

/\*\* Changes the state of an LED (ON or OFF)

\* @param pBase base address of I/O

\* @param ledNum LED number (0 to 7)

\* @param state State to change to (ON or OFF)

\*/

void Write1Led(char \*pBase, int ledNum, int state) {

if(ledNum == 0) {

int ledOffset = gpio\_led1\_offset;

RegisterWrite(pBase, ledOffset, state);

}

if(ledNum == 1) {

int ledOffset = gpio\_led2\_offset;

RegisterWrite(pBase, ledOffset, state);

}

if(ledNum == 2) {

int ledOffset = gpio\_led3\_offset;

RegisterWrite(pBase, ledOffset, state);

}

if(ledNum == 3) {

int ledOffset = gpio\_led4\_offset;

RegisterWrite(pBase, ledOffset, state);

}

if(ledNum == 4) {

int ledOffset = gpio\_led5\_offset;

RegisterWrite(pBase, ledOffset, state);

}

if(ledNum == 5) {

int ledOffset = gpio\_led6\_offset;

RegisterWrite(pBase, ledOffset, state);

}

if(ledNum == 6) {

int ledOffset = gpio\_led7\_offset;

RegisterWrite(pBase, ledOffset, state);

}

if(ledNum == 7) {

int ledOffset = gpio\_led8\_offset;

RegisterWrite(pBase, ledOffset, state);

}

}

**c)**

/\*\* Reads the value of a switch

\* - Uses base address of I/O

\* @param pBase base address of I/O

\* @param switchNum Switch number (0 to 7)

\* @return Switch value read

\*/

int Read1Switch(char \*pBase, int switchNum) {

if(switchNum == 0) {

int switchOffset = gpio\_sw1\_offset;

return RegisterRead(pBase, switchOffset);

}

if(switchNum == 1) {

int switchOffset = gpio\_sw2\_offset;

return RegisterRead(pBase, switchOffset);

}

if(switchNum == 2) {

int switchOffset = gpio\_sw3\_offset;

return RegisterRead(pBase, switchOffset);

}

if(switchNum == 3) {

int switchOffset = gpio\_sw4\_offset;

return RegisterRead(pBase, switchOffset);

}

if(switchNum == 4) {

int switchOffset = gpio\_sw5\_offset;

return RegisterRead(pBase, switchOffset);

}

if(switchNum == 5) {

int switchOffset = gpio\_sw6\_offset;

return RegisterRead(pBase, switchOffset);

}

if(switchNum == 6) {

int switchOffset = gpio\_sw7\_offset;

return RegisterRead(pBase, switchOffset);

}

if(switchNum == 7) {

int switchOffset = gpio\_sw8\_offset;

return RegisterRead(pBase, switchOffset);

}

//edge case

if(switchNum > 7 || switchNum < 0) {

return 0;

}

}