# CS 4/53201 Operating Systems

Project #3

Due via e-mail by 11:59pm on Tuesday, April 15, 2003

## Preliminaries: Nachos File System

Nachos has two file system interfaces. “Stub” file system maps Nachos file system interface onto host Unix filesystem. Stub file system is turned off by default. The second version of the file system interface implements a file system on top of Nachos disk. Nachos disk is a file in the Unix filesytem. Nachos machine simulates accessing a real disk using this file. The disk has one platter divided into tracks. Each track holds a fixed number of sectors. Each sector is of fixed size.

Nachos supports a single top-level directory that stores information about the files in the file system. Each entry contains: filename, file header node, and flag. Filename has fixed size, file header node is a structure similar to Unix i-node, flag indicates whether this directory entry is currently allocated. File header node takes up one sector and contains file’s current size and pointers to physical disk blocks.

Files implementing Nachos file system are located in `filesys` subdirectory of Nachos distribution.

In this project you are to modify the implementation of the file system on the Nachos disk. As distributed Nachos file system requires the file be created before it is used. The file’s length is declared at creation time. The function that creates the file is: `FileSystem::Create(filename, fileLength)`

To keep track of free sectors Nachos stores a bitmap of the sectors of the disk at the beginning of the disk. Member functions of class `BitMap` (defined in `Userprog` subdirectory of Nachos distribution) manipulate this bitmap.

Function that opens the file is `FileSystem::Open.` Object of class `OpenFile` stores information about an open file. Function `OpenFile::Read` reads data from open file and `OpenFile::Write` writes data to open file.

Note that Nachos file system does not need `Close` function since the size of the file is known at creation time. Nachos provides a number of procedures that test the behavior of the file system. These procedures are defined in `fstest.cc` These procedures are invoked when appropriate command-line argument is specified. In particular the following command copies Unix file `test` to Nachos file system.

```
prompt% ./nachos –cp test test
```

Note that before usage Nachos file system needs to be formatted. Use `–f` command-line argument to format Nachos file system.

## Programming assignment

All changes you make to Nachos source code must be clearly marked as follows:

```
//project 3 changes start here
<put your changes here>
//project 3 changes end here
```

The `Makefile` for this project has to be modified as follows. Comment (put # signs in front of) lines

```
cd threads; $(MAKE) depend
cd threads; $(MAKE) nachos
```

Uncomment lines (remove # signs):

```
#       cd filesys; $(MAKE) depend
#       cd filesys; $(MAKE) nachos
```

Your task is to modify Nachos file system interface as follows. Modify `Create` function so that it no longer accepts the file size (the file size is to be determined dynamically through `Write` operations). If the file already exists, the contents of the file should be discarded. Note that the blocks used in the discarded file should be made available. Modify `Open` function such that it accepts the parameter specifying whether the file is opened for reading only or for writing only. If the file is opened for writing – the contents of the file are discarded. Modify `OpenFile` class so that it stores information whether the file is opened for reading and writing. Modify `Read` and `Write` functions so that they do not operate (return –1) if the file is opened in incorrect mode. Note also that `Write` should allocate free sectors to the file if necessary. Create function: `FileSystem::close(OpenFile *).` This function should close the opened file. Note that this function should update the file size in the file header node. Modify functions `Copy` and `Print` in `fstest.cc` so that they work with the modified file system interface functions.

## Submitting Your Project

You have to submit your project to the teaching assistant (Meiduo Wu `mwu@mcs.kent.edu`) by e-mail. She will acknowledge the receipt of your project within a day. Send **only** the files you modified – for example `thread.cc` and `thread.h`. Your submission should be in the form of a tar archive. The command to create the archive file (named `project3.tar` containing files `thread.cc`, `thread.h` and `answers.txt`) is as follows:

```
tar cvf project3.tar thread.cc thread.h answers.txt
```

**Warning:** after you have submitted your project, do NOT touch or modify the files you submitted. If there is a problem with the submission, you will be asked to produce the files and the modification time will be used to determine the time of the submission. Partial submissions are not accepted.

## Grading

The project is worth the total of 50 points. At least 2/3 of the grade for the programming assignment is given if the program works correctly. If the program does not work you may still submit it and explain the reasons what is wrong with the program and why you think it does not work. Put your explanation in `answers.txt`. Partial credit may be assigned. Late projects are accepted. See syllabus for late policies.