Kevin Cao and Dhruvit Naik
CSC 345: Operating Systems
Class 29: File Systems Programming Project – Part 1: Simulated Disk

Problem Statement:
As part of the file systems project, we are tasked to create a virtual disk. A file will act as the disk. The file will be written, read, and created. In order to perform these actions, we will use system calls provided by the operating system.

User Manual:

1. Compilation
   - In order to compile and run, use the makefile provided.
   - Commands
     - clean: remove output files and text files
     - compile: compile disk.c
     - execute: run disk.out
2. How to Use
   When the disk simulation is executed, the disk will first write and read the first 6 blocks sequentially. Once finished, the simulation will prompt the user to read or write blocks one through five. This will show the disk is capable of reading and writing at random.

Design and Implementation:
The file system is designed such that a text file represents the disk, and the contents within the file represent the files on the disk. The disk is divided into blocks with a size of 4096 bytes. To represent a block, we used a 4096 char array. In C, the data type char can store a single byte or 8 bits. The disk was created with the size of 40960 bytes. Thus, the disk contains a total of ten blocks. The size of the disk can be modified with the macro *DISK_SIZE*.

The interfaces to be supported in the file system are *openDisk*, *readBlock*, *writeBlock*, and *syncDisk*. The function *openDisk*, opens a file simulating the disk for reading and writing. The function *readBlock* reads the block block number from the disk into a buffer pointed to by the block. The function *writeBlock* writes the data in block to the disk block block number. Lastly, the function *syncDisk* forces all outstanding writes to the disk.

The *openDisk* function uses the open system call to open the correct disk. The reading, writing and create flags are set within the open command to enable reading and writing. The create flag ensures that if file being opened does not exist, then the system call will create the file. To correctly create a disk of the correct size, the truncate system call is used to specify the size of the disk.

```
/*
        openDisk
        ----------------------------
        opens or creates a disk with nbytes

        filename: the file to be opened or created
        nbytes: size of disk in bytes

        returns: file descriptor of disk
*/

int openDisk(char *filename, int nbytes)
{
        /*
                open and possible create a file
                returns: a file descriptor
                filename: the given pathname for a file
                flags:
                        O_CREAT - if file doesn't exist, it will be created
                        O_RDWD - read and write access
        */
        int fd = open(filename, O_CREAT | O_RDWR );
        if(fd < 0)
        {
                perror("open failed");
                exit(EXIT_FAILURE);
        }

        /*
                given file's size will change to nbytes
        */
        int tr = truncate(filename, nbytes);
        if(tr == -1)
        {
                perror("truncate failed");
                exit(EXIT_FAILURE);
        }
        return fd;
}
```

The *writeBlock* function uses the *lseek* and write system calls to write to the opened disk. The *lseek* function is used to locate the correct block number to write to. This is calculated by multiplying the block number parameter value by 4096 as each block size is 4096 bytes. The write system call then locates the correct location within the disk to write to. It writes the buffer data into the disk.

```
/*
        writeDisk
        ----------------------------
        writes a block from disk

        disk: the disk being writen
        blocknum: the block being writen
        block: buffer being written into disk

        returns: number of bytes writen
*/

int writeBlock(int disk, int blocknum, void *block)
{
        /*
                convert virtual number to logical number
        */
        blocknum = blocknum * 4096;
        /*
                data type to represent file size
        */
        off_t offset = blocknum;
        /*
                for the given fd, it positions the file offset
                SEEK_SET: set by offset bytes, absolute value
        */
        off_t lk = lseek(disk, offset, SEEK_SET);
        if(lk == -1)
        {
                perror("seek failed");
                exit(EXIT_FAILURE);
        }
        /*
                writes to the file represented by the fd
                block: buffer that will be written
                4096: size of the buffer in bytes
        */
        int wr = write(disk, block, 4096);
        if (wr != 4096)
        {
                perror("write failed");
                exit(EXIT_FAILURE);
        }

        return wr;
}
```

The *readBlock* function uses the *lseek* and write system calls to read from the opened disk. The *lseek* function is used to locate the correct block number to read from. The block number is calculated by multiplying the input parameter by a value of 4096 as each block size is 4096 bytes. The read system call is then used to locate the correct location within the disk to read from. The read information is then stored into a buffer.

```
/*
        readDisk
        ----------------------------
        reads a block from disk

        disk: the disk being read
        blocknum: the block being read
        block: buffer being read into

        returns: number of bytes read
*/

int readBlock(int disk, int blocknum, void *block)
{
        /*
                convert virtual number to logical number
        */
        blocknum = blocknum * 4096;
        /*
                data type to represent file size
        */
        off_t offset = blocknum;
        /*
                for the given fd, it positions the file offset
                SEEK_SET: set by offset bytes, absolute value
        */
        off_t lk = lseek(disk, offset, SEEK_SET);
        if(lk == -1)
        {
                perror("seek failed");
                exit(EXIT_FAILURE);
        }
        /*
                reads the file represented by the fd
                block: buffer that will be read into
                4096: read up to 4096 bytes
        */
        int rd = read(disk, block, 4096);
        if (rd != 4096)
        {
                perror("read failed");
                exit(EXIT_FAILURE);
        }
        /*
                write read results to stdout for debugging
        */
        write(1, block, rd);
        printf("\n");
        return rd;
}
```

The *syncDisk* function uses the sync system call. The *syncDisk* function takes no parameters and neither does the sync function. The system call, syncs, writes all pending modifications to file system metadata and cached file data to the underlying file systems.

```
/*
        syncDisk
        ----------------------------
        syncs all outstanding writes
*/

void syncDisk()
{
        /*
                Sync is always successful
        */
        sync();
}
```

Discussion:

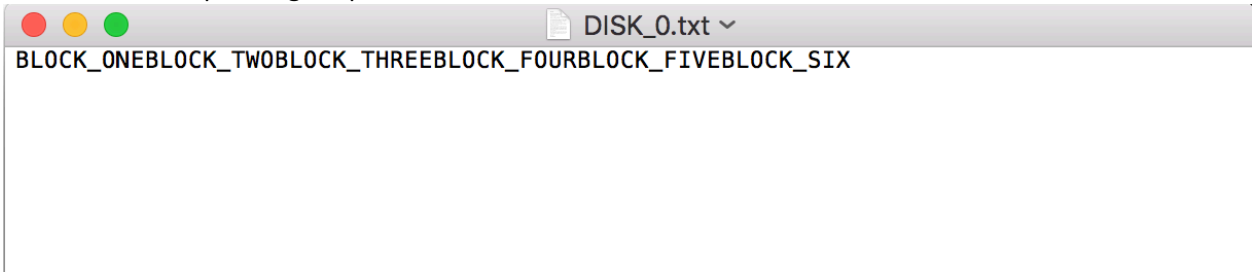
The file system implemented was tested by first testing out the individual functions by themselves. The openDisk function was first tested by opening the file DISK_0.txt which did not initially exist, using

"disk = openDisk(DISK_NAME, DISK_SIZE);"

where DISK_NAME is DISK_0.txt and DISK_SIZE is 40960. After confirming that the file was created, the file was verified if it was the same size as specified in the openDisk function. Next the writeBlock function was tested. The lseek and write system calls were tested. The text file created was checked to ensure that the write function wrote to the appropriate block numbers. The readBlock function was then tested. After writing data into the disk with the writeBlock function, the readBlock function was used to read the data. It was verified that the data being read matched that data written to the data blocks. This test was done be sequentially writing and reading the blocks from block one to block six. Below is the terminal output of the 6 reads and writes.

```
./disk.out
writing block...
reading block...
BLOCK_ONE
writing block...
reading block...
BLOCK_TWO
writing block...
reading block...
BLOCK_THREE
writing block...
reading block...
BLOCK_FOUR
writing block...
reading block...
BLOCK_FIVE
writing block...
reading block...
BLOCK_SIX
```

This is the corresponding output file of the disk.

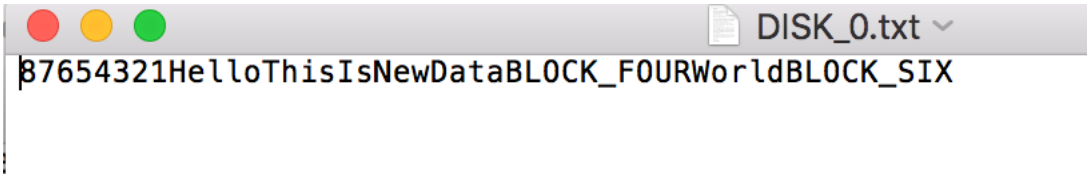


The overall program can be tested interactively. This will test the random writes and reads of the disk. The program will automatically open the DISK_0.txt file as the disk. The user will be able to choose between reading and writing blocks If writing, the user will be prompted to enter the data to be written to the block. If reading, the program will read the block chosen by the user.

Below is the terminal output for random writes and reads.

```
Update Disk
1)Write Blocks 1-5:
2)Read Blocks 1-5:
3)Exit:
1
1-5)Choose what Block to write (1-5):
6)Back:
5
Enter Data to write into Block 5:
World
1-5)Choose what Block to write (1-5):
6)Back:
2
Enter Data to write into Block 2:
Hello
1-5)Choose what Block to write (1-5):
6)Back:
3
Enter Data to write into Block 3:
ThisIsNewData
1-5)Choose what Block to write (1-5):
6)Back:
1
Enter Data to write into Block 1:
87654321
1-5)Choose what Block to write (1-5):
6)Back:
6
1)Write Blocks 1-5:
2)Read Blocks 1-5:
3)Exit:
2
1-5)Choose what Block to read (1-5):
6)Back:
3
ThisIsNewData-----Block 3 Read
1-5)Choose what Block to read (1-5):
6)Back:
2
Hello-----Block 2 Read
1-5)Choose what Block to read (1-5):
6)Back:
5
World-----Block 5 Read
1-5)Choose what Block to read (1-5):
6)Back:
1
87654321-----Block 1 Read
1-5)Choose what Block to read (1-5):
6)Back:
6
1)Write Blocks 1-5:
2)Read Blocks 1-5:
3)Exit:
3
Disk Closed
```

This is the corresponding output file of the disk.



The tests performed can be seen in the code below.

```
int main()
{
	int disk = 0;
	int rd = 0;
	int wr = 0;

	/*
		Open Disk
	*/

	disk = openDisk(DISK_NAME, DISK_SIZE);

	/*
		Read and Write Sequentially Into Disk
	*/
	char buf1[4096] = "BLOCK_ONE";
	printf("writing block...\n");
	wr = writeBlock(disk, 0, buf1);
	printf("reading block...\n");
	rd = readBlock(disk, 0, buf1);

	char buf2[4096] = "BLOCK_TWO";
	printf("writing block...\n");
	wr = writeBlock(disk, 1, buf2);
	printf("reading block...\n");
	rd = readBlock(disk, 1, buf2);

	char buf3[4096] = "BLOCK_THREE";
	printf("writing block...\n");
	wr = writeBlock(disk, 2, buf3);
	printf("reading block...\n");
	rd = readBlock(disk, 2, buf3);

	char buf4[4096] = "BLOCK_FOUR";
	printf("writing block...\n");
	wr = writeBlock(disk, 3, buf4);
	printf("reading block...\n");
	rd = readBlock(disk, 3, buf4);

	char buf5[4096] = "BLOCK_FIVE";
	printf("writing block...\n");
	wr = writeBlock(disk, 4, buf5);
	printf("reading block...\n");
	rd = readBlock(disk, 4, buf5);

	char buf6[4096] = "BLOCK_SIX";
	printf("writing block...\n");
	wr = writeBlock(disk, 5, buf6);
	printf("reading block...\n");
	rd = readBlock(disk, 5, buf6);

	/*
		Update Disk with user input
	*/
	printf("Update Disk\n");
	while(1)
	{
		int i = 0;
		printf("1)Write Blocks 1-5:\n");
		printf("2)Read Blocks 1-5:\n");
		printf("3)Exit:\n");
		scanf("%d", &i);
		if (i==3)
			break;
```

```
            if (i == 1)
            {
                    while(1)
                    {
                            i = 0;
                            char buf[4096] = "";
                            printf("1-5)Choose what Block to write (1-5):\n");
                            printf("6)Back:\n");
                            scanf("%d", &i);

                            if (i==6)
                                    break;

                            printf("Enter Data to write into Block %d:\n", i);
                            scanf("%4096s", buf);
                            wr = writeBlock(disk, i-1, buf);
                    }
            }

            if (i == 2)
            {
                    while(1)
                    {
                            i = 0;
                            char buf[4096] = "";
                            printf("1-5)Choose what Block to read (1-5):\n");
                            printf("6)Back:\n");
                            scanf("%d", &i);

                            if (i==6)
                                    break;

                            printf("-----Block %d Read", i);
                            rd = readBlock(disk, i-1, buf);
                    }
            }
        }
        syncDisk();
        printf("Disk Closed\n");
        close(disk);
    return 0;
}
```