Assignment 3 Report

CIS 657 Principles of Operating Systems
Xiaoqian Huang (878174727)

2019/12/04

CIS657 Fall 2019
Assignment Disclosure Form

Assignment #: Assignment 3

Name: Xiaoqian Huang

1. Did you consult with anyone other than instructor or TA/grader on parts of this assignment?
   If Yes, please give the details.
   Yes, about the exact requirements and some design concept.

2. Did you consult an outside source such as an Internet forum or a book on parts of this assignment?
   If Yes, please give the details.
   Yes, referring some design of https://github.com/RyanBeatty/CS420-FileSystems

I assert that, to the best of my knowledge, the information on this sheet is true.

Signature:_____Xiaoqian Huang_______________ Date : 2019/12/4

1. Time Consuming

1) Analyze the problem, determine specifications, create the design: 2 days
2) Implement the design: 2 days
3) Test/debug the program: 4 days (make the program compiled and run, fix logical errors, fix segmentation fault errors, print and analyze large amount of output data to debug).

2. Design/Solution for each requirement

1) Task 1

Syscall

- Create()

The implementation is written at SysCreate() in ksyscall.h.

In SysCreate():

1. call FileSystem::Create(), if it return ture, the create succeeded.

In FileSystem::Create():

1. check the protection bit, if is invalid then return FALSE.
2. parse the input file name and check the privilege of parent directory. If has no write privilege, the request will be denied.
3. check if the file name exists at current directory. If exists, return FALSE.
4. find sector and allocate space for header. If fail, return FALSE.
5. update all the information of created file's header and parent directory's header and write back to disk.

- Remove()

The implementation is written at SysRemove() in ksyscall.h.

In SysRemove():

1. acquire the full path of target file and parent directory. If not exists, return FALSE.
2. check the privilege of both target file and parent directory. If one of them has no read privilege, return FALSE.
3. traverse system-wide openfile table to check if the removed file is opened by any threads.
4. If it is not opened and referenced, it will call FileSystem::Remove().
5. If it is opened and referenced, the file will be added to pending delete list. The list will be checked and removed when a thread exits or a file is closed.

- Open()

The implementation is written at SysOpen() in ksyscall.h.

In SysOpen():

1. acquire the full path of target file and parent directory. If not exists, return FALSE.
2. check the privilege of both target file and parent directory. If one of them has no read privilege, return FALSE.
3. check if the target file is in pending delete list. If it is, return FALSE.
4. check per-process openfile table to find if the file has already been opened by the current process or parent process. If existed, return FALSE.

5. start open the file:
    a) call FileSystem::Open() to open the file. If fail, return FALSE.
    b) add the mapping information to system-wide openfile table and per-process openfile table.
    c) check if target file has corresponding read write lock. If not, update all the data structure corresponding to WR synchronization: write read locks list, read counter list, read count list (for read write problem).
    d) check if target file has corresponding key in openNum list (for remove). If not, add one.
    e) set the open mode.
    f) return the openfile id.

In FileSystem::Open():

1. parse the filename. If fail, return FALSE.
2. create an openfile object and fetch the disk sector of corresponding file.
3. return the openfile object.

- Write()

The implementation is written at SysWrite() in ksyscall.h.

In SysWrite():

1. if the id is console input id (1), acquire the input of user and return.
2. check if current thread has open the file for an open file id. if not, return FALSE.
3. check the open mode of file. Only RW and APPEND mode can write the file.
4. parse the input file name and check the privilege of parent directory. If has no write privilege, the request will be denied.
5. if is APPEND mode, set the call FileSystem::Seek() to set the position.
6. read the data from the memory by using Machine::ReadMem().
7. write the content to disk by using FileSystem::Write().

In FileSystem::Write():

1. call FileSystem::enterWriteRegion() to enter critical region.
2. call FileSystem::WriteAt() to write data;
3. update the position.
4. call FileSystem::leaveWriteRegion() to leave critical region.

- Read()

The implementation is written at SysRead() in ksyscall.h.

In SysRead():

1. if the id is console output id (0), write the intput data to memory.
2. check if current thread has open the file for an open file id. if not, return FALSE.
3. check the open mode of file. Only RO and RW mode can write the file.
4. parse the input file name and check the privilege of parent directory. If has no read privilege, the request will be denied.
5. read the data from disk by using FileSystem::Read().
6. read the data to the memory by using Machine::WriteMem().

In FileSystem::Read():

5. call FileSystem::enterReadRegion() to enter critical region.
6. call FileSystem::ReadAt() to write data;
7. update the position.
8. call FileSystem::leaveReadRegion() to leave critical region.

- Seek()

The implementation is written at SysSeek() in ksyscall.h.

In SysSeek():

1. check if current thread has open the file for an open file id. if not, return FALSE.
2. check the open mode of file. Only RO and RW mode can seek the file.
3. call FileSystem::Seek() to set the position of an openfile object.

- Close()

The implementation is written at SysClose() in ksyscall.h.

In SysClose():

1. check if current thread has open the file for an open file id. if not, return FALSE.
2. update the openNum list, decrease the counter of target file.
3. check if the counter opened from this file is 0. If is, call FileSystem::Remove() to remove the file.

All the system calls return -1 when then are failed.

Synchronization

There are 4 data structure to help implemement the Synchronization:

1. Kernel::readCount: used to record the read count of a file.
2. Kernel::openNumForWR: used to record the open number of a file.
3. Locks::WRLocks: used to map the write read lock and a file.
4. Locks::RCLocks: used to map the read counter lock and a file.

In openfile.cc:
OpenFile::enterWriteRegion():

1. check if the lock is held by current thread. If held by current thread, no need to acquired the lock.
2. if not held by current thread, call Lock::Acquire() to acquire lock.

OpenFile::leaveWriteRegion():

1. check if the lock is held by current thread. If not held by current thread, cannot release.
2. if held by current thread, call Lock::Release() to release lock.

OpenFile::enterReadRegion():

1. check if the lock is held by current thread. If held by current thread, no need to acquired the lock.
2. if not held by current thread, acquire RC lock.

3. increase read count of current referenced file.
4. if read count is 1, block all writers. Acquire the WR lock.
5. release RC lock.

OpenFile::leaveReadRegion():

1. check if the lock is held by current thread. If not held by current thread, cannot release.
2. if held by current thread, acquire RC lock.
3. increase read count of current referenced file.
4. if read count is 0, it means no threads read it anymore, call Lock::Release() to release lock so the writer will not be blocked.
5. release RC lock.

Make maximum size of a file beyond 4Kbytes and Extensible files

To achieve this, I modified filehdr.[h|cc] to change the data structure of file header:

The sector of disk is sperated into four parts:

- 0 for freemap
- 1 for root directory
- 2 ~ 97 for inodes (96)
- 98 ~ 1024 for data blocks (926)

So when we allocate space for files, we just find and set sectors at range (50 , 1024)

FileHeader:
The file header will not refer to data any more. Instead, it is used to store basic information of a file like numbyte of a file, sector number and protection bit. In addition, it store the first FileBlock and last FileBlock of the file.
So the caculation of offset implemeted in FileHeader::ByteToSector() will be different.
FileHeader::Allocate(): allocate data sector for a file with reqiuired size. It also used when expending the file.
FileHeader::Deallocate(): release all the sector allocated.
FileHeader::FetchFrom(): fetch data of FileHeader from the disk at certain sector.
FileHeader::Writeback(): write data of FileHeader to the disk at certain sector.
FileHeader::FileLength(): get data size of the file.
FileHeader::ByteToSector(): calculate the offset and find the exact position of a data.
FileHeader::setProtectionBit(): set protection bit of current file.
FileHeader::getProtectionBit(): get protection bit of current file.
FileHeader::isDir(): check if current header is the header of directory. (used in command line: rmdir)
FileHeader::setDir(): set directory flag. (used in command line: mkdir)

FileBlock:
This structure store number of byte stored, nextblock sector and an array dataSector to store the datablock allocated.
It has following function to manage the storage:
FileBlock::Allocate(): allocate data sector for a file with reqiuired size. Only used when first time allocation of a file.
FileBlock::Deallocate(): release all the sector allocated.
FileBlock::FetchFrom(): fetch data of FileBlock from the disk at certain sector.
FileBlock::Writeback(): write data of FileBlock to the disk at certain sector.

FileBlock::setNectBlock(): set next block sector num.
FileBlock::getNextBlock(): get next block sector num.
FileBlock::FileLength(): get data size allocated.
FileBlock::Expand(): expand the size of file and allocate extra sector for a file with reqiuired size.
FileBlock::ByteToSector(): calculate the offset and find the exact position of a data.

When a file size (position + number of bytes to write) is less than required, the FileHeader will call FileHeader::Allocate() to allocate data block:

The implementation of some functions is as following
FileHeader::Allocate():

1. if size is 0, return directly.
2. check if there is enough space for allocating. If not, return.
3. if has lastblock record, it mean it has data before, so need to call <u>FileBlock::Expand()</u> to allocate more data to the last block.
4. allocate data for more fileblock and datablock:
    a) if firstblock has no sector, it mean it's first time to allocate. No need to link it to last block.
    b) if has data before, link new allocated sector to last fileblock.

5. update information: file size.

FileBlock::Expand():

a) calculate the remain data for last allocatedblock.
b) calculate new required byte number and block number after making use of the remain.
c) allocate block to new data until the dataSectors[] of fileblock is full.

2) Task 2

Hierarchical Directory

Directory need to have ".." to point to parent directory(except root directory) and "." to point to itself.

When a path name is input, it may contain information of directory. The path need to be parse to find the path indicating an exact file. It is implemented at FileSystem::parsePath().

a) It will first treat the path as relative path and find if it is a valid path
b) If not, it will treat it as absolute path.

Each thread record its own current sector and they will be update when using cd.

So both type of path will be supported.

Syscall

To meet all the requirement, a <u>parent-children process table</u> (in addrspace.[h|cc]) and a <u>system-wide process table</u> (in kernel.[h|cc]) will be used and updated in time.

- Exec

To support the command line, when a string is input, it will first parse the string and extract the meaningful strings.

a) If the first string is matching the command lines, the input will be treated as command line and call corresponding functions.
b) If not matching, it will be treated as a path and executed.

When execute an execuatble file, it will add child thread to current thread's children able, and record current thread as new thread's parent.

- Join

A child process id will be passed by.

a) first check if this process id is in current thread's children list. If not, return FALSE.
b) set child thread's waitFlag to TRUE.
c) put current thread to sleep. It will be woken up when all the children's waitFlag is FALSE

- Exit

a) If the current thread has parent, set waitFlag to FALSE;

b) check the children process list of the parent. If all the children's waitFlag is FALSE, wake up the parent.

c) If current thread has no parent, check the openfile list. Decrease all the corresponding open count, and check if it can be deleted. If it can be deleted, call Filesys::Remove() to delete the file.

d) If current thread has parent, removed itself from its parent.

Commands

checkCMD() in ksyscall.h will check if an input string is valid command line. If not

>> ls

FileSystem::List():

It will print all the files and dirs under current sector and their privilege.
>> cd [dir name]

FileSystem::ChangeDir():

1. parse the path and fetch the sector at disk
2. check privilege
3. check if it is a directory
4. update current sector of both thread and file system, as well as updating the directory name to thread (used in pwd).

>> pwd

return current directory stored at address space of the thread. It is maintained in other operations.

>> mkdir [dir name]

<u>FileSystem::MakeDir():</u>

1. parse the path and fetch the sector at disk
2. check privilege
3. check duplication of name
4. allocate space
5. add new path to parent's entry
6. add "..", "." to new path
7. write back all the data.

>> cp [source file name] [target file name]

<u>FileSystem::copyFile():</u>

1. parse the path of both files and fetch the sector at disk
2. fetch the privilege of both files and their parent directory
3. read data from source file
4. write data to target file

>> mv [source file name] [target file name]

<u>FileSystem::moveDir():</u>

1. parse the path of both files and fetch the sector at disk
2. fetch the privilege of both files and their parent directory
3. add target file/dir to new parent directory
4. remove target file/dir to old parent directory
5. update all the information and write back

>> rm [file name]

<u>SysRemove():</u>
mentioned before.

>> rmdir [dir name]

1. check if it is a directory and if it is empty. if not, cannot remove.
2. if it can be removed, call <u>SysRemove()</u>

>> chmod [mode(0-7)] [file name]

1. parse the path of file and fetch the sector at disk
2. fetch the privilege of target file/dir
3. set protection bit by using header's method.

Multiple commands

";" will be parsed and seprated at shell.cc

3. Implementation of the solution

exception.cc

```
void
ExceptionHandler(ExceptionType which)
{
    int type = kernel->machine->ReadRegister(2);
    int result;
    char *string = new char[100];

    DEBUG(dbgSys, "Received Exception " << which << " type: " << type << "\n");

    switch (which) {
    case SyscallException:
      switch(type) {
      case SC_Halt:{
        DEBUG(dbgSys, "Shutdown, initiated by user program.\n");

        SysHalt();

        ASSERTNOTREACHED(); }

              break;

      case SC_Add: {

                        DEBUG(dbgSys,   "Add   "   <<   kernel->machine->ReadRegister(4)   <<   "   +   "   <<
kernel->machine->ReadRegister(5) << "\n");

        /* Process SysAdd Systemcall*/
        result = SysAdd(/* int op1 */(int)kernel->machine->ReadRegister(4),
          /* int op2 */(int)kernel->machine->ReadRegister(5));

        DEBUG(dbgSys, "Add returning with " << result << "\n");
        /* Prepare Result */
        kernel->machine->WriteRegister(2, (int)result);

        /* Modify return point */
        {
          /* set previous programm counter (debugging only)*/
          kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));

          /* set programm counter to next instruction (all Instructions are 4 byte wide)*/
          kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);

          /* set next programm counter for brach execution */
          kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg) + 4);
        }

        return;

        ASSERTNOTREACHED();
      }
              break;

      case SC_SysRead: {
        DEBUG(dbgSys, "SysRead.\n");
        // void SysRead(char* buffer, int size);
        int readbuffer = (int)kernel->machine->ReadRegister(4);
        int readsize = (int)kernel->machine->ReadRegister(5);
        result = SysSRead(readbuffer, readsize); // the content is stored at buffer

        cout << "The read content is: [";

        for (int i = 0; i < readsize; i++) {
          int temp;
          kernel->machine->ReadMem((readbuffer+i), 1, &temp);
          cout << (char)temp << flush;

        }
        cout << "]\n";

        kernel->machine->WriteRegister(2, result); // set return value

              /* Modify return point */
        {
          /* set previous programm counter (debugging only)*/
          kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));

          /* set programm counter to next instruction (all Instructions are 4 byte wide)*/
          kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);

          /* set next programm counter for brach execution */
          kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg) + 4);
        }
      return;
      ASSERTNOTREACHED();
}
  break;

case SC_SysWrite: {
  DEBUG(dbgSys, "SysWrite.\n");
  //void SysWrite(char* buffer, int size);
  int writebuffer = (int)kernel->machine->ReadRegister(4);
  int wirtesize = (int)kernel->machine->ReadRegister(5);

  result = SysSWrite(writebuffer, wirtesize); // the content is stored at buffer
  kernel->machine->WriteRegister(2, result);// set return value

      /* Modify return point */
  {
    /* set previous programm counter (debugging only)*/
    kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));

    /* set programm counter to next instruction (all Instructions are 4 byte wide)*/
    kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);

    /* set next programm counter for brach execution */
    kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg) + 4);
  }
  return;
  ASSERTNOTREACHED();
}
  break;

case SC_SysFork: {
  DEBUG(dbgSys, "Thread Fork.\n");
  // ThreadId ThreadFork(void (*func)());
  int func = (int)kernel->machine->ReadRegister(4);
  result = SysFork(func);
  kernel->machine->WriteRegister(2, result);// set return value

      /* Modify return point */
  {
    /* set previous programm counter (debugging only)*/
    kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));
    /* set programm counter to next instruction (all Instructions are 4 byte wide)*/
    kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);

    /* set next programm counter for brach execution */
    kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg) + 4);
  }
  return;
  ASSERTNOTREACHED();
}
  break;

case SC_Exec: {
  DEBUG(dbgSys, "Exec.\n");
  //SpaceId Exec(char* exec_name);
  //int output = 0;
  int addr = kernel->machine->ReadRegister(4);

  char *execName = getString(string, addr);
  result = checkCMD(execName); // command
  if (result == -1) {
     result = SysExec(execName); // finename
  }

  kernel->machine->WriteRegister(2, result);// set return value
        /* Modify return point */
  {
    /* set previous programm counter (debugging only)*/
    kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));

    /* set programm counter to next instruction (all Instructions are 4 byte wide)*/
    kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);

    /* set next programm counter for brach execution */
    kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg) + 4);
  }
  return;
  ASSERTNOTREACHED();
}
  break;

case SC_Exit: {
  DEBUG(dbgSys, "Exit.\n");
  // void Exit(int status);
  int status = (int)kernel->machine->ReadRegister(4);
  SysExit();
  /* Modify return point */
  {
    /* set previous programm counter (debugging only)*/
    kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));

    /* set programm counter to next instruction (all Instructions are 4 byte wide)*/
    kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);

    /* set next programm counter for brach execution */
    kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg) + 4);
  }
  return;
  ASSERTNOTREACHED();
}
  break;

case SC_Join: {
  DEBUG(dbgSys, "Join.\n");
  // void Exit(int status);
  int childid = (int)kernel->machine->ReadRegister(4);
  result = SysJoin(childid);
  kernel->machine->WriteRegister(2, result);// set return value
  /* Modify return point */
  {
    /* set previous programm counter (debugging only)*/
    kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));

    /* set programm counter to next instruction (all Instructions are 4 byte wide)*/
    kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);

    /* set next programm counter for brach execution */
    kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg) + 4);
  }
  return;
  ASSERTNOTREACHED();
}
  break;
```

```
case SC_Create: {
  DEBUG(dbgSys, "Create.\n");
  // int Create(char *name, int protection);
  int createNameAddr = (int)kernel->machine->ReadRegister(4);
  int createProtection = (int)kernel->machine->ReadRegister(5);

  //int output2 = 0;

  char *createName = getString(string, createNameAddr);
  DEBUG(dbgSys, "name " << createName);
  DEBUG(dbgSys, "pro " << createProtection);
  result = SysCreate(createName, createProtection);
  kernel->machine->WriteRegister(2, result);// set return value
  /* Modify return point */
  {
    /* set previous programm counter (debugging only)*/
    kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));

    /* set programm counter to next instruction (all Instructions are 4 byte wide)*/
    kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);

    /* set next programm counter for brach execution */
    kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg) + 4);
  }
  return;
  ASSERTNOTREACHED();
}
  break;

case SC_Remove: {
  DEBUG(dbgSys, "Remove.\n");
  // int Remove(char *name);
  int removeNameAddr = (int)kernel->machine->ReadRegister(4);

  char* removeName = getString(string, removeNameAddr);

  result = SysRemove(removeName);
  kernel->machine->WriteRegister(2, result);// set return value
  /* Modify return point */
  {
    /* set previous programm counter (debugging only)*/
    kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));
    /* set programm counter to next instruction (all Instructions are 4 byte wide)*/
    kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);

    /* set next programm counter for brach execution */
    kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg) + 4);
  }
  return;
  ASSERTNOTREACHED();
}
  break;

case SC_Open: {
  DEBUG(dbgSys, "Open.\n");
  // OpenFileId Open(char *name, int mode);
  int openNameAddr = (int)kernel->machine->ReadRegister(4);
  int openMode = (int)kernel->machine->ReadRegister(5);

  char *openName = getString(string, openNameAddr);

  DEBUG(dbgSys, "name " << openName);
  DEBUG(dbgSys, "mode " << openMode);

  result = SysOpen(openName, openMode);
  kernel->machine->WriteRegister(2, result);// set return value
  /* Modify return point */
  {
    /* set previous programm counter (debugging only)*/
    kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));

    /* set programm counter to next instruction (all Instructions are 4 byte wide)*/
    kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);

    /* set next programm counter for brach execution */
    kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg) + 4);
  }
  return;
  ASSERTNOTREACHED();
}
  break;

case SC_Write: {
  DEBUG(dbgSys, "Write.\n");
  // int Write(char *buffer, int size, OpenFileId id);
  int writeNameAddr = (int)kernel->machine->ReadRegister(4);
  int writeSize = (int)kernel->machine->ReadRegister(5);
  int writeId = (int)kernel->machine->ReadRegister(6);

  if (writeSize > 100) {
    cout << "size too large\n";
    result = -1;
  }
  else {
    //char *writeContent = getString(string, writeNameAddr);
    result = SysWrite(writeNameAddr, writeSize, writeId);
  }

  kernel->machine->WriteRegister(2, result);// set return value
  /* Modify return point */
  {
    /* set previous programm counter (debugging only)*/
    kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));

    /* set programm counter to next instruction (all Instructions are 4 byte wide)*/
    kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);

    /* set next programm counter for brach execution */
    kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg) + 4);
  }
  return;
  ASSERTNOTREACHED();
}
 break;

case SC_Read: {
  DEBUG(dbgSys, "Read.\n");
  // int Read(char *buffer, int size, OpenFileId id);
  int readNameAddr = (int)kernel->machine->ReadRegister(4);
  int readSize = (int)kernel->machine->ReadRegister(5);
  int readId = (int)kernel->machine->ReadRegister(6);

  if (readSize > 100) {
    cout << "size too large\n";
    result = -1;
  }
  else {
    result = SysRead(readNameAddr, readSize, readId);
  }
  kernel->machine->WriteRegister(2, result);// set return value

  /* Modify return point */
  {
    /* set previous programm counter (debugging only)*/
    kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));

    /* set programm counter to next instruction (all Instructions are 4 byte wide)*/
    kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);

    /* set next programm counter for brach execution */
    kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg) + 4);
  }
  return;
  ASSERTNOTREACHED();
}
  break;

case SC_Seek: {
  DEBUG(dbgSys, "Seek.\n");
  // int Seek(int position, OpenFileId id);
  int seekPos = (int)kernel->machine->ReadRegister(4);
  int seekOpenId = (int)kernel->machine->ReadRegister(5);

  result = SysSeek(seekPos, seekOpenId);
  kernel->machine->WriteRegister(2, result);// set return value
  /* Modify return point */
  {
    /* set previous programm counter (debugging only)*/
    kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));

    /* set programm counter to next instruction (all Instructions are 4 byte wide)*/
    kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);

    /* set next programm counter for brach execution */
    kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg) + 4);
  }
  return;
```

```
    ASSERTNOTREACHED();
  }
    break;

  case SC_Close: {
    DEBUG(dbgSys, "close.\n");
    // int Close(OpenFileId id);
    int closeOpenId = (int)kernel->machine->ReadRegister(4);

    result = SysClose(closeOpenId);
    kernel->machine->WriteRegister(2, result);// set return value
    /* Modify return point */
    {
      /* set previous programm counter (debugging only)*/
      kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));

      /* set programm counter to next instruction (all Instructions are 4 byte wide)*/
      kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);

      /* set next programm counter for brach execution */
      kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg) + 4);
    }
    return;
    ASSERTNOTREACHED();
  }
    break;

  default: {
    cerr << "Unexpected system call " << type << "\n";
  }
        break;
  }
  break;

...
```

ksyscall.h

```
bool CheckAndDelete(int id) {
  string name = string(kernel->openFileTable->find(id)->second->getFullName());
  // check delete list.
  for (int i = 0; i < kernel->pendingDeleteFiles->NumInList(); i++) {
    if (kernel->pendingDeleteFiles->getItem(i) == name) { // in pending delete list
      if ((*kernel->openNumForWR)[name] == 0) { // no file access anymore //after reducing counter of this
        kernel->fileSystem->Remove(kernel->openFileTable->find(id)->second->getFullName()); // remove file
        kernel->openFileTable->erase(id);
        kernel->pendingDeleteFiles->Remove(name);
        return TRUE;
      }
      break;
    }
  }
  return FALSE;
}

void SysHalt()
{
  kernel->interrupt->Halt();
}



int SysAdd(int op1, int op2)
{
  return op1 + op2;
}

int SysSRead(int buffer, int size) { // read content to the buffer
  char *content = "*********default content for SysRead() *********";
  if (size > 48 || size < 0) {
    cout << "invalid input\n";
      return 0;
  }

  for (int i = 0; i < size; i++) {
    kernel->machine->WriteMem(buffer, 1, (int)*(content + i));
    if (i == size - 1) {
      break;
    }
    buffer++;
  }
  return size;
}

int SysSWrite(int buffer, int size) { // write the content of buffer to the console
  int temp;
  if (size > 48 || size < 0) {
    cout << "invalid input\n";
    return 0;
  }

  cout << "Writing to console:";
  for (int i = 0; i < size; i++) {
    kernel->machine->ReadMem(buffer, 1, &temp);
    cout << (char)temp << flush;
    if (i == size - 1) {
      break;
    }
    buffer++;
  }
  cout << "\n";
  return size;
}

void RestoreFunction(int restore) { // restore the state of thread
  kernel->currentThread->RestoreUserState();
  kernel->currentThread->space->RestoreState(); // swap into machine
  kernel->machine->Run();
}

SpaceId SysFork(int func) { // thread fork
  Thread *newThread = new Thread("Forked Thread");
  // page switching
    AddrSpace *newSpace = new AddrSpace(*(kernel->currentThread->space)); // copy the memory of current
thread
  newThread->space = newSpace;
  newThread->SaveUserState(); // save the state of current registers to new thread
  int *registers;
  registers = newThread->getUserRegisters();
  registers[PCReg] = func; // save the PC
  registers[NextPCReg] =  func + 4;
  newThread->Fork((VoidFunctionPtr)RestoreFunction, (void*)0);
  cout << "New pid = " << newThread->pid << "\n";
  return (int)newThread->pid;
}

void ForkFunction(Thread* t) {
  t->space->Execute();
}

SpaceId SysExec(char* exec_name) {
                         char*              fullName              =              kernel->fileSystem->getFullName(exec_name,
kernel->currentThread->space->currentDirSector);
  if (fullName == NULL) {
    return -1;
  }

  // start check privilege //
  char* dirPath = kernel->fileSystem->getParentDirectory(fullName);
  int protectionFileBit = kernel->fileSystem->getProtectionBit(fullName, 1);
  int protectionDirBit = kernel->fileSystem->getProtectionBit(dirPath, 1);

  if (kernel->fileSystem->canExecute(protectionFileBit) == FALSE
    || kernel->fileSystem->canExecute(protectionDirBit) == FALSE) { //check file and dir protection bit
    cout << "Privilege Denied: file - " << protectionFileBit << " dir - " << protectionDirBit << "\n";
    return -1;
  }
  // end check privilege //

  Thread *newThread = new Thread("Exec Thread");
  newThread->space = new AddrSpace();
  int pid = -1;
  if (newThread->space->Load(fullName) == TRUE) { // only exec the prog when the prog exist
    cout << "[" << fullName << "] is opened \n";
    newThread->Fork((VoidFunctionPtr)ForkFunction, (void*)newThread);
    cout << "New pid = " << newThread->pid << "\n";
    pid = newThread->pid;
    kernel->currentThread->childrenTable->Append(newThread); // add to children table
    newThread->parent = kernel->currentThread; // set parent
  }
  else {
    cout << "Cannot open [" << fullName << "]! \n";
  }
  DEBUG(dbgSys, "Parent: " << kernel->currentThread << "name: " << kernel->currentThread->getName());
  DEBUG(dbgSys, "Child: " << newThread << "name: " << newThread->getName());

  return pid;
}

void SysExit() { // update TLB
  cout << "The thread exit\n";

  // wake up the blocked parent process
  if (kernel->currentThread->parent != NULL && kernel->currentThread->waitFlag == TRUE) {
    bool canWake = TRUE;
    kernel->currentThread->waitFlag = FALSE;
    for (int i = 0; i < kernel->currentThread->parent->childrenTable->NumInList(); i++) {
      if (kernel->currentThread->parent->childrenTable->getItem(i)->waitFlag == TRUE) {
        canWake = FALSE;
        break;
      }
    }
    if (canWake == TRUE) {
      IntStatus oldlevel = kernel->interrupt->SetLevel(IntOff);
      kernel->scheduler->ReadyToRun(kernel->currentThread->parent); // all the children is finished, can wake up
the parent
      kernel->interrupt->SetLevel(oldlevel);
    }
  }

  // clean the openfile list (delete each openfile, close each file)
  if (kernel->currentThread->parent == NULL) {
    map<string, int>::iterator iter;
    for (iter = kernel->currentThread->space->openFileTable->begin(); iter !=
kernel->currentThread->space->openFileTable->end(); iter++) {
      string name = iter->first;
      (*kernel->openNumForWR)[name] = (*kernel->openNumForWR)[name] - 1; // decrease the counter for
accessing a file
      if (CheckAndDelete(iter->second) == TRUE) {
        cout << "File " << iter->first << " Deleted.\n";
        kernel->removeFileFromTable(iter->second); // remove from system-wide table
      }
    }
  }

  // update parent-children table
  if (kernel->currentThread->parent != NULL) {
    kernel->currentThread->parent->childrenTable->Remove(kernel->currentThread); // remove this from parent's
children list
  }
  kernel->ProcessTable->erase(kernel->currentThread->pid);

  // terminate the process
  IntStatus old = kernel->interrupt->SetLevel(IntOff);
  kernel->currentThread->Finish();
  kernel->interrupt->SetLevel(old);
  // garbage collection done in destructor
}

int SysJoin(int childid) {
  bool isChild = FALSE;
  Thread* childThread;
  for (int i = 0; i < kernel->currentThread->childrenTable->NumInList(); i++) { // is in current children list
    if (kernel->currentThread->childrenTable->getItem(i)->pid == childid) {
      childThread = kernel->currentThread->childrenTable->getItem(i);
      DEBUG(dbgSys, "Parent: " << kernel->currentThread << "name: " << kernel->currentThread->getName());
      DEBUG(dbgSys, "Child: " << childThread << "name: " << childThread->getName());
      isChild = TRUE;
      break;
    }
  }

  if (isChild == FALSE) {
    DEBUG(dbgSys, "Join: no such child right now.");
    return -1;
  }

  DEBUG(dbgSys, "Parent: " << childThread->parent << "name: " << childThread->parent->getName());
  DEBUG(dbgSys, "Child: " << childThread << "name: " << childThread->getName());

  ASSERT(childThread->parent == kernel->currentThread);
  if (childThread->waitFlag == TRUE) {
    cout << "The child " << childid << " process has been joined!\n";
    return - 1;
  }

    DEBUG(dbgSys, "child process " << childThread->getName() << " is joined to parent process " <<
kernel->currentThread->getName());
  childThread->waitFlag = TRUE; // indicating the parent is blocked and waiting to be awaked.
  IntStatus oldlevel =  kernel->interrupt->SetLevel(IntOff);
  kernel->currentThread->Sleep(FALSE); // block the parent
  kernel->interrupt->SetLevel(oldlevel);

  return 1;
}

int SysCreate(char *name, int protection) {
  // privilege of current dir checked in Create()
    if (kernel->fileSystem->Create(name, 0, kernel->currentThread->space->currentDirSector, protection) == FALSE) {
    cout << "Create Fail.\n";
    return -1;
  }
  // success
  cout << "Create Success: " << name << "\n";
  return 1;
}

int SysRemove(char *name) {
  char* fullName = kernel->fileSystem->getFullName(name, kernel->currentThread->space->currentDirSector);
  if (fullName == NULL) {
    return -1;
  }

  // start check privilege //
  string path = string(fullName);
  char* dirPath = kernel->fileSystem->getParentDirectory(fullName);
  int protectionFileBit = kernel->fileSystem->getProtectionBit(fullName, 1);
  int protectionDirBit = kernel->fileSystem->getProtectionBit(dirPath, 1);

  if (kernel->fileSystem->canRead(protectionFileBit) == FALSE
    || kernel->fileSystem->canRead(protectionDirBit) == FALSE) { //check file and dir protection bit
    cout << "Privilege Denied: file - " << protectionFileBit << " dir - " << protectionDirBit << "\n";
    return -1;
  }
  // end check privilege //

  bool isOpened = FALSE;
  if (kernel->openFileTable->size() > 0) {
    map<int, OpenFile*>::iterator iter;
    for (iter = kernel->openFileTable->begin(); iter != kernel->openFileTable->end(); iter++) {
      if (iter->second != NULL) {
        string str = string(iter->second->getFullName());
        DEBUG(dbgSys, "table: " << str << " current: " << path);
        if (str == path) {
          isOpened = TRUE;
          break;
        }
      }
    }
  }

  if (isOpened == FALSE) { // no thread access to it, delete directly
    kernel->fileSystem->Remove(fullName);
  }

  DEBUG(dbgSys, "file " << fullName << " still open. Added to pending delete list.");

  kernel->pendingDeleteFiles->Append(path); // add to pending delete list, will delete when no other file access to
  return 1;
}

OpenFileId SysOpen(char *name, int mode) {
  char* fullName = kernel->fileSystem->getFullName(name, kernel->currentThread->space->currentDirSector);
  if (fullName == NULL) {
    return -1;
  }

  // start check privilege //
  string path = string(fullName);
  char* dirPath = kernel->fileSystem->getParentDirectory(fullName);
  int protectionFileBit = kernel->fileSystem->getProtectionBit(fullName, 1);
  int protectionDirBit = kernel->fileSystem->getProtectionBit(dirPath, 1);

  if (kernel->fileSystem->canRead(protectionFileBit) == FALSE
    || kernel->fileSystem->canRead(protectionDirBit) == FALSE) { //check file and dir protection bit
    cout << "Privilege Denied: file - " << protectionFileBit << " dir - " << protectionDirBit << "\n";
    return -1;
  }
  // end check privilege //

  if (kernel->pendingDeleteFiles->IsInList(path) == TRUE) { // if the file is deleted
    cout << "Cannot open. The file is deleted.\n";
    return -1;
  }

  if (kernel->currentThread->space->openFileTable->find(path) != kernel->currentThread->space->openFileTable->end()) { // existed in thread list
    cout << "The file has been opened!\n";
    return -1;
  }

  OpenFile *file = kernel->fileSystem->Open(name);

  if (file == NULL) {
    cout << "Cannot find the file.\n";
    return -1;
  }

  // start opening
  kernel->insertFileToTable(file); // add to system-wide table
   (*kernel->currentThread->space->openFileTable)[file->getFullName()] = file->getOpenfileId(); // add to thread table
  DEBUG(dbgSys, "file name " << file->getFullName());
  string filename = string(file->getFullName()); //update WR tables

  if (kernel->locks->WRLocks->find(filename) == kernel->locks->WRLocks->end()) { // no such lock
    (*kernel->locks->WRLocks)[filename] = new Lock(file->getFullName());// create rw lock
    (*kernel->readCount)[filename] = 0;// create rc
    (*kernel->locks->RCLocks)[filename] = new Lock(file->getFullName());// create rc lock
  }

  if ((*kernel->openNumForWR).find(filename) != (*kernel->openNumForWR).end()) { // file exist in map
    (*kernel->openNumForWR)[filename] == (*kernel->openNumForWR)[filename] + 1;
  }
  else {
    (*kernel->openNumForWR)[filename] = 1;
  }

  // set mode
  file->setMode(mode);
  DEBUG(dbgSys, "openfile id " << file->getOpenfileId());
  return file->getOpenfileId(); // return open file id
}

int SysWrite(int buffer, int size, OpenFileId id) {
  if (id == CONSOLEOUTPUT) { // console out
    for (int i = 0; i < size; i++) {
      int temp;
      kernel->machine->ReadMem(buffer, 1, &temp);
      kernel->synchConsoleOut->PutChar((char)temp);
      if (i == size - 1) {
        break;
      }
      buffer++;
    }
    return size;
  }

  if (kernel->currentThread->space->isExisted(id) == FALSE) { // cannot find in thread table
    cout << "Fail, no such id.\n";
    return -1;
  }

  if (kernel->openFileTable->find(id)->second->getMode() == 1) { // 2: RW, 3: APPEND
    cout << "Fail, mode: " << kernel->openFileTable->find(id)->second->getMode();
    return -1;
  }

  // start check privilege //
  char* fullName = kernel->openFileTable->find(id)->second->getFullName();
  string path = string(fullName);
  char* dirPath = kernel->fileSystem->getParentDirectory(fullName);
  int protectionFileBit = kernel->openFileTable->find(id)->second->getProtectionBit();
  int protectionDirBit = kernel->fileSystem->getProtectionBit(dirPath, 1);

  if (kernel->fileSystem->canRead(protectionFileBit) == FALSE
    || kernel->fileSystem->canRead(protectionDirBit) == FALSE) { //check file and dir protection bit
    cout << "Privilege Denied: file - " << protectionFileBit << " dir - " << protectionDirBit << "\n";
    return -1;
  }
  // end check privilege //

  if (kernel->openFileTable->find(id)->second->getMode() == 3) { // append mode
    kernel->openFileTable->find(id)->second->Seek(kernel->openFileTable->find(id)->second->Length() - 1);
  }
  DEBUG(dbgSys,  "File length: " <<kernel->openFileTable->find(id)->second->Length());

  char *content = new char[100];
```

```
  int temp;
  int i;
  for (i = 0; i < size; i++) {
    kernel->machine->ReadMem(buffer, 1, &temp);
    content[i] = (char)temp ;
    if (i == size - 1) {
      break;
    }
    buffer++;
  }
  i++;
  content[i] = '\0';

  size = kernel->openFileTable->find(id)->second->Write(content, size);
  DEBUG(dbgSys, "[Writing to file: " << content << "]");
  return size;
}


int SysRead(int buffer, int size, OpenFileId id) {
  if (id == CONSOLEINPUT) { // console in
    char str;
    for (int i = 0; i < size; i++) {
      str = kernel->synchConsoleIn->GetChar(); // get the char
      kernel->machine->WriteMem(buffer, 1, (int)str); // write to buffer
      if (i == size - 1) {
        break;
      }
      buffer++;
    }
    return size;
  }


  if (kernel->currentThread->space->isExisted(id) == FALSE) { // cannot find in thread table
    cout << "Fail, no such id.\n";
    return -1;
  }

  if (kernel->openFileTable->find(id)->second->getMode() == 3 ) { // 2: RW, 1: RO
    cout << "Fail, mode: " << kernel->openFileTable->find(id)->second->getMode() << "\n";
    return -1;
  }

  // start check privilege //
  char* fullName = kernel->openFileTable->find(id)->second->getFullName();
  string path = string(fullName);
  char* dirPath = kernel->fileSystem->getParentDirectory(fullName);
  int protectionFileBit = kernel->openFileTable->find(id)->second->getProtectionBit();
  int protectionDirBit = kernel->fileSystem->getProtectionBit(dirPath, 1);

  if (kernel->fileSystem->canRead(protectionFileBit) == FALSE
    || kernel->fileSystem->canRead(protectionDirBit) == FALSE) { //check file and dir protection bit
    cout << "Privilege Denied: file - " << protectionFileBit << " dir - " << protectionDirBit << "\n";
    return -1;
  }
  // end check privilege //

  char *content = new char[100];

  size = kernel->openFileTable->find(id)->second->Read(content, size);

  //read content
  for (int i = 0; i < size; i++) {
    kernel->machine->WriteMem(buffer, 1, (int)*(content + i)); // write to buffer
    if (i == size - 1) {
      break;
    }
    buffer++;
  }
  content[size] = '\0';

  DEBUG(dbgSys, "size: " << size );
  DEBUG(dbgSys, "[Reading from file: " << content << "]");

  return size;
}

int SysSeek(int position, OpenFileId id) {
  if (kernel->currentThread->space->isExisted(id) == FALSE) { // cannot find in thread table
    cout << "Fail, no such id.\n";
    return -1;
  }
  if (kernel->openFileTable->find(id)->second->getMode() == 3) { // 2: RW, 1: RO
    cout << "Fail, mode: " << kernel->openFileTable->find(id)->second->getMode() << "\n";
    return -1;
  }

  kernel->openFileTable->find(id)->second->Seek(position);
  DEBUG(dbgSys, "[The current position is: " << position << "]\n");
  return 1;
}

int SysClose(OpenFileId id) {
  if (kernel->currentThread->space->isExisted(id) == FALSE) { // cannot find
    cout << "Fail, no such id.\n";
    return -1;
  }

  string name = string(kernel->openFileTable->find(id)->second->getFullName());
    (*kernel->openNumForWR)[name] = (*kernel->openNumForWR)[name] - 1; // decrease the counter for
accessing a file

  if (CheckAndDelete(id) == TRUE) { // check if the file is pending deleted
    cout << "In close: file "<< id <<" Deleted.\n";
    kernel->removeFileFromTable(id); // remove from system-wide table
  }
  kernel->currentThread->space->removeById(id);// remove from per-process table

  return id;
}

char* getString(char *str, int addr) {
  int output = 0;
  int count = 0;
  kernel->machine->ReadMem(addr, 1, &output);
  str[0] = (char)output;
  while ((char)output != '\0') {
    count++;
    kernel->machine->ReadMem(addr + count, 1, &output);
    str[count] = (char)output;
  }
  return str;
}
```

```
int checkCMD(char *str) {
  string s = string(str);
  string cmd[10];
  size_t pos = s.find(' ');
  int i = 0;
  while(pos != string::npos) {
    if (s.substr(0, pos) != "") { // filter the space
      cmd[i] = s.substr(0, pos);
      i++;
    }
    s = s.substr(pos+1);
    pos = s.find(' ');
  }

  cmd[i] = s;

  DEBUG(dbgFile, "0 [" << cmd[0] << "] 1 [" << cmd[1] << "] 2 [" << cmd[2] << "]\n");

  if (cmd[0] == "ls") {
    kernel->fileSystem->List();
  }
  else if (cmd[0] == "cd") {
    if (i < 1) {
      cout << "Too few argument.\n";
      return 0;
    }
    char* filename = new char[cmd[1].length() + 1]; // arg1
    strcpy(filename, cmd[1].c_str());
     int sector = kernel->fileSystem->ChangeDir(&filename, kernel->currentThread->space->currentDirSector); //
fullpath will be output at filename
    if (sector == -1) {
      return 0;
    }
    else { // update thread info
      kernel->currentThread->space->currentDirSector = sector;
      kernel->currentThread->space->currentDir = kernel->fileSystem->getFullName(sector);
    }
  }
  else if (cmd[0] == "pwd") {
    cout << kernel->currentThread->space->currentDir << "\n";
  }
```

```
 else if (cmd[0] == "mkdir") {
   if (i < 1) {
     cout << "Too few argument.\n";
     return 0;
   }
   char* filename = new char[cmd[1].length() + 1]; // arg1
   strcpy(filename, cmd[1].c_str());
    bool success = kernel->fileSystem->MakeDir(&filename, kernel->currentThread->space->currentDirSector); //
fullpath will be output at filename
   if (success == FALSE) {
     return 0;
   }
   cout << filename << " is created\n";
 }
 else if (cmd[0] == "cp") {
   if (i < 2) {
     cout << "Too few argument.\n";
     return 0;
   }

   char* from = new char[cmd[1].length() + 1]; // arg1
   strcpy(from, cmd[1].c_str());

   char* to = new char[cmd[2].length() + 1]; // arg2
   strcpy(to, cmd[2].c_str());

   if (kernel->fileSystem->copyFile(from, to) == FALSE) {
     cout << "Canceled.\n";
     return 0;
   }

   cout << "Success!\n";
   return 0;
 }
 else if (cmd[0] == "mv") {
   if (i < 2) {
     cout << "Too few argument.\n";
     return 0;
   }

   char* from = new char[cmd[1].length() + 1]; // arg1
   strcpy(from, cmd[1].c_str());
```

```
  char* to = new char[cmd[2].length() + 1]; // arg2
  strcpy(to, cmd[2].c_str());

  if (kernel->fileSystem->moveDir(from, to) == FALSE) {
    cout << "Canceled.\n";
    return 0;
  }

  cout << "Success!\n";
  return 0;
}
else if (cmd[0] == "rm") {
  if (i < 1) {
    cout << "Too few argument.\n";
    return 0;
  }
  char* filename = new char[cmd[1].length() + 1]; // arg1
  strcpy(filename, cmd[1].c_str());
  SysRemove(filename);
}
else if (cmd[0] == "rmdir") {
  if (i < 1) {
    cout << "Too few argument.\n";
    return 0;
  }
  char* filename = new char[cmd[1].length() + 1]; // arg1
  strcpy(filename, cmd[1].c_str());
  if (kernel->fileSystem->isEmptyDir(filename, kernel->currentThread->space->currentDirSector) == FALSE) {
    return 0;
  }
  SysRemove(filename);
}
else if (cmd[0] == "chmod") {
  //arg 1 = protection bit(int)
  //arg 2 = program name
  if (i < 2) {
    cout << "Too few argument.\n";
    return 0;
  }
  char* filename = new char[cmd[2].length() + 1]; // arg2
  strcpy(filename, cmd[2].c_str());
```

```
    int protectionBit = atoi(cmd[1].c_str());    //atoi(cmd[1].c_str())
    if (kernel->fileSystem->setProtectionBit(filename, protectionBit) == FALSE) {
      return 0;
    }
    cout << "Success, Protection Bit: " << protectionBit << "\n";
  }
  else if (cmd[0] == "") { //  cmd is empty
    cout << "No command line\n";
    return 0;
  }
  else {
    return -1;
  }
  return 0;
}

#endif /* ! __USERPROG_KSYSCALL_H__ */
```

filesys.h

```
#else // FILESYS
class Locks;
class FileSystem {
  public:
    FileSystem(bool format);                    // Initialize the file system.
                                                // Must be called *after* "synchDisk"
                                                // has been initialized.
                                                // If "format", there is nothing on
                                                // the disk, so initialize the directory
                                                // and the bitmap of free blocks.

    bool Create(char *name, int initialSize, int currentDirSector, int protection);
                                                // Create a file (UNIX creat)

    OpenFile* Open(char *name);     // Open a file (UNIX open)

    bool Remove(char *name);                    // Delete a file (UNIX unlink)

    void List();                    // List all the files in the file system

    void Print();                   // List all the files and their contents

    int ChangeDir(char **name, int currentDirSector); // cd

    bool MakeDir(char **name, int currentDirSector); // mkdir

    int moveDir(char* from, char *to); // mv

    int copyFile(char* from, char *to); // cp

    void removeDirEntry(char *fullDirpath);

    int parsePath(char **name, int sector); // handle path

    int processPath(char **name, int sector); // get file name and get the dir sector

    void setCurrentDirSector(int sector); // set current dir section for file system

    int getCurrentDirSector();

    char* getFullName(char *name, int sector); // get full path of a path

    int getProtectionBit(char *name, int sector); // get protection bit of a path

    bool canWrite(int bit); // can write?

    bool canRead(int bit); // can read?

    bool canExecute(int bit); // can execute?

    char* getParentDirectory(char* fullPath);

    bool setProtectionBit(char* fullPath, int bit);

    char*  getFullName(int sector); // get full path of a sector

    bool isEmptyDir(char* path, int sector);

  private:
   OpenFile* freeMapFile;           // Bit map of free disk blocks, // 1024 blocks. 0 1
                                                // represented as a file
   OpenFile* directoryFile;         // "Root" directory -- list of
                                                // file names, represented as a file
   int currentDirSector;

};

#endif // FILESYS

#endif // FS_H
```

filesys.cc

```
#include "copyright.h"
#include "debug.h"
#include "disk.h"
#include "pbitmap.h"
#include "directory.h"
#include "filehdr.h"
#include "filesys.h"
#include <string>
#include "synch.h"

// Sectors containing the file headers for the bitmap of free sectors,
// and the directory of files.  These file headers are placed in well-known
// sectors, so that they can be located on boot-up.
#define FreeMapSector           0
#define DirectorySector     1 // root

// Initial file sizes for the bitmap and directory; until the file system
// supports extensible files, the directory size sets the maximum number
// of files that can be loaded onto the disk.
#define FreeMapFileSize   (NumSectors / BitsInByte)
#define NumDirEntries           10
#define DirectoryFileSize   (sizeof(DirectoryEntry) * NumDirEntries)

static char *protectionName[] = { "- - x", "- w -", "- w x",
    "r - -", "r - x",
    "r w -", "r w x" };

//----------------------------------------------------------------------
// FileSystem::FileSystem
//      Initialize the file system.  If format = TRUE, the disk has
//      nothing on it, and we need to initialize the disk to contain
//      an empty directory, and a bitmap of free sectors (with almost but
//      not all of the sectors marked as free).
//
//      If format = FALSE, we just have to open the files
//      representing the bitmap and the directory.
//
//      "format" -- should we initialize the disk?
//----------------------------------------------------------------------

FileSystem::FileSystem(bool format)
```

```
{
    DEBUG(dbgFile, "Initializing the file system.");
    currentDirSector = DirectorySector;

    if (format) {
        PersistentBitmap *freeMap = new PersistentBitmap(NumSectors);
        Directory *directory = new Directory(NumDirEntries);
            FileHeader *mapHdr = new FileHeader;
            FileHeader *dirHdr = new FileHeader;

        DEBUG(dbgFile, "Formatting the file system.");

    // First, allocate space for FileHeaders for the directory and bitmap
    // (make sure no one else grabs these!)
            freeMap->Mark(FreeMapSector);
            freeMap->Mark(DirectorySector);

    // Second, allocate space for the data blocks containing the contents
    // of the directory and bitmap files.  There better be enough space!

            ASSERT(mapHdr->Allocate(freeMap, FreeMapFileSize));
            ASSERT(dirHdr->Allocate(freeMap, DirectoryFileSize));

  char *name = new char[4];

  dirHdr->setParentSector(1029);
  dirHdr->setDir(); // mark the dir flag
  dirHdr->setdirnum(NumDirEntries);

    // Flush the bitmap and directory FileHeaders back to disk
    // We need to do this before we can "Open" the file, since open
    // reads the file header off of disk (and currently the disk has garbage
    // on it!).

   DEBUG(dbgFile, "Writing headers back to disk.");
            mapHdr->WriteBack(FreeMapSector);
            dirHdr->WriteBack(DirectorySector);

    // OK to open the bitmap and directory files now
    // The file system operations assume these two files are left open
    // while Nachos is running.
  freeMapFile = new(nothrow) OpenFile(FreeMapSector);
  directoryFile = new(nothrow)  OpenFile(DirectorySector);

    // Once we have the files "open", we can write the initial version
    // of each file back to disk.  The directory at this point is completely
    // empty; but the bitmap has been changed to reflect the fact that
    // sectors on the disk have been allocated for the file headers and
    // to hold the file data for the directory and bitmap.

        DEBUG(dbgFile, "Writing bitmap and directory back to disk.");
            freeMap->WriteBack(freeMapFile);    // flush changes to disk

  directory->Add(".", DirectorySector); // add entry to point itself.
            directory->WriteBack(directoryFile);

            if (debug->IsEnabled('f')) {
               freeMap->Print();
               directory->Print();
        }
        delete freeMap;
            delete directory;
            delete mapHdr;
            delete dirHdr;
    } else {
    // if we are not formatting the disk, just open the files representing
    // the bitmap and directory; these are left open while Nachos is running
        freeMapFile = new OpenFile(FreeMapSector);
        directoryFile = new OpenFile(DirectorySector);
    }
}

//----------------------------------------------------------------------
// FileSystem::Create
//          Create a file in the Nachos file system (similar to UNIX create).
//          Since we can't increase the size of files dynamically, we have
//          to give Create the initial size of the file.
//
//          The steps to create a file are:
//            Make sure the file doesn't already exist
//       Allocate a sector for the file header
//            Allocate space on disk for the data blocks for the file
//            Add the name to the directory
//              Store the new file header on disk
//              Flush the changes to the bitmap and the directory back to disk
//
//      Return TRUE if everything goes ok, otherwise, return FALSE.
//
//      Create fails if:
//              file is already in directory
//              no free space for file header
//              no free entry for file in directory
//              no free space for data blocks for the file
//
//      Note that this implementation assumes there is no concurrent access
//      to the file system!
//
//      "name" -- name of file to be created
//      "initialSize" -- size of file to be created
//----------------------------------------------------------------------

bool
FileSystem::Create(char *name, int initialSize, int currentDirSector, int protection)
{
  if (name == NULL) {
    return FALSE;
  }

    Directory *directory;
    PersistentBitmap *freeMap;
    FileHeader *hdr;
    OpenFile *openFile;
    int sector;
    bool success;

    DEBUG(dbgFile, "Creating file " << name << " size " << initialSize);

    if (protection > 7 || protection < 1) {
      cout << "Fail, invalid protection bit.\n";
      return FALSE;
    }

    if (currentDirSector == -1) {
      currentDirSector = DirectorySector;
    }
    // check the protection bit of dir?
    int parentSector = parsePath(&name, currentDirSector);
    openFile = new OpenFile(parentSector);
    if (canWrite(openFile->getProtectionBit()) == FALSE) {
      cout << "Privilege Denied: dir - " << openFile->getProtectionBit() << "\n";
    }

    directory = new Directory(NumDirEntries);
    directory->FetchFrom(openFile);

    if (directory->Find(name) != -1) { // file is already in directory
      success = FALSE;
      cout << "The file - " << name << " exist at current dir - " << getFullName(currentDirSector) << "\n";
      directory->Print();
    }
    else {
        freeMap = new PersistentBitmap(freeMapFile,NumSectors);
        sector = freeMap->FindAndSetRange(2, 2 + NumInode);   // find a sector to hold the file header (sector 2 -
47)
          if (sector == -1)
        success = FALSE;              // no free block for file header
      else if (!directory->Add(name, sector)) {
        cout << "Fail, the current directory is full! \n";
        success = FALSE;      // no space in directory
      }
          else {
              hdr = new FileHeader;
              if (!hdr->Allocate(freeMap, initialSize))
          success = FALSE;            // no space on disk for data
              else {
                      success = TRUE;

        hdr->setParentSector(parentSector); // save parent sector
        hdr->setProtectionBit(protection); // set protection bit
                      // everthing worked, flush all changes back to disk
                      hdr->WriteBack(sector);
                      directory->WriteBack(openFile);
                      freeMap->WriteBack(freeMapFile);
              }
        delete hdr;
          }
        delete freeMap;
    }
    delete directory;
    delete openFile;
    return success;
}

//----------------------------------------------------------------------
// FileSystem::Open
//          Open a file for reading and writing.
//          To open a file:
//            Find the location of the file's header, using the directory
//            Bring the header into memory
//
//          "name" -- the text name of the file to be opened
//----------------------------------------------------------------------

OpenFile *
FileSystem::Open(char *name)
{
    Directory *directory = new Directory(NumDirEntries);
    OpenFile *openFile = NULL;
    int sector;

    DEBUG(dbgFile, "Opening file " << name);
    sector = parsePath(&name, currentDirSector); // accquire the reqiured dir sector

    if (sector == -1) {
      DEBUG(dbgFile, "cannot find sector " << sector);
    }

    openFile = new OpenFile(sector); // fetch dir file
    directory->FetchFrom(openFile);
    delete openFile;
    openFile = NULL;

    sector = directory->Find(name);
    if (sector > -1) { // find the file
      openFile = new OpenFile(sector);
    }

    delete directory;
```

```
   return openFile;

 // // single dir
 //   directory->FetchFrom(directoryFile);
 //   sector = directory->Find(name);
 //   if (sector >= 0)
          //openFile = new OpenFile(sector);   // name was found in directory
 //   delete directory;
 //   return openFile;                                      // return NULL if not found
}

//----------------------------------------------------------------------
// FileSystem::Remove
//          Delete a file from the file system.  This requires:
//              Remove it from the directory
//              Delete the space for its header
//              Delete the space for its data blocks
//              Write changes to directory, bitmap back to disk
//
//          Return TRUE if the file was deleted, FALSE if the file wasn't
//          in the file system.
//
//          "name" -- the text name of the file to be removed
//----------------------------------------------------------------------

bool
FileSystem::Remove(char *name) // make sure no process access to the file any more
{
   Directory *directory;
   PersistentBitmap *freeMap;
   FileHeader *fileHdr;
   int sector;

   DEBUG(dbgFile, "Removing file " << name);
   int dirSector = parsePath(&name, currentDirSector); // accquire the reqiured dir sector

   OpenFile *openFile = new OpenFile(dirSector); // fetch dir file
   directory = new Directory(NumDirEntries);
   directory->FetchFrom(openFile);
   sector = directory->Find(name);
   if (sector == -1) {
      delete directory;
      return FALSE;                         // file not found
   }
   fileHdr = new FileHeader;
   fileHdr->FetchFrom(sector);

   freeMap = new PersistentBitmap(freeMapFile,NumSectors);

   fileHdr->Deallocate(freeMap);           // remove data blocks
   freeMap->Clear(sector);                 // remove header block
   directory->Remove(name);

   freeMap->WriteBack(freeMapFile);                // flush to disk
   directory->WriteBack(openFile);     // flush to disk
   delete fileHdr;
   delete directory;
   delete freeMap;
   delete openFile;
   return TRUE;
}

//----------------------------------------------------------------------
// FileSystem::List
//          List all the files in the file system directory.
//----------------------------------------------------------------------

void
FileSystem::List()
{
   Directory *directory = new Directory(NumDirEntries);
   OpenFile *openFile = NULL;
   int sector;

   openFile = new OpenFile(currentDirSector);
   directory->FetchFrom(openFile);
   DEBUG(dbgFile, "Listing dir " << openFile->getFullName());
   DirectoryEntry* table = directory->getEntry();
   for (int i = 0; i < NumDirEntries; i++) {
      if (table[i].inUse) {
        FileHeader *hdr = new FileHeader();
        hdr->FetchFrom(table[i].sector);
        printf("%-50s%s\n", table[i].name, protectionName[hdr->getProtectionBit() - 1]);
        delete hdr;
      }
    }

    delete openFile;
    delete directory;
}

//----------------------------------------------------------------------
// FileSystem::Print
//          Print everything about the file system:
//            the contents of the bitmap
//            the contents of the directory
//            for each file in the directory,
//                the contents of the file header
//                the data in the file
//----------------------------------------------------------------------

void
FileSystem::Print()
{
    FileHeader *bitHdr = new FileHeader;
    FileHeader *dirHdr = new FileHeader;
    PersistentBitmap *freeMap = new PersistentBitmap(freeMapFile,NumSectors);
    Directory *directory = new Directory(NumDirEntries);

    printf("Bit map file header:\n");
    bitHdr->FetchFrom(FreeMapSector);
    //bitHdr->Print();

    printf("Directory file header:\n");
    dirHdr->FetchFrom(DirectorySector);
    //dirHdr->Print();

    printf("Current file header:\n");
    OpenFile *openFile = NULL;
    dirHdr->FetchFrom(currentDirSector);
    //dirHdr->Print();
    openFile = new OpenFile(currentDirSector);

    freeMap->Print();

    directory->FetchFrom(openFile);
    directory->Print();

    delete bitHdr;
    delete dirHdr;
    delete freeMap;
    delete directory;
    delete openFile;
}

int
FileSystem::ChangeDir(char **name, int currentDir){
  Directory *directory = new Directory(NumDirEntries);
  OpenFile *openFile = NULL;
  int sector;

  DEBUG(dbgFile, "Change to dir " << *name);
  sector = parsePath(name, currentDir); // accquire the reqiured dir sector
  if (sector == -1) { // cannot open
    cout << "Error path! \n";
    return -1;
  }

  openFile = new OpenFile(sector);
  directory->FetchFrom(openFile);
  sector = directory->Find(*name);
  if (sector == -1) {
    cout << "Error path! \n";
    return -1;
  }

  if (canRead(openFile->getProtectionBit()) == FALSE) {
    cout << "Privilege Denied: dir - " << openFile->getProtectionBit() << "\n";
    return -1;
  }

  FileHeader *hdr = new FileHeader();
  hdr->FetchFrom(sector); // target hdr
  if (hdr->isDir() == FALSE) {
    cout << "Not a dir! \n";
    return -1;
  }
  currentDirSector = sector;
  cout << "Now in dir: " << getFullName(sector) << "\n";
  *name = getFullName(sector);

  delete directory;
  delete openFile;
  delete hdr;
  return sector;
  // return current dir sector
}

bool
FileSystem::MakeDir(char **name, int currentDir) { // name cannot contains "/"
  Directory *directory = new Directory(NumDirEntries);
  OpenFile *openFile = NULL;
  FileHeader *dirHdr;
  bool success;
  int pSector;

  pSector = parsePath(name, currentDir);
  if (pSector < 0) {
    DEBUG(dbgFile, "cannot find path: " << *name);
    return FALSE;
  }

  openFile = new OpenFile(pSector); // fetch dir file
  directory->FetchFrom(openFile);

  if (canWrite(openFile->getProtectionBit()) == FALSE) {
    cout << "Privilege Denied: dir - " << openFile->getProtectionBit() << "\n";
    return -1;
  }

  if (directory->Find(*name) != -1) {
    cout << "Fail, the directory " << *name << " has already existed.\n";
    success = FALSE; // dir already in directory
  }
  else {
    PersistentBitmap *freeMap = new PersistentBitmap(freeMapFile, NumSectors);
    int hdrSector = freeMap->FindAndSetRange(2, 2 + NumInode);
    if (hdrSector == -1) {
      success = FALSE;
    }
    else if (directory->Add(*name, hdrSector) == FALSE) {
      cout << "Fail, the current directory is full!\n";
      success = FALSE;
    }
    else {
      dirHdr = new FileHeader();
      if (dirHdr->Allocate(freeMap, DirectoryFileSize) == FALSE) {
        success = FALSE;
      }
      else {
        success = TRUE;
        dirHdr->setProtectionBit(7); // default: r w x
        dirHdr->setParentSector(pSector); // record parent sector
        dirHdr->setDir(); // mark as dir
        dirHdr->setdirnum(NumDirEntries);
        dirHdr->WriteBack(hdrSector); // save hdr
        directory->WriteBack(openFile); // save parent dir

        freeMap->WriteBack(freeMapFile);// save free map;
        Directory *newDir = new Directory(NumDirEntries);
        OpenFile *newDirFile = new OpenFile(hdrSector);

        newDir->Add(".", hdrSector);
        newDir->Add("..", pSector);
        newDir->WriteBack(newDirFile);
        FileHeader *pHdr = new FileHeader();
        pHdr->FetchFrom(pSector);
        pHdr->setdirnum(directory->getSize());
        pHdr->WriteBack(pSector);
        delete pHdr;
        *name = newDirFile->getFullName();
        delete newDir;
        delete newDirFile;
      }
      delete dirHdr;
    }
    delete freeMap;
  }

  delete directory;
  delete openFile;
```

```
  return success;
  // add new dir to inode
}

int
FileSystem::parsePath(char **name, int sector) { // find the corresponding file, [out] name will hold the full name
  // treat as relative path
  sector = processPath(name, sector);
  if (sector == -1) {
    // treat as full path
    sector = processPath(name, DirectorySector);
  }
  return sector; // sector of dir
}

int FileSystem::processPath(char **name, int sector) {
  string path = string(*name), dir;
  size_t pos;
  char *filename;
  //cout << *name << "]name\n";
 // cout << path << "]path\n"; int count = 0;
  // if is relative path
  while ((pos = path.find("/")) != string::npos) {
    dir = path.substr(0, pos);
    path = path.substr(pos + 1, path.size());

   // cout << dir << "] dir - " << count++  <<" \n";
    //cout << path << "] path - " << count++ << " \n";

    if (dir == "") {
      dir = ".";
    }

    Directory *dirObj = new Directory(NumDirEntries);
    OpenFile *dirFile = new OpenFile(sector);
    dirObj->FetchFrom(dirFile);
    if (dirObj->Find((char*)dir.c_str()) != -1) { // can find the dir
      sector = dirObj->Find((char*)dir.c_str());  // get sector of the dir
      delete dirObj;
      delete dirFile;
    }
    else { // cannot find
```

```
      return -1;
    }
  }

  filename = new char[path.length() + 1];
  strcpy(filename, path.c_str());
  *name = filename; // get file name
  return sector; // sector of dir
}

void FileSystem::setCurrentDirSector(int sector) { // update by calling process
  currentDirSector = sector;
}

int
FileSystem::getCurrentDirSector() {
  return currentDirSector;
}

char*
FileSystem::getFullName(char *name, int sec) { // will change "name"
  int sector = parsePath(&name, sec);
  if (sector == -1) { // cannot open
    cout << "Error path! \n";
    return NULL;
  }

  OpenFile* openFile = new OpenFile(sector);
  Directory* directory = new Directory(NumDirEntries);
  directory->FetchFrom(openFile);
  sector = directory->Find(name);
  if (sector == -1) {
    cout << "Error filename! \n";
    return NULL;
  }
  char *result = getFullName(sector); // dir path
  delete openFile;
  delete directory;
  return result;
}

int
```

```
FileSystem::getProtectionBit(char *name, int sec) { // will not change name
  FileHeader *hdr = new FileHeader();
  Directory *dir = new Directory(NumDirEntries);
  OpenFile *parentFile;

  char* temp = new char[strlen(name) + 1];
  memcpy(temp, name, strlen(name) + 1);

  int sector = parsePath(&temp, sec);
  int targetSector;
  if (sector == -1) { // treat as root
    sector = DirectorySector;
  }

  parentFile = new OpenFile(sector);
  dir->FetchFrom(parentFile);
  targetSector = dir->Find(temp);

  if (targetSector == -1) {
    targetSector = DirectorySector; // treat as root
  }
  delete parentFile;

  hdr->FetchFrom(targetSector);
  int result = hdr->getProtectionBit(); // dir path
  delete hdr;
  delete dir;
  return result;
}

bool
FileSystem::canWrite(int bit) { // - w - (2), r w - (6), r w x (7), - w x (3)
  if (bit == 2 || bit == 6 || bit == 7 || bit == 3) {
    return TRUE;
  }
  return FALSE;
}

bool
FileSystem::canRead(int bit) { // r - - (4), r w - (6), r - x (5), r w x (7)
  if (bit == 4|| bit == 6 || bit == 5 || bit == 7) {
    return TRUE;
  }
  return FALSE;
}

bool
FileSystem::canExecute(int bit) { // - - x (1), r - x (5), - w x (3), r w x(7)
  if (bit == 1 || bit == 5 || bit == 3 || bit == 7) {
    return TRUE;
  }
  return FALSE;
}

char*
FileSystem::getParentDirectory(char* fullPath){
  string path = string(fullPath), dir;
  size_t pos;
  char *result;

  pos = path.find_last_of("/");
  if (pos  != string::npos) {
    dir = path.substr(0, pos);
  }

  result = new char[dir.length() + 1];
  strcpy(result, dir.c_str());
  return result;
}

bool
FileSystem::setProtectionBit(char* fullPath, int bit) {
  char* temp = new char[strlen(fullPath) + 1];
  memcpy(temp, fullPath, strlen(fullPath) + 1);

  if (!(bit <= 7 && bit >= 1)) {
    cout << "No Such protection bit!\n";
      return FALSE;
  }

  int sector = parsePath(&temp, currentDirSector);
  if (sector == -1) {
    cout << "No such path\n";
    return FALSE;
  }
  FileHeader *hdr = new FileHeader();
  OpenFile *openfile = new OpenFile(sector);
  Directory *directory = new Directory(NumDirEntries);
  directory->FetchFrom(openfile);
  int targetSector = directory->Find(temp);

  FileHeader *parentHdr = new FileHeader();
  parentHdr->FetchFrom(sector);
  if (canWrite(parentHdr->getProtectionBit()) == FALSE){
    cout << "Privilege Denied: dir - " << parentHdr->getProtectionBit() << "\n";
    return FALSE;
  }

  hdr->FetchFrom(targetSector);
  hdr->setProtectionBit(bit); // dir path
  hdr->WriteBack(targetSector);
  delete openfile;
  delete directory;
  delete parentHdr;
  delete hdr;
  return 1;
}

char*
FileSystem::getFullName(int sector) {
  string path = "";
  string name;
  char *filename = new char[100];

  if (sector < 1 || sector > 1029) {
    return "invalid";
  }

  while (sector != 1029) {
    FileHeader *currentHdr = new FileHeader();
    currentHdr->FetchFrom(sector);
    int parentSector = currentHdr->getParentSector();

    if (parentSector != 1029) { // not root
      OpenFile *parentFile = new OpenFile(parentSector);
      Directory* directory = new Directory(NumDirEntries);
      directory->FetchFrom(parentFile);
      char* temp  = directory->getName(sector);
      name = string(temp);
      if (path != "") {
        path = name + "/" + path;
      }
      else {
        path = name;
      }

      delete parentFile;
      delete directory;
    }
    else {
      // root dir
      ASSERT(sector == 1);
      path = "/" + path;
    }
    sector = parentSector;
    delete currentHdr;
  }

  strcpy(filename, path.c_str());
  return filename; // sector of dir
}

bool
FileSystem::isEmptyDir(char* path, int sector) { // whether a path is dir or not
  char* temp = new char[strlen(path) + 1];
  memcpy(temp, path, strlen(path) + 1);
  int dirSector = parsePath(&temp, sector);
  if (dirSector == -1) {
    cout << "No such path\n";
    return FALSE;
  }
  FileHeader *hdr = new FileHeader();
  OpenFile *openfile = new OpenFile(dirSector);
  Directory *directory = new Directory(NumDirEntries); // parent
  directory->FetchFrom(openfile);
  int targetSector = directory->Find(temp);
  Directory *targetDir = new Directory(NumDirEntries);
  OpenFile *targetFile = new OpenFile(targetSector);
  targetDir->FetchFrom(targetFile);
  if (targetDir->isEmpty() == FALSE) {
    cout << "Fail, not an empty directory\n";
    return FALSE;
  }

  hdr->FetchFrom(targetSector);
  bool isDir = hdr->isDir();
  if (isDir == FALSE) {
    cout << "Fail, not a dir.\n";
  }

  delete hdr;
  delete openfile;
  delete directory;
  return isDir;
}

int
FileSystem::moveDir(char* from, char *to) {
  // acquire sector
  // fetch dir
  // write to target dir
  // if success, delete current dir
  char* fromDir = new char[strlen(from) + 1];
  memcpy(fromDir, from, strlen(from) + 1);

  char* toDir = new char[strlen(to) + 1];
  memcpy(toDir, to, strlen(to) + 1);

  int fromPSector = parsePath(&fromDir, currentDirSector);
  int toPSector = parsePath(&toDir, currentDirSector);

  if (fromPSector == -1 || toPSector == -1) {
    cout << "No such path\n";
    return FALSE;
  }


  FileHeader *hdr = new FileHeader();
  OpenFile *fromOpenfile = new OpenFile(fromPSector);
  Directory *fromDirectory = new Directory(NumDirEntries);
  OpenFile *toOpenfile = new OpenFile(toPSector);
  Directory *toDirectory = new Directory(NumDirEntries);
  fromDirectory->FetchFrom(fromOpenfile);
  toDirectory->FetchFrom(toOpenfile);
  int targetSector = fromDirectory->Find(fromDir);
  if (targetSector == -1) {
    cout << "No such target file!\n";
    return FALSE;
  }

  hdr->FetchFrom(targetSector);

  // start check privilege //
  FileHeader *fromPHdr = new FileHeader();
  FileHeader *toPHdr = new FileHeader();
  fromPHdr->FetchFrom(fromPSector);
  toPHdr->FetchFrom(toPSector);
  if (canRead(fromPHdr->getProtectionBit()) == FALSE || canWrite(fromPHdr->getProtectionBit()) == FALSE) {
    cout << "Privilege Denied: file - " << from << " dir - " << fromPHdr->getProtectionBit() << "\n"; // should can
read and write from parent dir
    delete fromPHdr;
    delete toPHdr;
    return FALSE;
  }
  if (canWrite(toPHdr->getProtectionBit()) == FALSE) {
    cout << "Privilege Denied: file - " << to << " dir - " << toPHdr->getProtectionBit() << "\n";// should can write from
parent dir
    delete fromPHdr;
    delete toPHdr;
    return FALSE;
  }
  if (canRead(hdr->getProtectionBit()) == FALSE || canWrite(hdr->getProtectionBit()) == FALSE) {
    cout << "Privilege Denied: file - " << to << " dir - " << hdr->getProtectionBit() << "\n"; // should can read and
write source file
    delete fromPHdr;
    delete toPHdr;
    return FALSE;
  }

  delete fromPHdr;
  delete toPHdr;
  // end check privilege //
  hdr->setParentSector(toPSector); // update info
  hdr->WriteBack(targetSector);

  fromDirectory->Remove(fromDir); // change parent dir entry
  toDirectory->Add(toDir, targetSector);
  fromDirectory->WriteBack(fromOpenfile); // update
  toDirectory->WriteBack(toOpenfile);

  if (hdr->isDir() == TRUE) { // if the path is dir path
    OpenFile *openfile = new OpenFile(targetSector);
    Directory *dirFile = new Directory(NumDirEntries);
    dirFile->FetchFrom(openfile);

    if (dirFile->Find("..") != -1) { // update entry
      dirFile->Remove("..");
      dirFile->Add("..", toPSector);
      dirFile->WriteBack(openfile); // save
    }

    delete openfile;
    delete dirFile;
  }

  delete hdr;
  delete fromOpenfile;
  delete toOpenfile;
  delete fromDirectory;
  delete toDirectory;
  return TRUE;
}

int
FileSystem::copyFile(char* from, char *to) {

  int amountRead, fileLength;
  char *buffer;

  char* fromDir = new char[strlen(from) + 1];
  memcpy(fromDir, from, strlen(from) + 1);

  char* toDir = new char[strlen(to) + 1];
  memcpy(toDir, to, strlen(to) + 1);
  DEBUG(dbgFile, "From:" << from);
  DEBUG(dbgFile, "To:" << to);

  int fromPSector = parsePath(&fromDir, currentDirSector);
  int toPSector = parsePath(&toDir, currentDirSector);
  DEBUG(dbgFile, "From:" << fromPSector);
  DEBUG(dbgFile, "To:" << toPSector);

  if (fromPSector == -1 || toPSector == -1) {
    cout << "No such path\n";
    return FALSE;
  }

  FileHeader *hdr = new FileHeader();
  OpenFile *fromOpenfile = new OpenFile(fromPSector);
  Directory *fromDirectory = new Directory(NumDirEntries);
  OpenFile *toOpenfile = new OpenFile(toPSector);
  Directory *toDirectory = new Directory(NumDirEntries);
  fromDirectory->FetchFrom(fromOpenfile);
  int targetSector = fromDirectory->Find(fromDir);
  if (targetSector == -1) {
    cout << "No such target file!\n";
    return FALSE;
  }

  hdr->FetchFrom(targetSector);

  // start check privilege //
  FileHeader *fromPHdr = new FileHeader();
  FileHeader *toPHdr = new FileHeader();
  fromPHdr->FetchFrom(fromPSector);
  toPHdr->FetchFrom(toPSector);
  if (canRead(fromPHdr->getProtectionBit()) == FALSE || canWrite(fromPHdr->getProtectionBit()) == FALSE) {
     cout << "Privilege Denied: file - " << from << " dir - " << fromPHdr->getProtectionBit() << "\n"; // should can
read and write from parent dir
    delete fromPHdr;
    delete toPHdr;
    return FALSE;
  }
  if (canWrite(toPHdr->getProtectionBit()) == FALSE) {
    cout << "Privilege Denied: file - " << to << " dir - " << toPHdr->getProtectionBit() << "\n";// should can write from
parent dir
```

```
    delete fromPHdr;
    delete toPHdr;
    return FALSE;
  }
  if (canRead(hdr->getProtectionBit()) == FALSE || canWrite(hdr->getProtectionBit()) == FALSE) {
     cout << "Privilege Denied: file - " << to << " dir - " << hdr->getProtectionBit() << "\n"; // should can read and
write source file
    delete fromPHdr;
    delete toPHdr;
    return FALSE;
  }

  delete fromPHdr;
  delete toPHdr;
  // end check privilege //

  //read data
  OpenFile *fromFile = new OpenFile(targetSector);

  // Create a Nachos file of the same length
  fileLength = hdr->FileLength();
  DEBUG('f', "Copying file " << from << " of size " << fileLength << " to file " << to);
  if (!kernel->fileSystem->Create(to, fileLength, toPSector, 7)) {   // Create Nachos file
    printf("Copy: couldn't create output file %s\n", to);
    return FALSE;
  }

  OpenFile *openFile;
  openFile = kernel->fileSystem->Open(to);
  ASSERT(openFile != NULL);

  buffer = new char[fileLength + 1];
  fromFile->Read(buffer, fileLength);
  openFile->Write(buffer, fileLength);
  delete buffer;

  // Close the Nachos files
  delete openFile;
  delete fromFile;
  delete hdr;
  delete fromOpenfile;
  delete fromDirectory;
  delete toOpenfile;

  delete toDirectory;

  cout << "The -cp operation is completed.\n";

}


#endif // FILESYS_STUB
```

filehdr.h

```
#include "copyright.h"

#ifndef FILEHDR_H
#define FILEHDR_H

#include "disk.h"
#include "pbitmap.h"

#define NumDirect           ((SectorSize - 8 * sizeof(int)) / sizeof(int)) // 22?
//#define MaxFileSize       (NumDirect * SectorSize) // 4 kb for now
#define MaxFileSize (SectorSize * SectorsPerTrack * NumTracks -  (NumInode + 2) * SectorSize);

#define NumInode 96
#define NumInDirect         ((SectorSize - 2 * sizeof(int)) / sizeof(int))
#define MaxBlockSize        (NumInDirect * SectorSize)

class FileBlock {
public:
  FileBlock(/*int index*/);
  ~FileBlock();
  int Allocate(PersistentBitmap *bitMap, int fileSize);// Initialize a file header,
        //  including allocating space
        //  on disk for the file data
  void Deallocate(PersistentBitmap *bitMap);  // De-allocate this file's
        //  data blocks

  void FetchFrom(int sectorNumber);         // Initialize file header from disk
  void WriteBack(int sectorNumber);         // Write modifications to file header
      //  back to disk

  int ByteToSector(int offset);         // Convert a byte offset into the file
  //      // to the disk sector containing
  //      // the byte

  void Print();                         // Print the contents of the file.
  void setNextBlock(int block);
  int FileLength();
  int getNextBlock();
  int Expand(PersistentBitmap *freeMap, int expendedSize);

private:
  //int index;
  int dataSectors[NumInDirect];             // Disk sector numbers for each data  //sector[30], making file max
size = 4kb
      // block in the file
  int nextBlock;
  int numBytes;
};

// The following class defines the Nachos "file header" (in UNIX terms,
// the "i-node"), describing where on disk to find all of the data in the file.
// The file header is organized as a simple table of pointers to
// data blocks.
```

```
//
// The file header data structure can be stored in memory or on disk.
// When it is on disk, it is stored in a single sector -- this means
// that we assume the size of this data structure to be the same
// as one disk sector.  Without indirect addressing, this
// limits the maximum file length to just under 4K bytes.
//
// There is no constructor; rather the file header can be initialized
// by allocating blocks for the file (if it is a new file), or by
// reading it from disk.

class FileHeader {
  public:
    FileHeader();
    bool Allocate(PersistentBitmap *bitMap, int fileSize);// Initialize a file header,
                                                          //  including allocating space
                                                          //  on disk for the file data
    void Deallocate(PersistentBitmap *bitMap);  // De-allocate this file's
                                                          //  data blocks

   // int Expend(PersistentBitmap *bitMap, int expendedSize);

    void FetchFrom(int sectorNumber);           // Initialize file header from disk
    void WriteBack(int sectorNumber);           // Write modifications to file header
                                                //  back to disk

    int ByteToSector(int offset);        // Convert a byte offset into the file
                                                // to the disk sector containing
                                                // the byte

    int FileLength();                           // Return the length of the file
                                                // in bytes

    void Print();                        // Print the contents of the file.

    void setProtectionBit(int protection);
    int getProtectionBit();
    void setParentSector(int sector);
    int getParentSector();
    bool isDir();
    void setDir();
    int getdirnum();
    void setdirnum(int n);

  private:
    int numBytes;                        // Number of bytes in the file
    int numSectors;                             // Number of data sectors in the file
    //int dataSectors[NumDirect];               // Disk sector numbers for each data  //sector[30], making file max
size = 4kb
                                                // block in the file
    int protectionBit; // protection bit
    int firstBlock;
    int lastBlock; // used for speedup expend
    bool dirFlag;
    int parentSector; // speedup searching
    int dataSectors[NumDirect];
    int dirnum;
};

#endif // FILEHDR_H
```

filehdr.cc

```
#include "copyright.h"

#include "filehdr.h"
#include "debug.h"
#include "synchdisk.h"
#include "main.h"


int
FileBlock::Allocate(PersistentBitmap *freeMap, int fileSize)
{
  DEBUG(dbgFile, " - Allocate at file block, size: " << fileSize);
  numBytes = fileSize;
  int numSectors = divRoundUp(fileSize, SectorSize);
  if (freeMap->NumClearRange(NumInode + 2, NumSectors) < numSectors) {
    DEBUG(dbgFile, " - Not enough space: " << fileSize);
    return -1;                    // not enough space
  }

  for (int i = 0; i < numSectors; i++) {
    dataSectors[i] = freeMap->FindAndSetRange(NumInode + 2, NumSectors);
    DEBUG(dbgFile, " - Allocated sector: " << dataSectors[i]);
    // since we checked that there was enough free space,
    // we expect this to succeed
    ASSERT(dataSectors[i] >= 0);
  }

  return fileSize;
}

void
FileBlock::Deallocate(PersistentBitmap *freeMap)
{
  int numSectors = divRoundUp(numBytes, SectorSize);
  for (int i = 0; i < numSectors; i++) {
    ASSERT(freeMap->Test((int)dataSectors[i]));  // ought to be marked!
    freeMap->Clear((int)dataSectors[i]);
  }
}
```

```
void
FileBlock::FetchFrom(int sector)
{
  kernel->synchDisk->ReadSector(sector, (char *)this);
}

void
FileBlock::WriteBack(int sector)
{
  kernel->synchDisk->WriteSector(sector, (char *)this);
}

int
FileBlock::ByteToSector(int offset)
{
  return(dataSectors[offset / SectorSize]);
}


int
FileBlock::FileLength()
{
  return numBytes;
}

FileBlock::FileBlock(/*int i*/) {
  for (int i = 0; i < NumInDirect; ++i) {
    dataSectors[i] = -1;
  }
  //index = i;
  nextBlock = NULL;
}

FileBlock::~FileBlock() { //gc
}

void
FileBlock::setNextBlock(int block) {
  nextBlock = block;
}
```

```
int
FileBlock::getNextBlock() {
  return nextBlock;
}

int
FileBlock::Expand(PersistentBitmap *freeMap, int expendedSize) {
  int requiredByte = expendedSize;
  int remain = 0;
  int index = numBytes / SectorSize; // when numBytes % SectorSize == 0, full or empty?
  if (numBytes % SectorSize == 0 && dataSectors[index] != -1) { // full
    remain = 0;
  }
  else {
    remain = SectorSize - (numBytes % SectorSize);
  }

  requiredByte = requiredByte - remain;

  if (remain == SectorSize) { // block not assigned yet
    if (freeMap->NumClearRange(NumInode + 2, NumSectors) < 1) {
      DEBUG(dbgFile, " - Not enough space: " << 1);
      return -1;                    // not enough space
    }

    dataSectors[index] = freeMap->FindAndSetRange(NumInode + 2, NumSectors);
  }

  DEBUG(dbgFile, " - Expand at file block, size: " << expendedSize);
  DEBUG(dbgFile, " - remain: " << remain);

  int requiredSectors = divRoundUp(requiredByte, SectorSize);
  int allocatedSectors = 0;
  int allocatedSize = remain;
  if (freeMap->NumClearRange(NumInode + 2, NumSectors) < requiredSectors)
    return -1;                      // not enough space

  for (int i = index + 1; i < NumInDirect; i++) {
    if (allocatedSize >= requiredByte) {
      break;
    }
    if (dataSectors[i] == -1) { // empty
```

```
      dataSectors[i] = freeMap->FindAndSetRange(NumInode + 2, NumSectors);
      allocatedSectors++;
      allocatedSize = allocatedSize + SectorSize;
      DEBUG(dbgFile, " - Expand at file block, size: " << allocatedSize);
      DEBUG(dbgFile, "   Expand at file block, sector: " << dataSectors[i]);
    }
  }

  numBytes = numBytes + allocatedSize;
  ASSERT(numBytes <= NumInDirect * SectorSize);
  return allocatedSize;
}

//----------------------------------------------------------------------
// FileHeader::Allocate
// 	Initialize a fresh file header for a newly created file.
//	Allocate data blocks for the file out of the map of free disk blocks.
//	Return FALSE if there are not enough free blocks to accomodate
//	the new file.
//
//	"freeMap" is the bit map of free disk sectors
//	"fileSize" is the size of file to allocate
//----------------------------------------------------------------------

bool
FileHeader::Allocate(PersistentBitmap *freeMap, int fileSize)
{
    DEBUG(dbgFile, "Allocating, size: " << fileSize);
    int requiredSize = fileSize;
    int reqSector  = divRoundUp(fileSize, SectorSize);
    int numBlocks = divRoundUp(reqSector, NumInDirect); // calculate the indirect block required to allocate

    if (fileSize == 0) {
      firstBlock = -1;
      lastBlock = -1;
      return TRUE;
    }

    if (freeMap->NumClearRange(NumInode + 2, NumSectors) - numBlocks < numSectors)
	return FALSE;		// not enough space

    FileBlock *newBlock;
  FileBlock *preBlock = NULL; //preBlock
  int allocateSize = MaxBlockSize;
  int allocatedSize = 0;
  int preSector = -1;
  int i = 0;
   DEBUG(dbgFile, "[First sector] " << firstBlock << "[Last sector] " << lastBlock << "[Previous numbyte] " <<
numBytes << "[Required]: " << requiredSize << "[Allocated]: " << allocatedSize);

  if (lastBlock != -1) { // has data before, link the old
    preSector = lastBlock;
    newBlock = new FileBlock();
    newBlock->FetchFrom(lastBlock); // fetch last block.

    allocateSize = MaxBlockSize - newBlock->FileLength(); // assgining remaining space
    allocatedSize = newBlock->Expand(freeMap, allocateSize);
    requiredSize = requiredSize - allocatedSize; // update require size
    newBlock->WriteBack(lastBlock);
    preBlock = newBlock;
  }

  while(requiredSize > allocatedSize) {

    //print info , why 10240 fail?
    DEBUG(dbgFile, "[required]: " << requiredSize);
    DEBUG(dbgFile, "[allocated]: " << allocatedSize);

    newBlock = new FileBlock(); // new block
    int allocatedSector = freeMap->FindAndSetRange(NumInode + 2, NumSectors);
    DEBUG(dbgFile, "Allocating sector: " << allocatedSector);
    if ((requiredSize - allocatedSize) >= MaxBlockSize) {
      allocateSize = MaxBlockSize;
    }
    else {
      allocateSize = requiredSize - allocatedSize;
    }

    DEBUG(dbgFile, "[allocated after]: " << allocateSize);

    if (newBlock->Allocate(freeMap, allocateSize) == -1) {
      DEBUG(dbgFile, "Allocating fail");
      return FALSE; // allocation fail
    }
```

```
      if (firstBlock == -1) { // no data before
        firstBlock = allocatedSector;
      }
      else {
        preBlock->setNextBlock(allocatedSector); // link previous block to current block
        preBlock->WriteBack(preSector);
      }
      newBlock->WriteBack(allocatedSector); // flush the disk at the allocated sector
      allocatedSize = allocatedSize + allocateSize;

      if (preBlock != NULL) {
        delete preBlock; // gc
      }

      preBlock = newBlock;
      preSector = allocatedSector;
      if (allocatedSize >= requiredSize) {
        lastBlock = allocatedSector;
        delete newBlock;
      }
      i++;
    }

    numBytes = numBytes + allocatedSize;
    numSectors = numSectors + numBlocks; // update new blocks
    return TRUE;
}

//----------------------------------------------------------------------
// FileHeader::Deallocate
//          De-allocate all the space allocated for data blocks for this file.
//
//          "freeMap" is the bit map of free disk sectors
//----------------------------------------------------------------------

void
FileHeader::Deallocate(PersistentBitmap *freeMap)
{
  DEBUG(dbgFile, "Deallocating..");
  int numBlocks = divRoundUp(numSectors, NumInDirect);
  int blockToDeall = firstBlock;
  FileBlock *block;
  for (int i = 0; i < numBlocks; i++) {
    block = new FileBlock();
    block->FetchFrom(blockToDeall); // fetch the block back
    block->Deallocate(freeMap);
    ASSERT(freeMap->Test(blockToDeall));
    freeMap->Clear(blockToDeall);
    blockToDeall = block->getNextBlock(); // fetch next block
    delete block;
  }
  firstBlock = -1;
  lastBlock = -1;
}

//----------------------------------------------------------------------
// FileHeader::FetchFrom
//          Fetch contents of file header from disk.
//
//          "sector" is the disk sector containing the file header
//----------------------------------------------------------------------

void
FileHeader::FetchFrom(int sector)
{
    kernel->synchDisk->ReadSector(sector, (char *)this);
}

//----------------------------------------------------------------------
// FileHeader::WriteBack
//          Write the modified contents of the file header back to disk.
//
//          "sector" is the disk sector to contain the file header
//----------------------------------------------------------------------

void
FileHeader::WriteBack(int sector)
{
    kernel->synchDisk->WriteSector(sector, (char *)this);
}

//----------------------------------------------------------------------
// FileHeader::ByteToSector
//          Return which disk sector is storing a particular byte within the file.
//      This is essentially a translation from a virtual address (the
//            offset in the file) to a physical address (the sector where the
//            data at the offset is stored).
//
//            "offset" is the location within the file of the byte in question
//----------------------------------------------------------------------

int
FileHeader::ByteToSector(int offset)
{
  int index = offset / MaxBlockSize;
  int blocknum = (offset % MaxBlockSize) / SectorSize;
  FileBlock *block;
  int blockFetched = firstBlock;

  for (int i = 0; i < numSectors; i++) {
    block = new FileBlock();
    block->FetchFrom(blockFetched);
    if (i == index) { // corresponding FileBlock
      int result = block->ByteToSector(offset % MaxBlockSize);
      delete block;

      DEBUG(dbgFile, " => index:" << index << " is at block no. " << blocknum);
      DEBUG(dbgFile, " ** The offset:" << offset << " is at block " << result);
      return result;
    }
    blockFetched = block->getNextBlock(); // get next block sector number
    delete block;
  }
}

//----------------------------------------------------------------------
// FileHeader::FileLength
//            Return the number of bytes in the file.
//----------------------------------------------------------------------

int
FileHeader::FileLength()
{
    return numBytes;
}
```

```
//----------------------------------------------------------------------
// FileHeader::Print
//          Print the contents of the file header, and the contents of all
//          the data blocks pointed to by the file header.
//----------------------------------------------------------------------

FileHeader::FileHeader() {
  //for (int i = 0; i < NumDirect; ++i) {
  //  dataSectors[i] = -1;
  //}
  numBytes = 0;
  numSectors = 0; // initial file size to 0
  protectionBit = 7; // initial to r w x
  for (int i = 0; i < NumDirect; i++) {
    dataSectors[i] = -1;
  }
  parentSector = 1;
  dirFlag = FALSE;
  lastBlock = -1;
  firstBlock = -1;

}

void
FileHeader::setProtectionBit(int protection) {
  protectionBit = protection;
}

int
FileHeader::getProtectionBit() {
  return protectionBit;
}



void
FileHeader::setParentSector(int sector) {
  parentSector = sector;
}

int
FileHeader::getParentSector() {
  return parentSector;
  }

bool
FileHeader::isDir(){
  return dirFlag;
}

void
FileHeader::setDir() {
  dirFlag = TRUE;
}

int
FileHeader::getdirnum() {
  return dirnum;
}

void
FileHeader::setdirnum(int n) {
  dirnum = n;
}
```

<u>openfile.h</u>

```
class OpenFile {
  public:
    OpenFile(int sector);                   // Open a file whose header is located
                                                    // at "sector" on the disk
    ~OpenFile();                            // Close the file

    void Seek(int position);                // Set the position from which to
                                                    // start reading/writing -- UNIX lseek

    int Read(char *into, int numBytes); // Read/write bytes from the file,
                                                    // starting at the implicit position.
                                                    // Return the # actually read/written,
                                                    // and increment position in file.
    int Write(char *from, int numBytes);

    int ReadAt(char *into, int numBytes, int position);
                                                    // Read/write bytes from the file,
                                                    // bypassing the implicit position.
    int WriteAt(char *from, int numBytes, int position);

    int Length();                           // Return the number of bytes in the
                                                    // file (this interface is simpler
                                                    // than the UNIX idiom -- lseek to
                                                    // end of file, tell, lseek back

    char* getFullName();

    int getOpenfileId();

    void setOpenfileId(int id);

    int getProtectionBit();

    void enterReadRegion();

    void leaveReadRegion();

    void enterWriteRegion();

    void leaveWriteRegion();

    void setMode(int m);

    int getMode();

    int getSize();

  private:
    FileHeader *hdr;                                // Header for this file
    int seekPosition;                               // Current position within the file
    char* path; // full path
    int hdrSector;
    int openfileId;
    bool release;
    int mode;
};

#endif // FILESYS

#endif // OPENFILE_H
```

openfile.cc

```
OpenFile::OpenFile(int sector)
{
    hdr = new FileHeader();
    hdr->FetchFrom(sector);
    seekPosition = 0;
    hdrSector = sector;
    path = kernel->fileSystem->getFullName(sector);
}

//----------------------------------------------------------------------
// OpenFile::~OpenFile
//          Close a Nachos file, de-allocating any in-memory data structures.
//----------------------------------------------------------------------

OpenFile::~OpenFile()
{
  delete hdr;
}

//----------------------------------------------------------------------
// OpenFile::Seek
//          Change the current location within the open file -- the point at
//          which the next Read or Write will start from.
//
//          "position" -- the location within the file for the next Read/Write
//----------------------------------------------------------------------

void
OpenFile::Seek(int position)
{
    seekPosition = position;
}

//----------------------------------------------------------------------
// OpenFile::Read/Write
//          Read/write a portion of a file, starting from seekPosition.
//          Return the number of bytes actually written or read, and as a
//          side effect, increment the current position within the file.
//
//          Implemented using the more primitive ReadAt/WriteAt.
//
//          "into" -- the buffer to contain the data to be read from disk
//          "from" -- the buffer containing the data to be written to disk
//          "numBytes" -- the number of bytes to transfer
//----------------------------------------------------------------------

int
OpenFile::Read(char *into, int numBytes)
{
   enterReadRegion();
   int result = ReadAt(into, numBytes, seekPosition);
   seekPosition += result;
   leaveReadRegion();
   return result;
}

int
OpenFile::Write(char *into, int numBytes)
{
  enterWriteRegion();

   int result = WriteAt(into, numBytes, seekPosition);
   seekPosition += result;
  leaveWriteRegion();
  return result;
}

//----------------------------------------------------------------------
// OpenFile::ReadAt/WriteAt
//          Read/write a portion of a file, starting at "position".
//          Return the number of bytes actually written or read, but has
//          no side effects (except that Write modifies the file, of course).
//
//          There is no guarantee the request starts or ends on an even disk sector
//          boundary; however the disk only knows how to read/write a whole disk
//          sector at a time.  Thus:
//
//          For ReadAt:
//             We read in all of the full or partial sectors that are part of the
//             request, but we only copy the part we are interested in.
//          For WriteAt:
//             We must first read in any sectors that will be partially written,
//             so that we don't overwrite the unmodified portion.  We then copy
//             in the data that will be modified, and write back all the full
//             or partial sectors that are part of the request.
//
//          "into" -- the buffer to contain the data to be read from disk
//          "from" -- the buffer containing the data to be written to disk
//          "numBytes" -- the number of bytes to transfer
//          "position" -- the offset within the file of the first byte to be
//                                  read/written
//----------------------------------------------------------------------

int
OpenFile::ReadAt(char *into, int numBytes, int position)
{
   //enterReadRegion();

   // -- enter critical region
   int fileLength = hdr->FileLength();
   int i, firstSector, lastSector, numSectors;
   char *buf;

   if ((numBytes <= 0) || (position >= fileLength)) {
     //leaveReadRegion();
     return 0;                                          // check request
   }
   if ((position + numBytes) > fileLength)
          numBytes = fileLength - position;
   DEBUG(dbgFile, "Reading " << numBytes << " bytes at " << position << " from file of length " << fileLength);

   firstSector = divRoundDown(position, SectorSize);
   lastSector = divRoundDown(position + numBytes - 1, SectorSize);
   numSectors = 1 + lastSector - firstSector;

   // read in all the full and partial sectors that we need
   buf = new char[numSectors * SectorSize];
   for (i = firstSector; i <= lastSector; i++)
       kernel->synchDisk->ReadSector(hdr->ByteToSector(i * SectorSize),
                                               &buf[(i - firstSector) * SectorSize]);

   // copy the part we want
   bcopy(&buf[position - (firstSector * SectorSize)], into, numBytes);
   //leaveReadRegion();
   //  -- leave critical regoin --

   delete [] buf;
   return numBytes;
```

```
}

int
OpenFile::WriteAt(char *from, int numBytes, int position)
{
   if (position + numBytes > hdr->FileLength()) {
   OpenFile *mapFile = new OpenFile(0);
   PersistentBitmap *freeMap = new PersistentBitmap(mapFile, NumSectors);
   hdr->Allocate(freeMap, position + numBytes - hdr->FileLength() +1);
   hdr->WriteBack(hdrSector);
   freeMap->WriteBack(mapFile);
   delete freeMap;
   delete mapFile;
   }

   //enterWriteRegion();
   // -- enter critical region
   int fileLength = hdr->FileLength();
   int i, firstSector, lastSector, numSectors;
   bool firstAligned, lastAligned;
   char *buf;


   if ((numBytes <= 0) || (position >= fileLength)) {
     //leaveWriteRegion();
     return 0;                                    // check request
   }
   if ((position + numBytes) > fileLength)
           numBytes = fileLength - position;
   DEBUG(dbgFile, "Writing " << numBytes << " bytes at " << position << " from file of length " << fileLength);

   firstSector = divRoundDown(position, SectorSize);
   lastSector = divRoundDown(position + numBytes - 1, SectorSize);
   numSectors = 1 + lastSector - firstSector;

   buf = new char[numSectors * SectorSize];

   firstAligned = (position == (firstSector * SectorSize));
   lastAligned = ((position + numBytes) == ((lastSector + 1) * SectorSize));

// read in first and last sector, if they are to be partially modified
   if (!firstAligned)
      ReadAt(buf, SectorSize, firstSector * SectorSize);
   if (!lastAligned && ((firstSector != lastSector) || firstAligned))
      ReadAt(&buf[(lastSector - firstSector) * SectorSize],
                                    SectorSize, lastSector * SectorSize);

// copy in the bytes we want to change
   bcopy(from, &buf[position - (firstSector * SectorSize)], numBytes);

// write modified sectors back
   for (i = firstSector; i <= lastSector; i++)
      kernel->synchDisk->WriteSector(hdr->ByteToSector(i * SectorSize),
                                          &buf[(i - firstSector) * SectorSize]);
   //leaveWriteRegion();
   // -- leave critical region --
   delete [] buf;
   return numBytes;
}

//----------------------------------------------------------------------
// OpenFile::Length
//         Return the number of bytes in the file.
//----------------------------------------------------------------------

int
```

```
OpenFile::Length()
{
   return hdr->FileLength();
}

char*
OpenFile::getFullName() {
  return path;
}

int
OpenFile::getOpenfileId() {
  return openfileId;
}

void
OpenFile::setOpenfileId(int id) {
  openfileId = id;
}

int
OpenFile::getProtectionBit() {
  return hdr->getProtectionBit();
}

void
OpenFile::enterReadRegion() {
  release = FALSE;
  string filename = string(path);

  if (kernel->locks->RCLocks->find(filename) == kernel->locks->RCLocks->end()) {
    return;
  }

   if ((*kernel->locks->WRLocks)[filename]->IsHeldByCurrentThread() == FALSE) { // if the lock not held by current
thread
    (*kernel->locks->RCLocks)[filename]->Acquire(); // down(RC)
    (*kernel->readCount)[filename]++;
    if ((*kernel->readCount)[filename] == 1) {
      (*kernel->locks->WRLocks)[filename]->Acquire(); // when there is a reader, acquire RW lock
      release = TRUE; // need to release
      DEBUG(dbgSynch, "Reading: RW Lock Aquired.");
    }
    (*kernel->locks->RCLocks)[filename]->Release(); // up(RC)
  }
}

void
OpenFile::leaveReadRegion() {
   if (release == TRUE) { // only when acquiring a lock at read region will release the lock (WriteAt() will also call
ReadAt())
    string filename = string(path);

    if (kernel->locks->RCLocks->find(filename) == kernel->locks->RCLocks->end()) {
      return;
    }

    if ((*kernel->locks->WRLocks)[filename]->IsHeldByCurrentThread() == TRUE) {
      (*kernel->locks->RCLocks)[filename]->Acquire(); // down(RC)
      (*kernel->readCount)[filename]--;
      if ((*kernel->readCount)[filename] == 0) {
        (*kernel->locks->WRLocks)[filename]->Release(); // no reader any more, release RW lock
        DEBUG(dbgSynch, "Reading: RW Lock Released.");

      }
      (*kernel->locks->RCLocks)[filename]->Release(); // up(RC)
    }
  }
}

void
OpenFile::enterWriteRegion() {
  string filename = string(path);

  if (kernel->locks->RCLocks->find(filename) == kernel->locks->RCLocks->end()) {
    return;
  }

   if ((*kernel->locks->WRLocks)[filename]->IsHeldByCurrentThread() == FALSE) { // if the lock not held by current
thread
    (*kernel->locks->WRLocks)[filename]->Acquire(); // acquire RC lock
    DEBUG(dbgSynch, "Writing: RW Lock Aquired.");
  }

}

void
OpenFile::leaveWriteRegion() {
  string filename = string(path);

  if (kernel->locks->RCLocks->find(filename) == kernel->locks->RCLocks->end()) {
    return;
  }

  if ((*kernel->locks->WRLocks)[filename]->IsHeldByCurrentThread() == TRUE) {
    (*kernel->locks->WRLocks)[filename]->Release(); // release RC lock
    DEBUG(dbgSynch, "Writing: RW Lock Released.");
  }
}

void
OpenFile::setMode(int m) {
  mode = m;
}

int
OpenFile::getMode() {
  return mode;
}

int
OpenFile::getSize() {
  return hdr->getdirnum();
}
#endif //FILESYS_STUB
```

kernel.h

```
class PostOfficeInput;
class PostOfficeOutput;
class SynchConsoleInput;
class SynchConsoleOutput;
class SynchDisk;

class Locks;

class Kernel {
  public:
    Kernel(int argc, char **argv);
                                // Interpret command line arguments
    ~Kernel();                  // deallocate the kernel

    void Initialize(int quantum, bool isRandom, bool usetlb);          // initialize the kernel -- separated
                                // from constructor because
                                // refers to "kernel" as a global

    void ThreadSelfTest();  // self test of threads and synchronization

    void ConsoleTest();     // interactive console self test

    void NetworkTest();     // interactive 2-machine network test

    int getQuantum();

// These are public for notational convenience; really,
// they're global variables used everywhere.

    Thread *currentThread;  // the thread holding the CPU
    Scheduler *scheduler;   // the ready list
    Interrupt *interrupt;   // interrupt status
    Statistics *stats;              // performance metrics
    Alarm *alarm;           // the software alarm clock
    Machine *machine;        // the simulated CPU
    SynchConsoleInput *synchConsoleIn;
    SynchConsoleOutput *synchConsoleOut;
    SynchDisk *synchDisk;
    FileSystem *fileSystem;
    PostOfficeInput *postOfficeIn;
    PostOfficeOutput *postOfficeOut;

    Locks *locks;
    //map<string, Lock*> *WRLocks; // locks for read write for each file
    //map<string, Lock*> *RCLocks;
    List<string> *pendingDeleteFiles; // files that is pending deleted
    map<string, int> *readCount; // record readers count for each file
    map<string, int> *openNumForWR; // record the current opened times of a file

    map<int, OpenFile*> *openFileTable;
    int openFileIdCount;
    OpenFile *stdinFile;
    OpenFile *stdoutFile;
    void insertFileToTable(OpenFile* file);
    void removeFileFromTable(int id);

    int hostName;           // machine identifier
    OpenFile *swapSpace;
    int swapCounter;
    Bitmap *freeMap;
    LRUCache *EntryCache;  // LRU cache
    List<TranslationEntry * > *FIFO;
    int ThreadId;
    map<int,Thread*> *ProcessTable;
    bool isRandom;
    bool useTLB;
    LRUCache *TLBCache; // TLB cache
    char *consoleIn;            // file to read console input from
    char *consoleOut;           // file to send console output to

  private:
    bool randomSlice;                          // enable pseudo-random time slicing
    bool debugUserProg;         // single step user program
    double reliability;         // likelihood messages are dropped
    int quantum;
#ifndef FILESYS_STUB
    bool formatFlag;            // format the disk if this is true
#endif
};
```

kernel.cc

```
…
void
Kernel::Initialize(int q, bool israndom, bool usetlb)
{
   quantum = q;
   isRandom = israndom; // set if random pick page
   useTLB = usetlb;
   // We didn't explicitly allocate the current thread we are running in.
   // But if it ever tries to give up the CPU, we better have a Thread
   // object to save its state.
   currentThread = new Thread("main");
   currentThread->setStatus(RUNNING);

   stats = new Statistics();                // collect statistics
   interrupt = new Interrupt;                        // start up interrupt handling
   scheduler = new Scheduler();      // initialize the ready queue
   alarm = new Alarm(randomSlice);// start up time slicing
   machine = new Machine(debugUserProg);
   synchConsoleIn = new SynchConsoleInput(consoleIn); // input from stdin
   synchConsoleOut = new SynchConsoleOutput(consoleOut); // output to stdout
   synchDisk = new SynchDisk();
   locks = new Locks();

   readCount = new map<string, int>(); // record readers count for each file
   openNumForWR = new map<string, int>(); // record writer + reader count for each file.


   pendingDeleteFiles = new List<string>(); // files that is pending to delete

   openFileTable = new map<int, OpenFile*>(); // system-wide openfile table
   openFileIdCount = 0;

#ifdef FILESYS_STUB
   fileSystem = new FileSystem();
#else
   fileSystem = new FileSystem(formatFlag);
#endif // FILESYS_STUB
   postOfficeIn = new PostOfficeInput(10);
   postOfficeOut = new PostOfficeOutput(reliability);

   if (formatFlag == TRUE) {

     bool success = TRUE;

      if (!fileSystem->Create("swapspace", /*1024 * 10*/ 0, fileSystem->getCurrentDirSector(), 7)) { // create a file to
simulate the disk
       success = FALSE;
     }

     if (success == FALSE) {
       cout << "Initialization fail. \n";
     }
     else {
       cout << "File: [" << "/swapspace" << "] created.\n";
       cout << "The FileSystem is fomatted successfully! \n";
       cout << "The main thread is sleeping.. \n";
     }
}
   insertFileToTable(NULL); // 0 stdin
   insertFileToTable(NULL); // 1 stdout

   swapSpace = fileSystem->Open("swapspace");
   cout << "File: [" << "/swapspace" << "] opened.\n";
```

```
    insertFileToTable(swapSpace);


    swapCounter = 0;
    // initialize data structure
    freeMap = new Bitmap(NumPhysPages);
    EntryCache = new LRUCache(NumPhysPages);
    TLBCache = new LRUCache(TLBSize);
    FIFO = new List<TranslationEntry *>();
    ThreadId = 1;
    interrupt->Enable();

}

//----------------------------------------------------------------------
// Kernel::~Kernel
//      Nachos is halting.  De-allocate global data structures.
//----------------------------------------------------------------------

Kernel::~Kernel()
{

    delete stats;
    delete interrupt;
    delete scheduler;
    delete alarm;
    delete machine;
    delete synchConsoleIn;
    delete synchConsoleOut;
    delete synchDisk;
    delete fileSystem;
    delete postOfficeIn;
    delete postOfficeOut;
    delete freeMap;
    delete EntryCache;
    delete FIFO;
    delete ProcessTable;
    delete TLBCache;
    delete pendingDeleteFiles;
    delete locks;
    delete pendingDeleteFiles;
    delete readCount;
    delete openFileTable;

    Exit(0);
}

//----------------------------------------------------------------------
// Kernel::ThreadSelfTest
//   Test threads, semaphores, synchlists
//----------------------------------------------------------------------

void
Kernel::ThreadSelfTest() {
   Semaphore *semaphore;
   SynchList<int> *synchList;

   LibSelfTest();              // test library routines
   currentThread->SelfTest();         // test thread switching

                               // test semaphore operation
   semaphore = new Semaphore("test", 0);
   semaphore->SelfTest();
   delete semaphore;

                               // test locks, condition variables
```

```
                                        // using synchronized lists
   synchList = new SynchList<int>;
   synchList->SelfTest(9);
   delete synchList;

}

//----------------------------------------------------------------------
// Kernel::ConsoleTest
//      Test the synchconsole
//----------------------------------------------------------------------

void
Kernel::ConsoleTest() {
    char ch;

    cout << "Testing the console device.\n"
        << "Typed characters will be echoed, until ^D is typed.\n"
        << "Note newlines are needed to flush input through UNIX.\n";
    cout.flush();

    do {
        ch = synchConsoleIn->GetChar();
        if(ch != EOF) synchConsoleOut->PutChar(ch);   // echo it!
    } while (ch != EOF);

    cout << "\n";

}

//----------------------------------------------------------------------
// Kernel::NetworkTest
//      Test whether the post office is working. On machines #0 and #1, do:
//
//      1. send a message to the other machine at mail box #0
//      2. wait for the other machine's message to arrive (in our mailbox #0)
//      3. send an acknowledgment for the other machine's message
//      4. wait for an acknowledgement from the other machine to our
//          original message
//
//  This test works best if each Nachos machine has its own window
//----------------------------------------------------------------------

void
Kernel::NetworkTest() {

    if (hostName == 0 || hostName == 1) {
        // if we're machine 1, send to 0 and vice versa
        int farHost = (hostName == 0 ? 1 : 0);
        PacketHeader outPktHdr, inPktHdr;
        MailHeader outMailHdr, inMailHdr;
        char *data = "Hello there!";
        char *ack = "Got it!";
        char buffer[MaxMailSize];

        // construct packet, mail header for original message
        // To: destination machine, mailbox 0
        // From: our machine, reply to: mailbox 1
        outPktHdr.to = farHost;
        outMailHdr.to = 0;
        outMailHdr.from = 1;
        outMailHdr.length = strlen(data) + 1;

        // Send the first message
        postOfficeOut->Send(outPktHdr, outMailHdr, data);
        // Wait for the first message from the other machine
        postOfficeIn->Receive(0, &inPktHdr, &inMailHdr, buffer);
        cout << "Got: " << buffer << " : from " << inPktHdr.from << ", box "
                                        << inMailHdr.from << "\n";
        cout.flush();

        // Send acknowledgement to the other machine (using "reply to" mailbox
        // in the message that just arrived
        outPktHdr.to = inPktHdr.from;
        outMailHdr.to = inMailHdr.from;
        outMailHdr.length = strlen(ack) + 1;
        postOfficeOut->Send(outPktHdr, outMailHdr, ack);

        // Wait for the ack from the other machine to the first message we sent
              postOfficeIn->Receive(1, &inPktHdr, &inMailHdr, buffer);
        cout << "Got: " << buffer << " : from " << inPktHdr.from << ", box "
                                        << inMailHdr.from << "\n";
        cout.flush();
    }

    // Then we're done!
}

int
Kernel::getQuantum() {
  return quantum;
}

void
Kernel::insertFileToTable(OpenFile* file) {
  if (file != NULL) {
    (*openFileTable)[openFileIdCount] = file;
    file->setOpenfileId(openFileIdCount);
  }
  openFileIdCount++;
}

void
Kernel::removeFileFromTable(int id) {
  delete (*openFileTable)[id]; // gc
  openFileTable->erase(id);
}
```

addrspace.h

```
…
    int currentDirSector;
    char* currentDir;
    map<string, int> *openFileTable;// per-process openfile table
    void removeById(int id);
    bool isExisted(int id);
    //bool isExisted(char* name);

  private:
    TranslationEntry *pageTable;       // Assume linear page table translation
…
```

addrspace.cc

```
AddrSpace::AddrSpace(const AddrSpace& copiedItem) { // copy constructor
  numPages = copiedItem.numPages;
  pageTable = new TranslationEntry[numPages];
  for (int i = 0; i < numPages; i++) {
      pageTable[i].virtualPage = copiedItem.pageTable[i].virtualPage;
      pageTable[i].physicalPage = copiedItem.pageTable[i].physicalPage;
      pageTable[i].valid = copiedItem.pageTable[i].valid;
      pageTable[i].use = copiedItem.pageTable[i].use;
      pageTable[i].dirty = copiedItem.pageTable[i].dirty;
      pageTable[i].readOnly = copiedItem.pageTable[i].readOnly;
  }
  currentDirSector = copiedItem.currentDirSector;
  currentDir = copiedItem.currentDir;

  openFileTable = new map<string, int>();
  map<string, int>::iterator iter;
  iter = (*copiedItem.openFileTable).begin();
  while (iter != (*copiedItem.openFileTable).end()) {
    (*openFileTable)[iter->first] = iter->second; // copy
    (*kernel->openNumForWR)[iter->first]++; // increase the count
    iter++;
  }
}

int
AddrSpace::getNumPage() {
  return numPages;
}


TranslationEntry*
AddrSpace::getPageTable() {
  return pageTable;
}

bool
AddrSpace::isExisted(int id) {
  map<string, int>::iterator iter;
  iter = openFileTable->begin();
  while (iter != openFileTable->end()) {
    if (iter->second == id) { //  find the pair
      return TRUE;
    }
    iter++;
  }
  return FALSE;
}

void
AddrSpace::removeById(int id) {
  map<string, int>::iterator iter;
  iter = openFileTable->begin();
  while (iter != openFileTable->end()) {
    if (iter->second == id) { //  find the pair
      openFileTable->erase(iter);
      break;
    }
    iter++;
  }
}
```

synchdisk.h

```
class Locks {
public:
  Locks();
  map<string, Lock*> *WRLocks; // locks for read write for each file
  map<string, Lock*> *RCLocks;
};
```

synchdisk.cc

```
Locks::Locks() {
  WRLocks = new map<string, Lock*>(); // locks for read write for each file
  RCLocks = new map<string, Lock*>();
}
```

shell.c

```
#include "syscall.h"

int
main()
{
    SpaceId newProc;
    OpenFileId input = CONSOLEINPUT;
    OpenFileId output = CONSOLEOUTPUT;
    char prompt[2], ch, buffer[60], prog[60];
    int i, j, z, lastIndex;

    prompt[0] = '>';
    prompt[1] = '>';

    while(1)
    {
            Write(prompt, 2, output);

            i = 0;

            do {

                Read(&buffer[i], 1, input);

            } while( buffer[i++] != '\n' );

            buffer[--i] = '\0';

  lastIndex = 0;
            if( i > 0 ) {
    for (j = 0; j < i; j++) {
      if (buffer[j] == ';') { // seperate the command

        for ( z = 0; z < j - lastIndex; z++) {
          prog[z] = buffer[z + lastIndex];
        }
        prog[z] = '\0';
        newProc = Exec(prog);
        Join(newProc);
        lastIndex = j + 1;
      }
    }

    for (z = 0; z < i; z++) {
      prog[z] = buffer[z + lastIndex];
    }
    prog[z] = '\0';
    newProc = Exec(prog);
    Join(newProc);
            }
    }
}
```

Open.c

```
#include "syscall.h"

int
main()
{
  int fd;
  char name[9] = "testOpen";
  char content[9] = "testOpen";
  char result[9];
  int i;

  fd = Open(name, 2); // wr
  Write(content, 8, fd);
  Seek(0, fd);
  Read(result, 8, fd);
  Close(fd);

  fd = Open(name, 3); // append
  Write(content, 8, fd);
  Read(result, 8, fd);
  Close(fd);

  fd = Open(name, 1); // ro
  Write(content, 8, fd);
  Seek(0, fd);
  Read(result, 16, fd);
  Close(fd);
}
```

create.c

```
#include "syscall.h"

int
main()
{
  int fd;
  char name[9] = "testOpen";

  Create(name, 7);
}
```

remove.c

```
#include "syscall.h"

int
main()
{
  int fd;
  char name[9] = "testOpen";
  char content[9] = "testOpen";
  char result[9];
  int i;

  fd = Open(name, 2); // wr
  Remove(name);
  Close(fd);
}
```

4. Testing

Command Line:

1. Build the nachos system:

   make depend

   make

2. Run and test:

   ./nachos -f - When using the file system at first time, the disk should be formatted.

   ./nachos -cp [file from Unix] [file to Nachos] - The file should be copied to Nachos disk for running as user program.

   ./nachos -x [Nachos user program] - Run the user program at Nachos

3. Use Shel Application:

   ./nachos -cp ../test/shell shell - Before running Shell application, the file shuold be copied to Nachos disk.

   The supported command line is listed as following:

   >> ls

   >> cd [dir name]

   >> pwd

   >> mkdir [dir name]

   >> cp [source file name] [target file name]

   >> mv [source file name] [target file name]

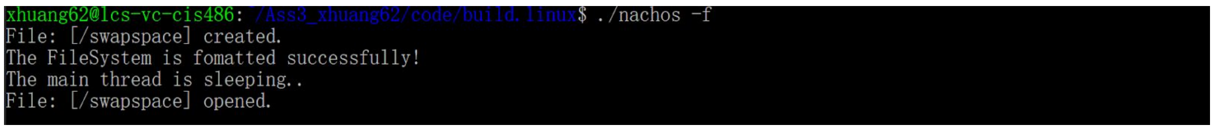
   >> rm [file name]

   >> rmdir [dir name]

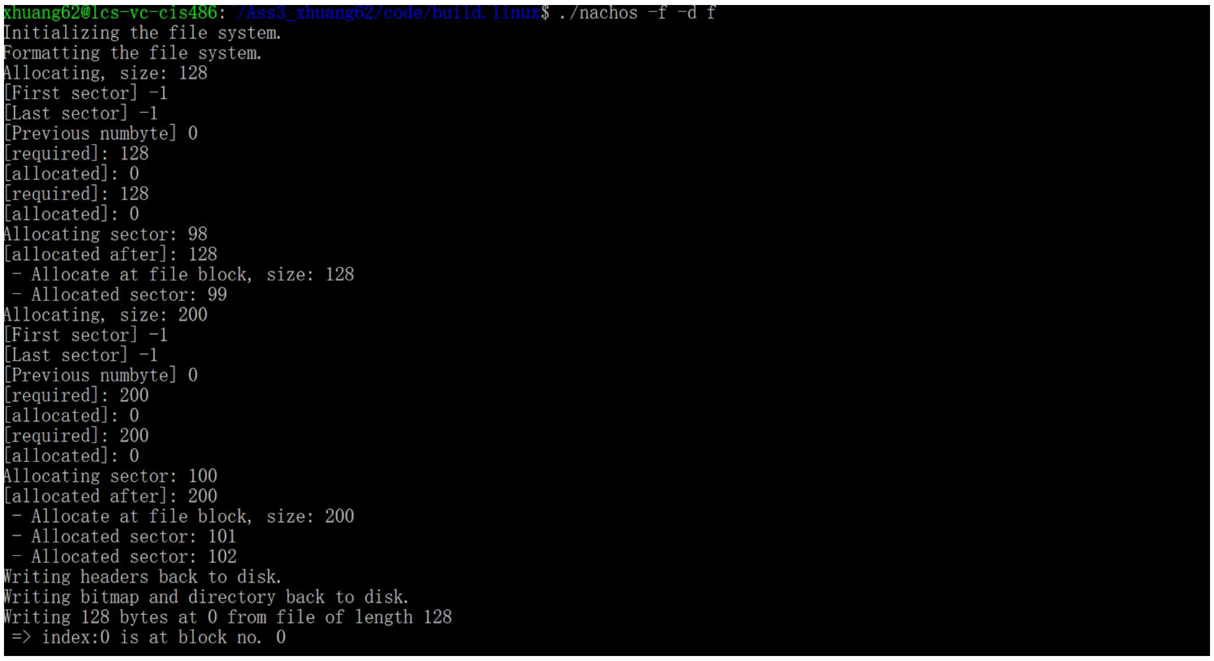
   >> chmod [mode(0-7)] [file name]

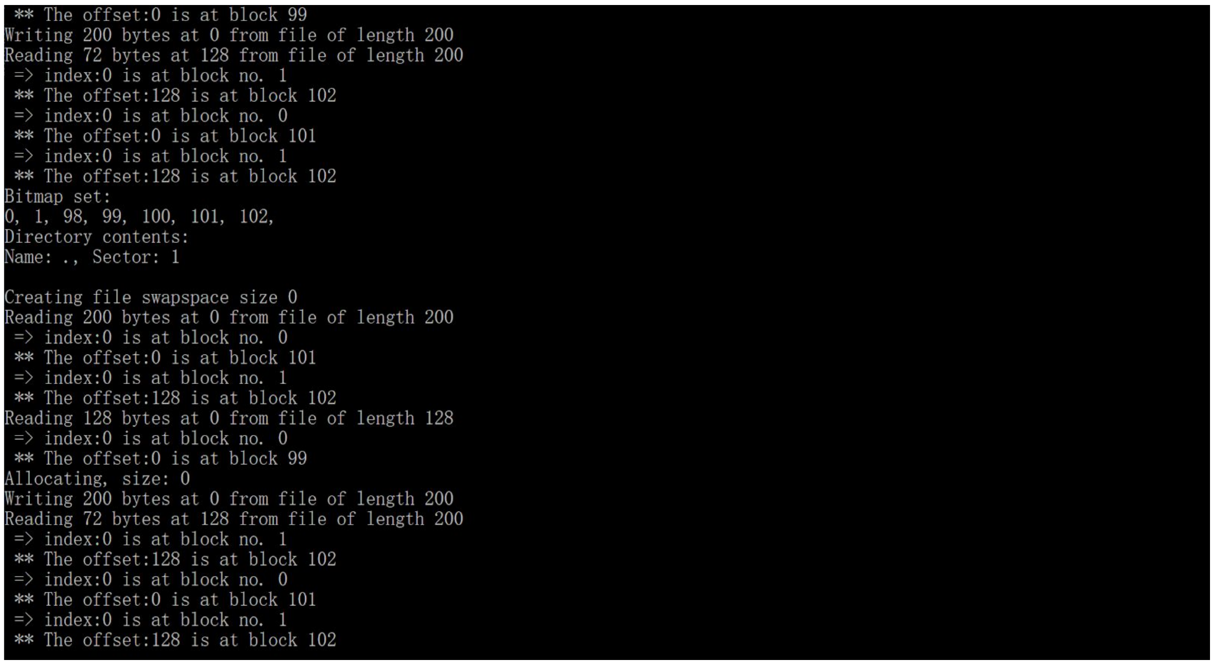
Note: Both relative path and absolute path will be supported.

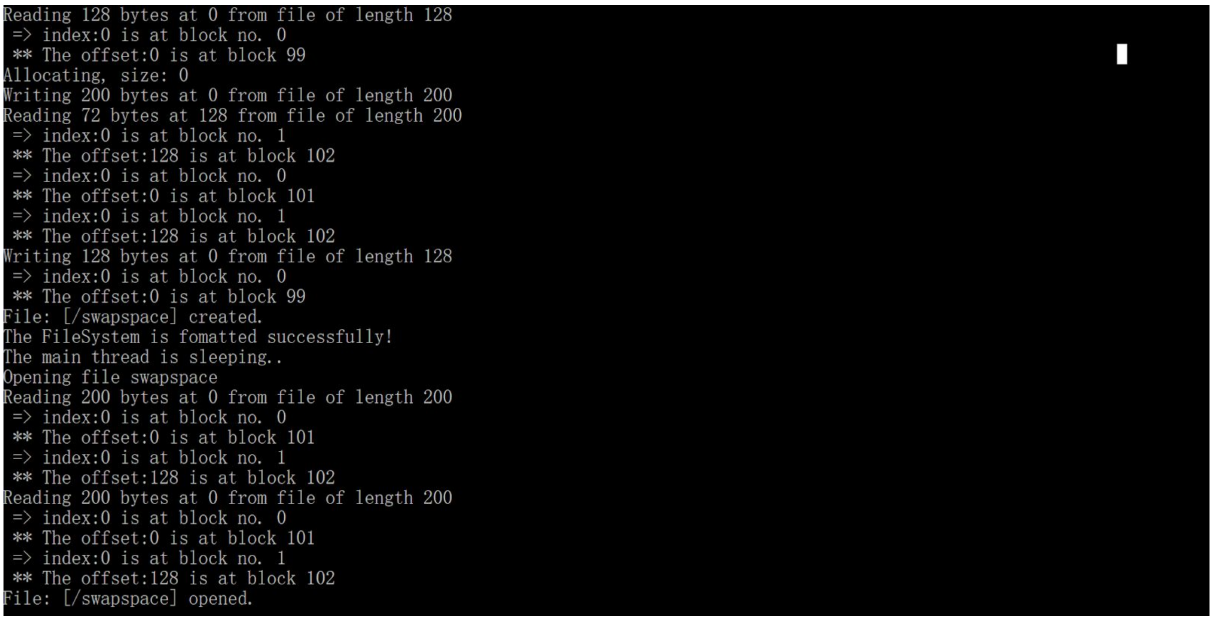
./nachos -f - When using the file system at first time, the disk should be formatted.


```
xhuang62@lcs-vc-cis486:~/Ass3_xhuang62/code/build.linux$ ./nachos -f
File: [/swapspace] created.
The FileSystem is fomatted successfully!
The main thread is sleeping..
File: [/swapspace] opened.
```

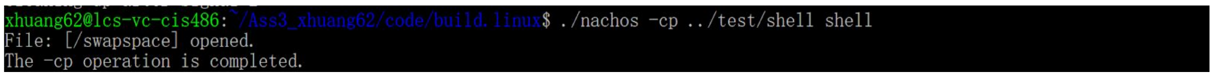
with debug information: **./nachos -f -d f**


```
xhuang62@lcs-vc-cis486:~/Ass3_xhuang62/code/build.linux$ ./nachos -f -d f
Initializing the file system.
Formatting the file system.
Allocating, size: 128
[First sector] -1
[Last sector] -1
[Previous numbyte] 0
[required]: 128
[allocated]: 0
[required]: 128
[allocated]: 0
Allocating sector: 98
[allocated after]: 128
 - Allocate at file block, size: 128
 - Allocated sector: 99
Allocating, size: 200
[First sector] -1
[Last sector] -1
[Previous numbyte] 0
[required]: 200
[allocated]: 0
[required]: 200
[allocated]: 0
Allocating sector: 100
[allocated after]: 200
 - Allocate at file block, size: 200
 - Allocated sector: 101
 - Allocated sector: 102
Writing headers back to disk.
Writing bitmap and directory back to disk.
Writing 128 bytes at 0 from file of length 128
 => index:0 is at block no. 0
```


```
 ** The offset:0 is at block 99
Writing 200 bytes at 0 from file of length 200
Reading 72 bytes at 128 from file of length 200
 => index:0 is at block no. 1
 ** The offset:128 is at block 102
 => index:0 is at block no. 0
 ** The offset:0 is at block 101
 => index:0 is at block no. 1
 ** The offset:128 is at block 102
Bitmap set:
0, 1, 98, 99, 100, 101, 102,
Directory contents:
Name: ., Sector: 1

Creating file swapspace size 0
Reading 200 bytes at 0 from file of length 200
 => index:0 is at block no. 0
 ** The offset:0 is at block 101
 => index:0 is at block no. 1
 ** The offset:128 is at block 102
Reading 128 bytes at 0 from file of length 128
 => index:0 is at block no. 0
 ** The offset:0 is at block 99
Allocating, size: 0
Writing 200 bytes at 0 from file of length 200
Reading 72 bytes at 128 from file of length 200
 => index:0 is at block no. 1
 ** The offset:128 is at block 102
 => index:0 is at block no. 0
 ** The offset:0 is at block 101
 => index:0 is at block no. 1
 ** The offset:128 is at block 102
```

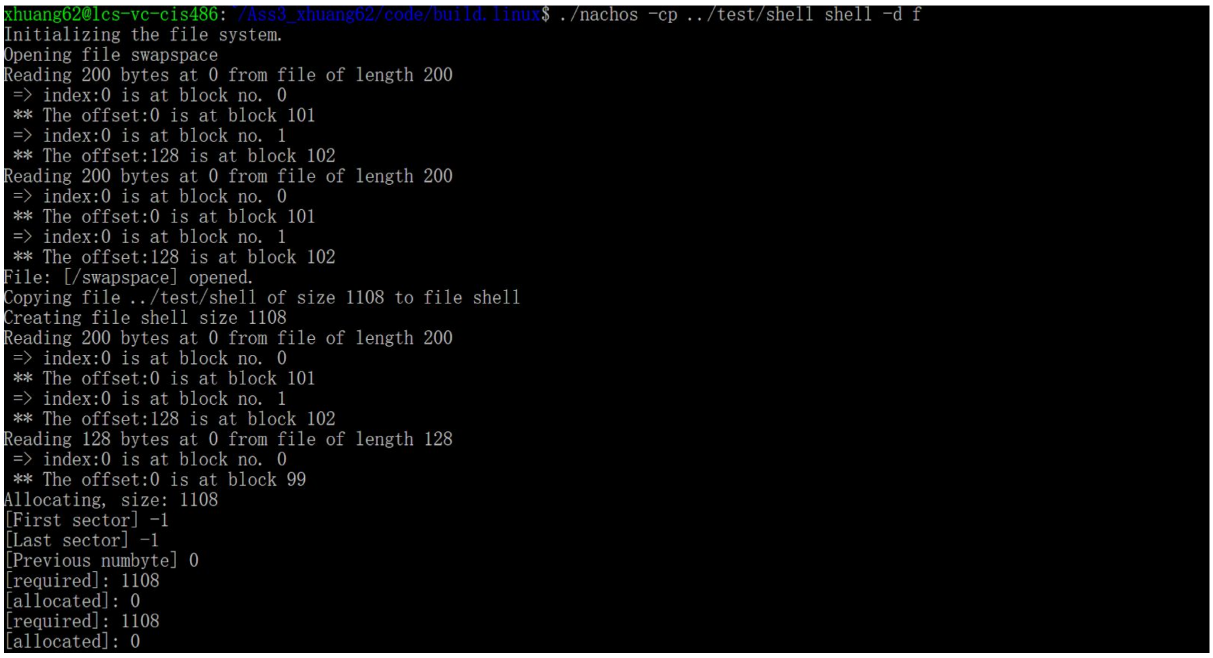

```
Reading 128 bytes at 0 from file of length 128
 => index:0 is at block no. 0
 ** The offset:0 is at block 99
Allocating, size: 0
Writing 200 bytes at 0 from file of length 200
Reading 72 bytes at 128 from file of length 200
 => index:0 is at block no. 1
 ** The offset:128 is at block 102
 => index:0 is at block no. 0
 ** The offset:0 is at block 101
 => index:0 is at block no. 1
 ** The offset:128 is at block 102
Writing 128 bytes at 0 from file of length 128
 => index:0 is at block no. 0
 ** The offset:0 is at block 99
File: [/swapspace] created.
The FileSystem is fomatted successfully!
The main thread is sleeping..
Opening file swapspace
Reading 200 bytes at 0 from file of length 200
 => index:0 is at block no. 0
 ** The offset:0 is at block 101
 => index:0 is at block no. 1
 ** The offset:128 is at block 102
Reading 200 bytes at 0 from file of length 200
 => index:0 is at block no. 0
 ** The offset:0 is at block 101
 => index:0 is at block no. 1
 ** The offset:128 is at block 102
File: [/swapspace] opened.
```

./nachos -cp ../test/shell shell - Before running Shell application, the file shuold be copied to Nachos disk.

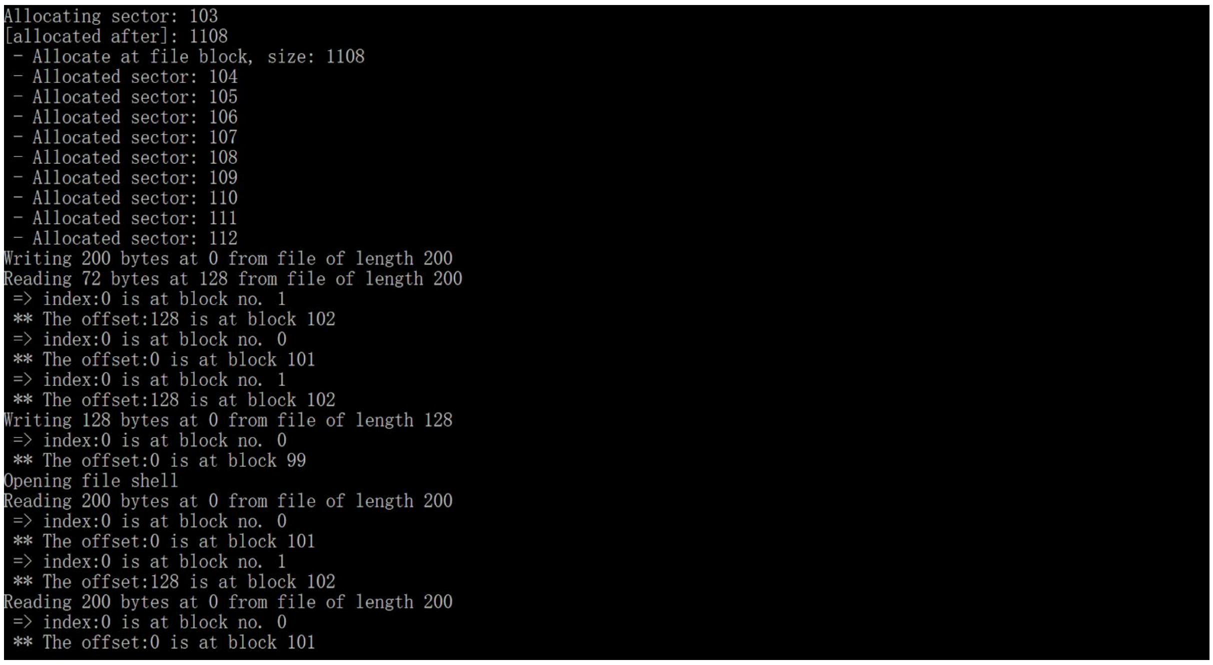


```
xhuang62@lcs-vc-cis486:~/Ass3_xhuang62/code/build.linux$ ./nachos -cp ../test/shell shell
File: [/swapspace] opened.
The -cp operation is completed.
```

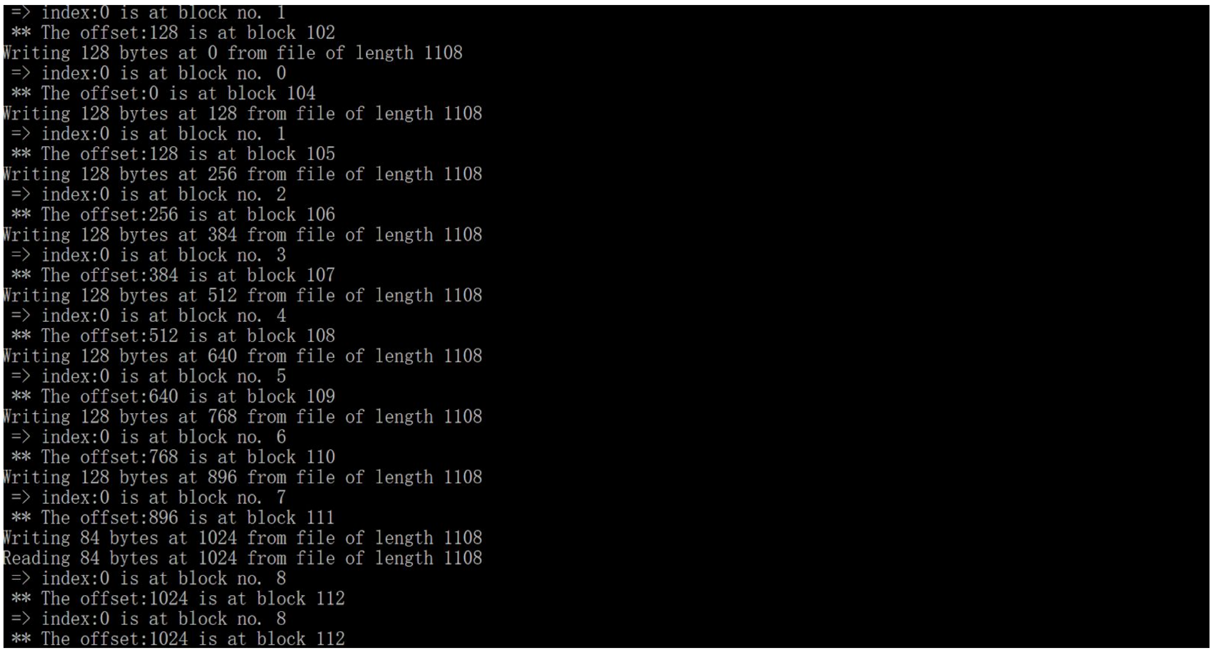
with debug mode: **./nachos -cp ../test/shell shell -d f**



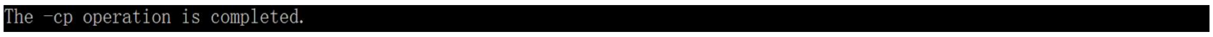
```
xhuang62@lcs-vc-cis486:~/Ass3_xhuang62/code/build.linux$ ./nachos -cp ../test/shell shell -d f
Initializing the file system.
Opening file swapspace
Reading 200 bytes at 0 from file of length 200
 => index:0 is at block no. 0
 ** The offset:0 is at block 101
 => index:0 is at block no. 1
 ** The offset:128 is at block 102
Reading 200 bytes at 0 from file of length 200
 => index:0 is at block no. 0
 ** The offset:0 is at block 101
 => index:0 is at block no. 1
 ** The offset:128 is at block 102
File: [/swapspace] opened.
Copying file ../test/shell of size 1108 to file shell
Creating file shell size 1108
Reading 200 bytes at 0 from file of length 200
 => index:0 is at block no. 0
 ** The offset:0 is at block 101
 => index:0 is at block no. 1
 ** The offset:128 is at block 102
Reading 128 bytes at 0 from file of length 128
 => index:0 is at block no. 0
 ** The offset:0 is at block 99
Allocating, size: 1108
[First sector] -1
[Last sector] -1
[Previous numbyte] 0
[required]: 1108
[allocated]: 0
[required]: 1108
[allocated]: 0
```



```
Allocating sector: 103
[allocated after]: 1108
 - Allocate at file block, size: 1108
 - Allocated sector: 104
 - Allocated sector: 105
 - Allocated sector: 106
 - Allocated sector: 107
 - Allocated sector: 108
 - Allocated sector: 109
 - Allocated sector: 110
 - Allocated sector: 111
 - Allocated sector: 112
Writing 200 bytes at 0 from file of length 200
Reading 72 bytes at 128 from file of length 200
 => index:0 is at block no. 1
 ** The offset:128 is at block 102
 => index:0 is at block no. 0
 ** The offset:0 is at block 101
 => index:0 is at block no. 1
 ** The offset:128 is at block 102
Writing 128 bytes at 0 from file of length 128
 => index:0 is at block no. 0
 ** The offset:0 is at block 99
Opening file shell
Reading 200 bytes at 0 from file of length 200
 => index:0 is at block no. 0
 ** The offset:0 is at block 101
 => index:0 is at block no. 1
 ** The offset:128 is at block 102
Reading 200 bytes at 0 from file of length 200
 => index:0 is at block no. 0
 ** The offset:0 is at block 101
```

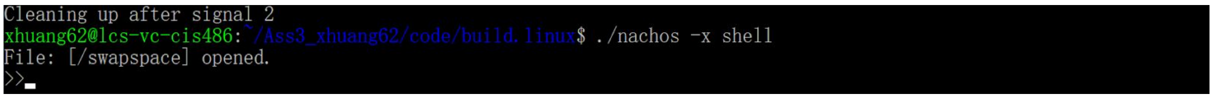


```
 => index:0 is at block no. 1
 ** The offset:128 is at block 102
Writing 128 bytes at 0 from file of length 1108
 => index:0 is at block no. 0
 ** The offset:0 is at block 104
Writing 128 bytes at 128 from file of length 1108
 => index:0 is at block no. 1
 ** The offset:128 is at block 105
Writing 128 bytes at 256 from file of length 1108
 => index:0 is at block no. 2
 ** The offset:256 is at block 106
Writing 128 bytes at 384 from file of length 1108
 => index:0 is at block no. 3
 ** The offset:384 is at block 107
Writing 128 bytes at 512 from file of length 1108
 => index:0 is at block no. 4
 ** The offset:512 is at block 108
Writing 128 bytes at 640 from file of length 1108
 => index:0 is at block no. 5
 ** The offset:640 is at block 109
Writing 128 bytes at 768 from file of length 1108
 => index:0 is at block no. 6
 ** The offset:768 is at block 110
Writing 128 bytes at 896 from file of length 1108
 => index:0 is at block no. 7
 ** The offset:896 is at block 111
Writing 84 bytes at 1024 from file of length 1108
Reading 84 bytes at 1024 from file of length 1108
 => index:0 is at block no. 8
 ** The offset:1024 is at block 112
 => index:0 is at block no. 8
 ** The offset:1024 is at block 112
```



```
The -cp operation is completed.
```

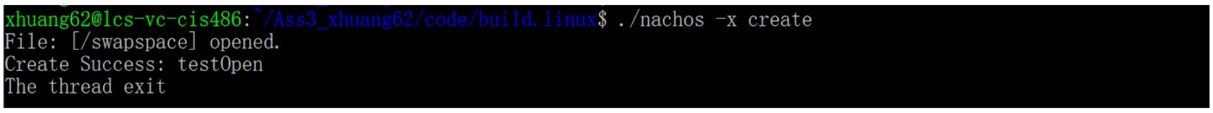
./nachos -x [Nachos user program] - Run the user program at Nachos



```
Cleaning up after signal 2
xhuang62@lcs-vc-cis486:~/Ass3_xhuang62/code/build.linux$ ./nachos -x shell
File: [/swapspace] opened.
>>
```

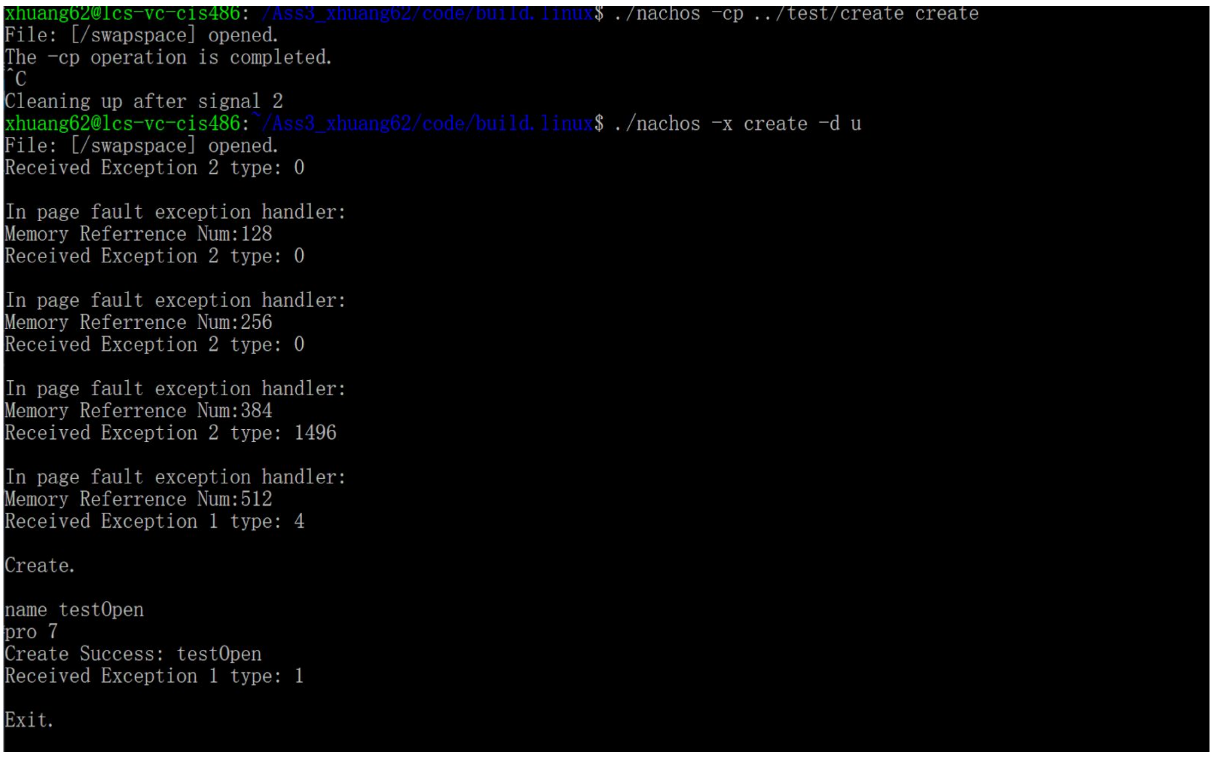
SysCall Test

- Create()



```
xhuang62@lcs-vc-cis486:~/Ass3_xhuang62/code/build.linux$ ./nachos -x create
File: [/swapspace] opened.
Create Success: testOpen
The thread exit
```

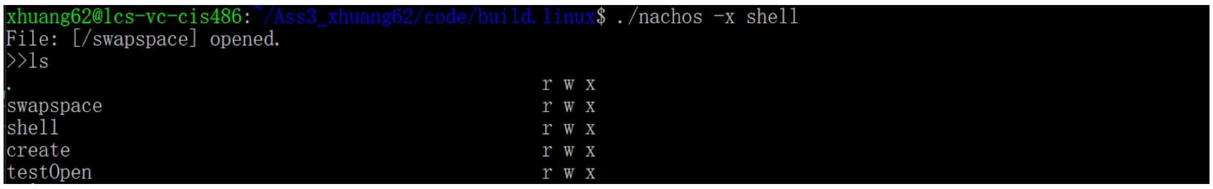
With debug mode -d u:



```
xhuang62@lcs-vc-cis486:~/Ass3_xhuang62/code/build.linux$ ./nachos -cp ../test/create create
File: [/swapspace] opened.
The -cp operation is completed.
^C
Cleaning up after signal 2
xhuang62@lcs-vc-cis486:~/Ass3_xhuang62/code/build.linux$ ./nachos -x create -d u
File: [/swapspace] opened.
Received Exception 2 type: 0

In page fault exception handler:
Memory Referrence Num:128
Received Exception 2 type: 0

In page fault exception handler:
Memory Referrence Num:256
Received Exception 2 type: 0

In page fault exception handler:
Memory Referrence Num:384
Received Exception 2 type: 1496

In page fault exception handler:
Memory Referrence Num:512
Received Exception 1 type: 4

Create.

name testOpen
pro 7
Create Success: testOpen
Received Exception 1 type: 1

Exit.
```

I copied the file to the nachos disk.

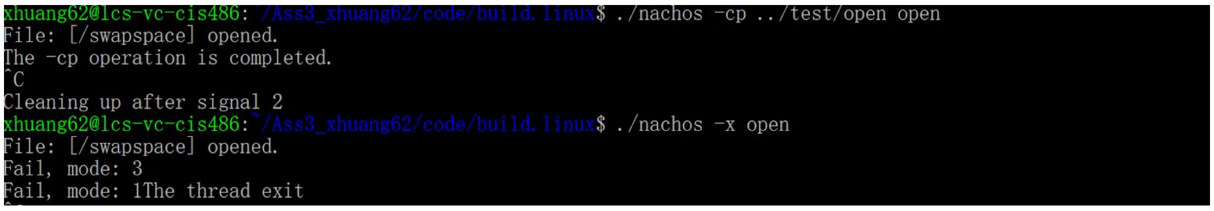
In create.c, I created a file named “testOpen”.



```
xhuang62@lcs-vc-cis486:~/Ass3_xhuang62/code/build.linux$ ./nachos -x shell
File: [/swapspace] opened.
>>ls
.                                                 r w x
swapspace                                         r w x
shell                                             r w x
create                                            r w x
testOpen                                          r w x
```

As seen in shell, the file testOpen is created.

-Open(), Write(), Seek(), Read(), Close()

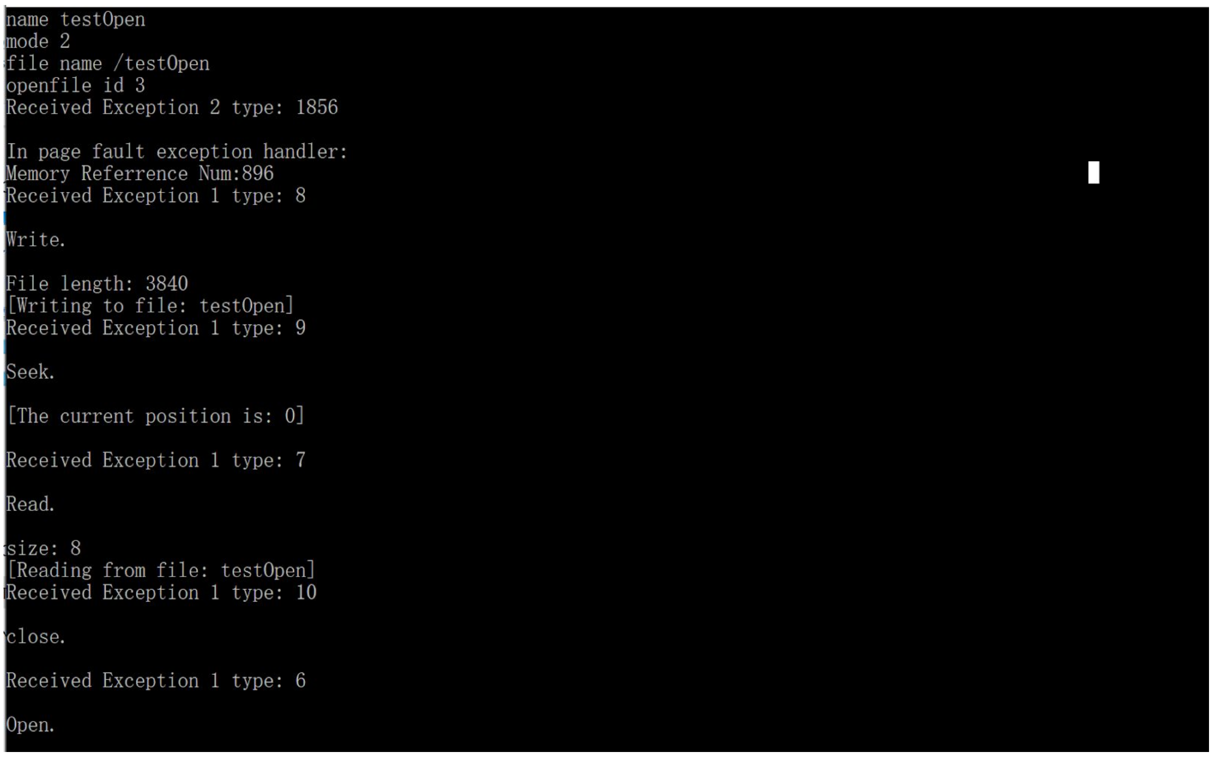


```
xhuang62@lcs-vc-cis486:~/Ass3_xhuang62/code/build.linux$ ./nachos -cp ../test/open open
File: [/swapspace] opened.
The -cp operation is completed.
^C
Cleaning up after signal 2
xhuang62@lcs-vc-cis486:~/Ass3_xhuang62/code/build.linux$ ./nachos -x open
File: [/swapspace] opened.
Fail, mode: 3
Fail, mode: 1The thread exit
```

This result can prove all the system calls work. Let’s see in debug mode -d u for detail:

In open.c:

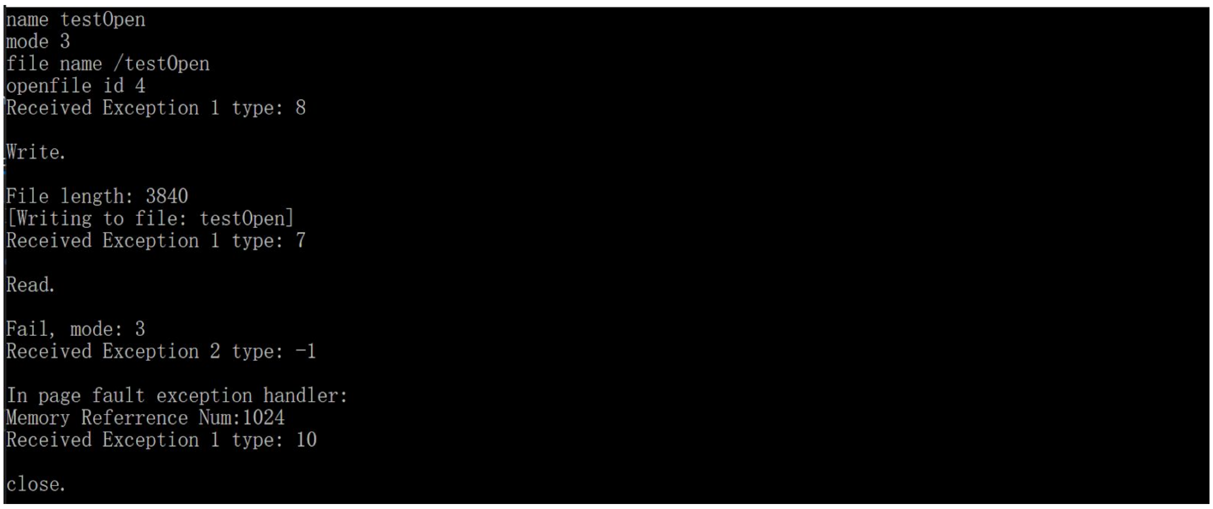
1) I opened the file testOpen in WR mode, and then write content “testOpen” to it. Then I use Seek(0) to set the position to 0, read the data and close the file. Everything should go fine.



```
name testOpen
mode 2
file name /testOpen
openfile id 3
Received Exception 2 type: 1856

In page fault exception handler:
Memory Referrence Num:896
Received Exception 1 type: 8

Write.

File length: 3840
[Writing to file: testOpen]
Received Exception 1 type: 9

Seek.

[The current position is: 0]

Received Exception 1 type: 7

Read.

size: 8
[Reading from file: testOpen]
Received Exception 1 type: 10

close.

Received Exception 1 type: 6

Open.
```

2) I opened the file testOpen in APPEND mode, and then write “testOpen” to it.

   Then I tried to read it and then close the file.

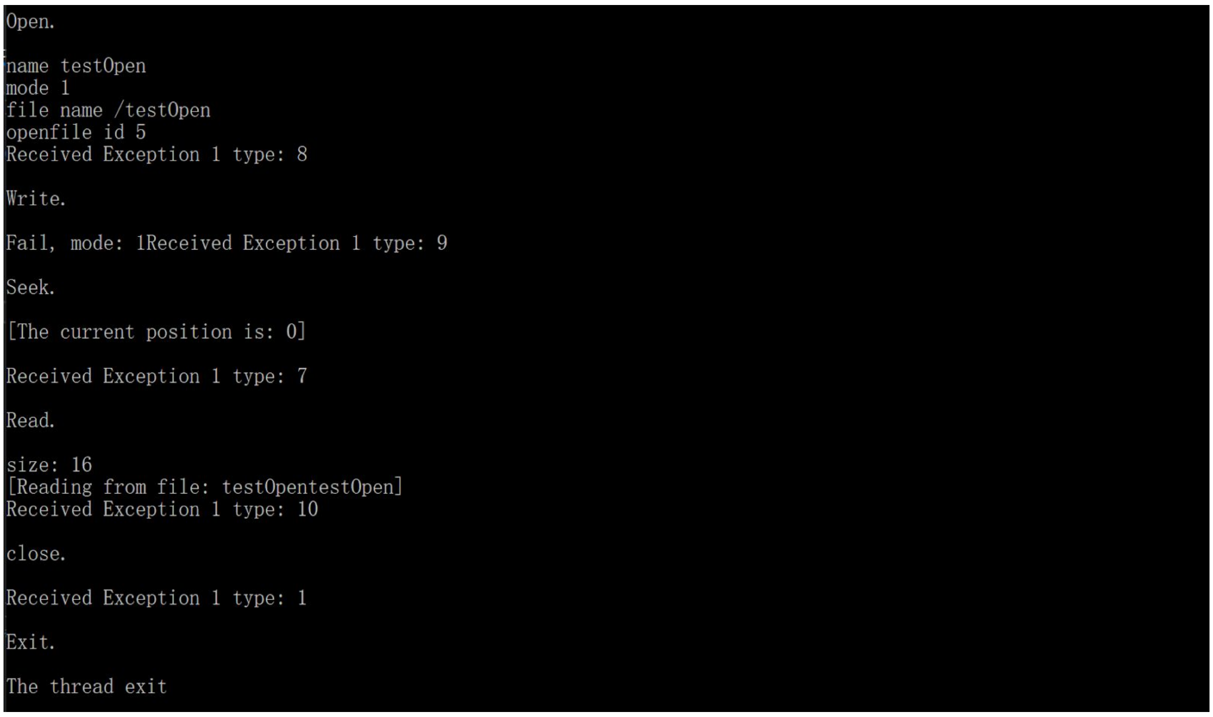


```
name testOpen
mode 3
file name /testOpen
openfile id 4
Received Exception 1 type: 8

Write.

File length: 3840
[Writing to file: testOpen]
Received Exception 1 type: 7

Read.

Fail, mode: 3
Received Exception 2 type: -1

In page fault exception handler:
Memory Referrence Num:1024
Received Exception 1 type: 10

close.
```

   The system call for read will be failed because there is no need to read in this mode.

3) I opened the file testOpen in RO mode, and then write content “testOpen” to it.

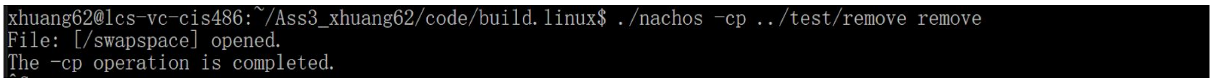
   Then I use Seek(0) to set the position to 0, read the data of length 16 and close the file.



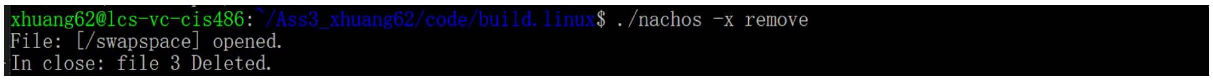
```
Open.

name testOpen
mode 1
file name /testOpen
openfile id 5
Received Exception 1 type: 8

Write.

Fail, mode: 1Received Exception 1 type: 9

Seek.

[The current position is: 0]

Received Exception 1 type: 7

Read.

size: 16
[Reading from file: testOpentestOpen]
Received Exception 1 type: 10

close.

Received Exception 1 type: 1

Exit.

The thread exit
```

   In RO mode, the write operation will fail. And the result for fetching first 16 bytes works correctly.

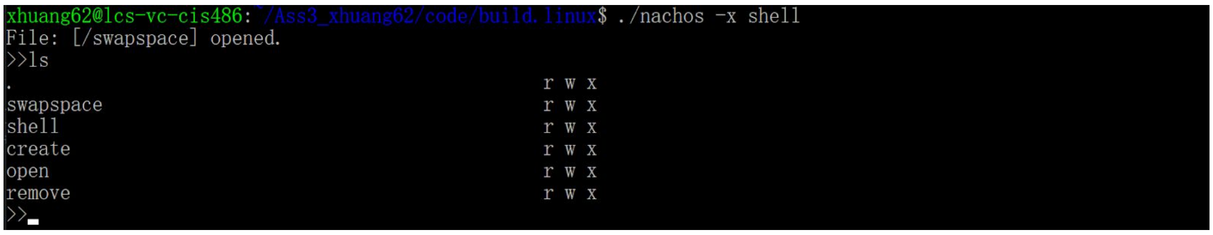
-Remove()



```
xhuang62@lcs-vc-cis486:~/Ass3_xhuang62/code/build.linux$ ./nachos -cp ../test/remove remove
File: [/swapspace] opened.
The -cp operation is completed.
```



```
xhuang62@lcs-vc-cis486:~/Ass3_xhuang62/code/build.linux$ ./nachos -x remove
File: [/swapspace] opened.
In close: file 3 Deleted.
```



```
xhuang62@lcs-vc-cis486:~/Ass3_xhuang62/code/build.linux$ ./nachos -x shell
File: [/swapspace] opened.
>>ls
.                                                r w x
swapspace                                        r w x
shell                                            r w x
create                                           r w x
open                                             r w x
remove                                           r w x
>>
```

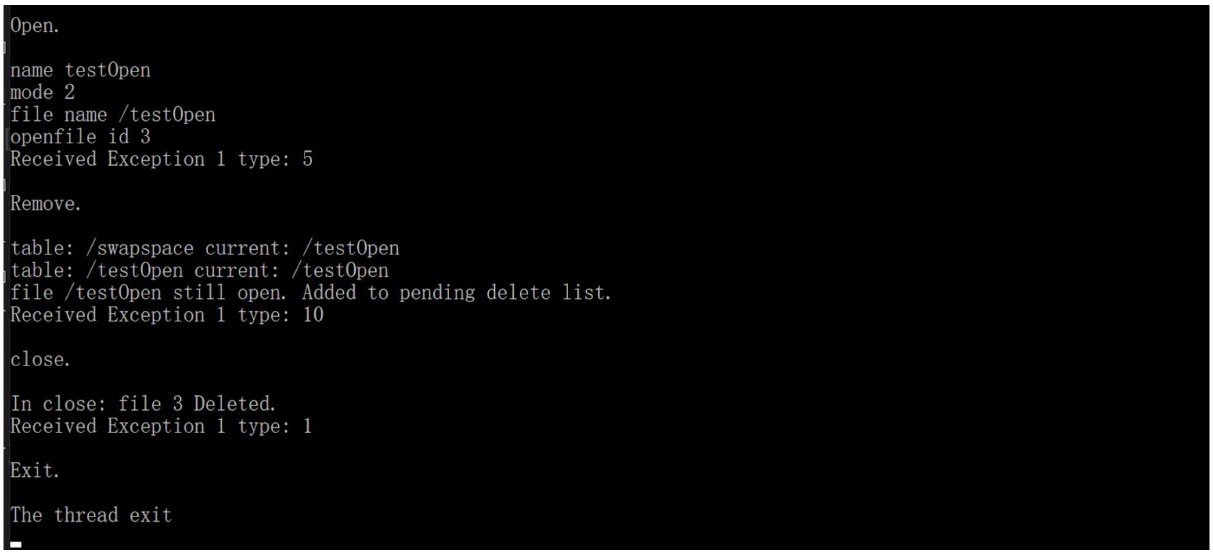
The the file testOpen will be removed.

check in shell: the file testOpen is removed.

In debug mode:

In remove.c:

I opened a file testOpen and removed it before closing. It will be added to pending delete list and wait until the file is not opened by any threads. Then I closed the file, and the file is removed.

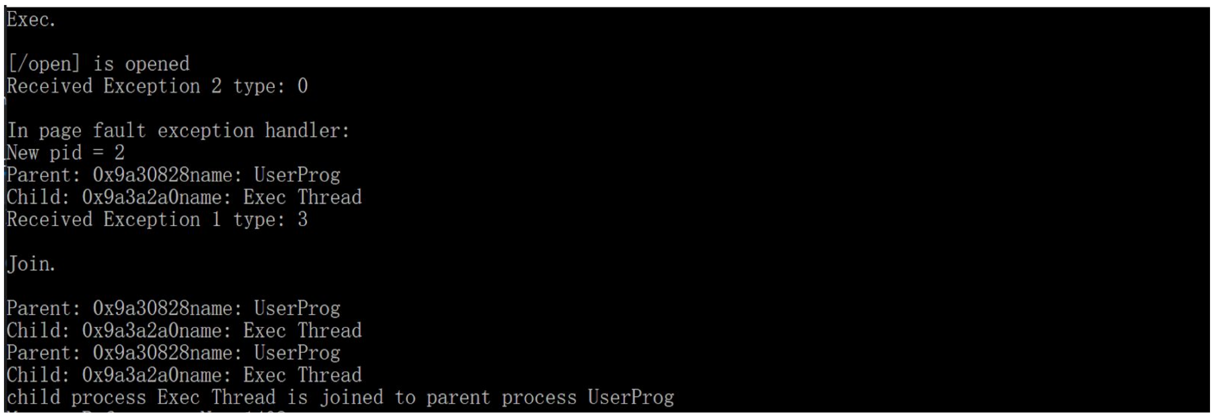


```
Open.

name testOpen
mode 2
file name /testOpen
openfile id 3
Received Exception 1 type: 5

Remove.

table: /swapspace current: /testOpen
table: /testOpen current: /testOpen
file /testOpen still open. Added to pending delete list.
Received Exception 1 type: 10

close.

In close: file 3 Deleted.
Received Exception 1 type: 1

Exit.

The thread exit
```

Exec(), Exit(), Join()

Just test using shell in debug -u mode:

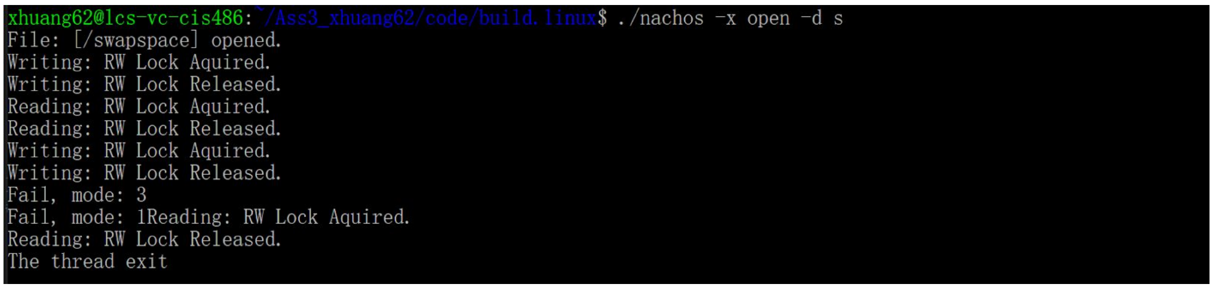
When typing “open” to execute “open” program:



```
Exec.

[/open] is opened
Received Exception 2 type: 0

In page fault exception handler:
New pid = 2
Parent: 0x9a30828name: UserProg
Child: 0x9a3a2a0name: Exec Thread
Received Exception 1 type: 3

Join.

Parent: 0x9a30828name: UserProg
Child: 0x9a3a2a0name: Exec Thread
Parent: 0x9a30828name: UserProg
Child: 0x9a3a2a0name: Exec Thread
child process Exec Thread is joined to parent process UserProg
```

Synchronization

Run open file in synch debug mode -d s



```
xhuang62@lcs-vc-cis486:~/Ass3_xhuang62/code/build.linux$ ./nachos -x open -d s
File: [/swapspace] opened.
Writing: RW Lock Aquired.
Writing: RW Lock Released.
Reading: RW Lock Aquired.
Reading: RW Lock Released.
Writing: RW Lock Aquired.
Writing: RW Lock Released.
Fail, mode: 3
Fail, mode: 1Reading: RW Lock Aquired.
Reading: RW Lock Released.
The thread exit
```

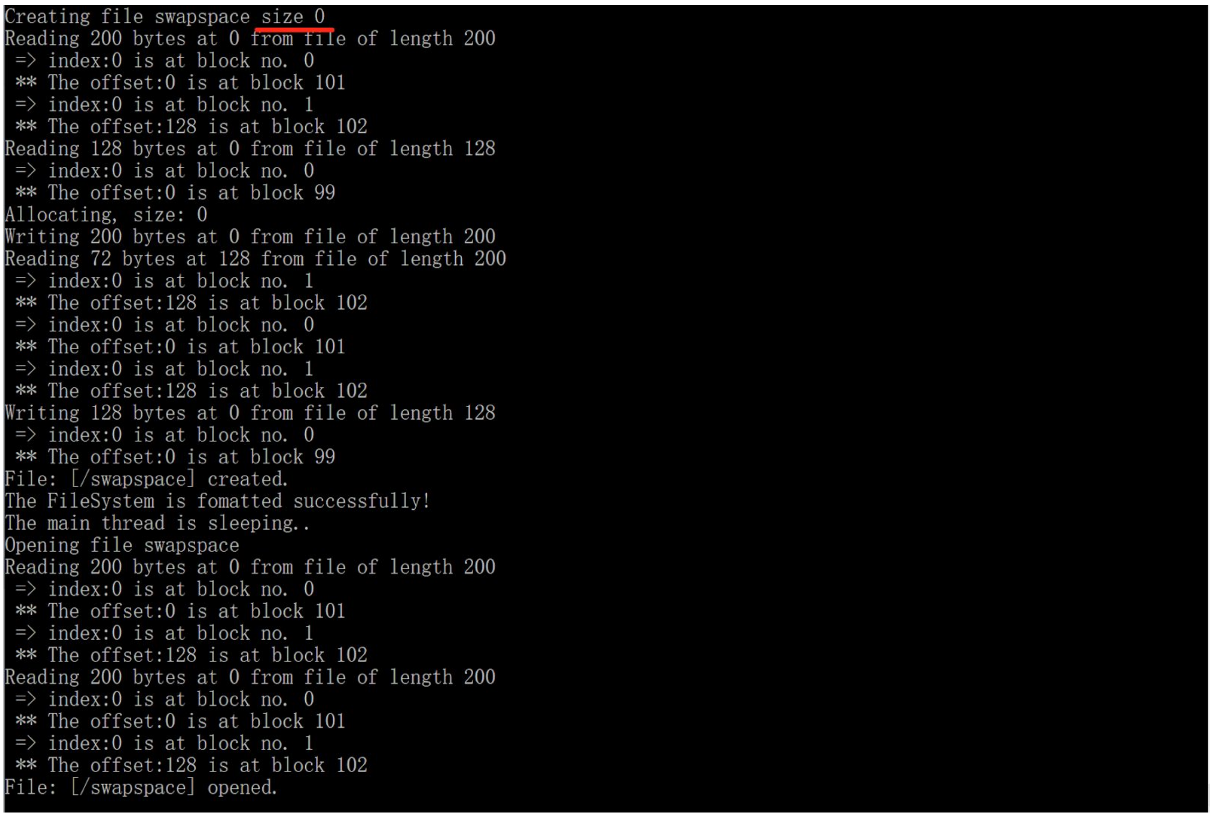
The synchronize information is printed.

The output is as expected.

Make maximum size of a file beyond 4Kbytes and Extensible files

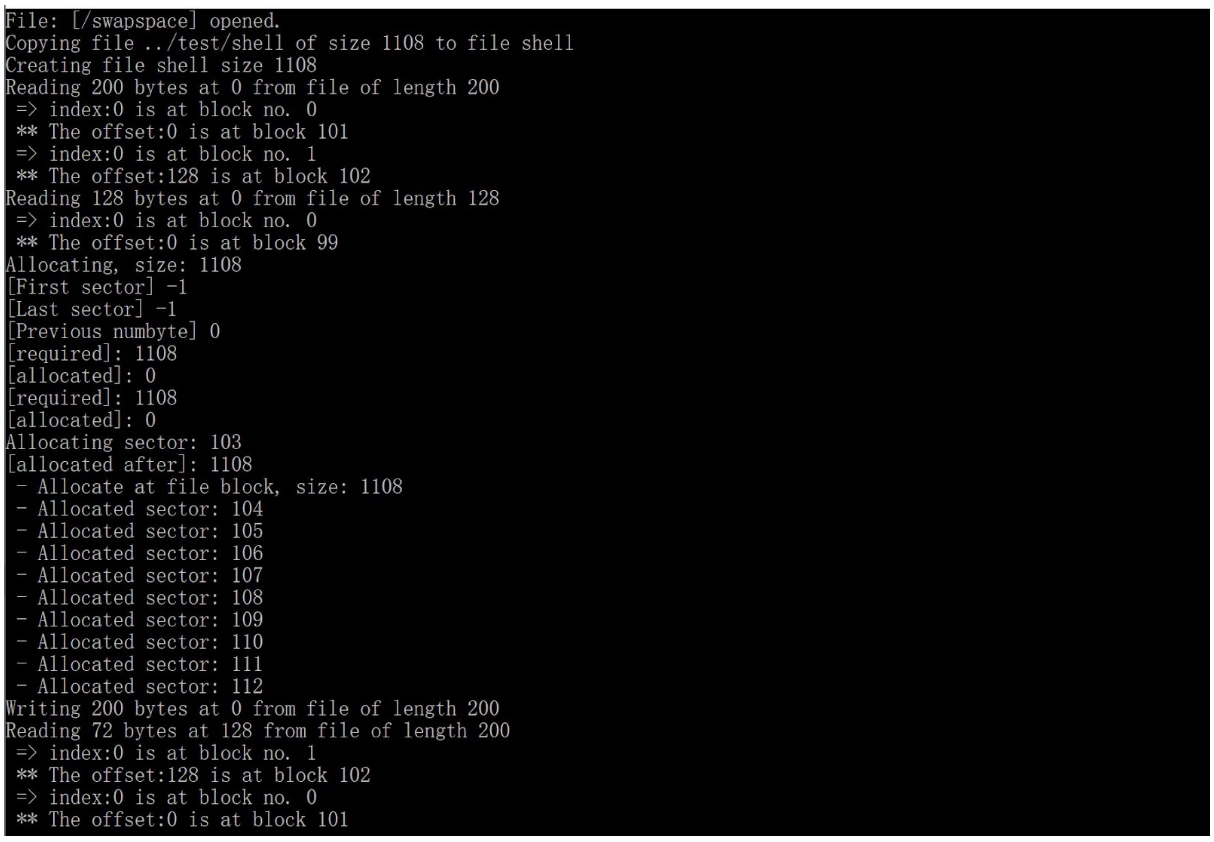
I created a file swapspace for memory management. First it was 0, when I run different programs, it will be expanded.

The information can be seen at debug -d f mode:



```
Creating file swapspace size 0
Reading 200 bytes at 0 from file of length 200
 => index:0 is at block no. 0
 ** The offset:0 is at block 101
 => index:0 is at block no. 1
 ** The offset:128 is at block 102
Reading 128 bytes at 0 from file of length 128
 => index:0 is at block no. 0
 ** The offset:0 is at block 99
Allocating, size: 0
Writing 200 bytes at 0 from file of length 200
Reading 72 bytes at 128 from file of length 200
 => index:0 is at block no. 1
 ** The offset:128 is at block 102
 => index:0 is at block no. 0
 ** The offset:0 is at block 101
 => index:0 is at block no. 1
 ** The offset:128 is at block 102
Writing 128 bytes at 0 from file of length 128
 => index:0 is at block no. 0
 ** The offset:0 is at block 99
File: [/swapspace] created.
The FileSystem is fomatted successfully!
The main thread is sleeping..
Opening file swapspace
Reading 200 bytes at 0 from file of length 200
 => index:0 is at block no. 0
 ** The offset:0 is at block 101
 => index:0 is at block no. 1
 ** The offset:128 is at block 102
Reading 200 bytes at 0 from file of length 200
 => index:0 is at block no. 0
 ** The offset:0 is at block 101
 => index:0 is at block no. 1
 ** The offset:128 is at block 102
File: [/swapspace] opened.
```

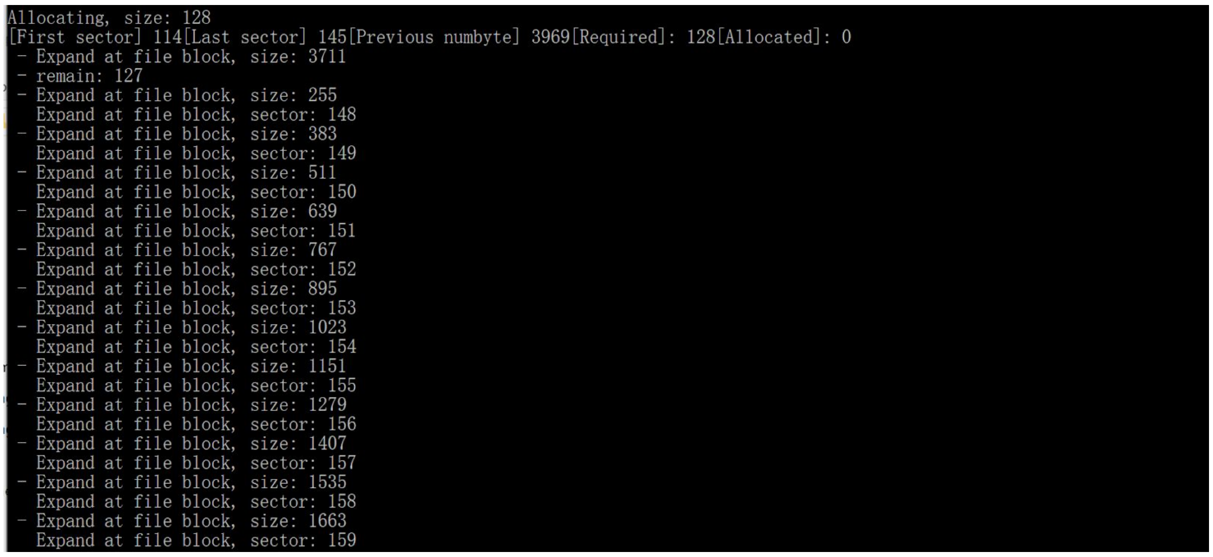
Then run -cp ../test/shell shell -d f command:



```
File: [/swapspace] opened.
Copying file ../test/shell of size 1108 to file shell
Creating file shell size 1108
Reading 200 bytes at 0 from file of length 200
 => index:0 is at block no. 0
 ** The offset:0 is at block 101
 => index:0 is at block no. 1
 ** The offset:128 is at block 102
Reading 128 bytes at 0 from file of length 128
 => index:0 is at block no. 0
 ** The offset:0 is at block 99
Allocating, size: 1108
[First sector] -1
[Last sector] -1
[Previous numbyte] 0
[required]: 1108
[allocated]: 0
[required]: 1108
[allocated]: 0
Allocating sector: 103
[allocated after]: 1108
 - Allocate at file block, size: 1108
 - Allocated sector: 104
 - Allocated sector: 105
 - Allocated sector: 106
 - Allocated sector: 107
 - Allocated sector: 108
 - Allocated sector: 109
 - Allocated sector: 110
 - Allocated sector: 111
 - Allocated sector: 112
Writing 200 bytes at 0 from file of length 200
Reading 72 bytes at 128 from file of length 200
 => index:0 is at block no. 1
 ** The offset:128 is at block 102
 => index:0 is at block no. 0
 ** The offset:0 is at block 101
```

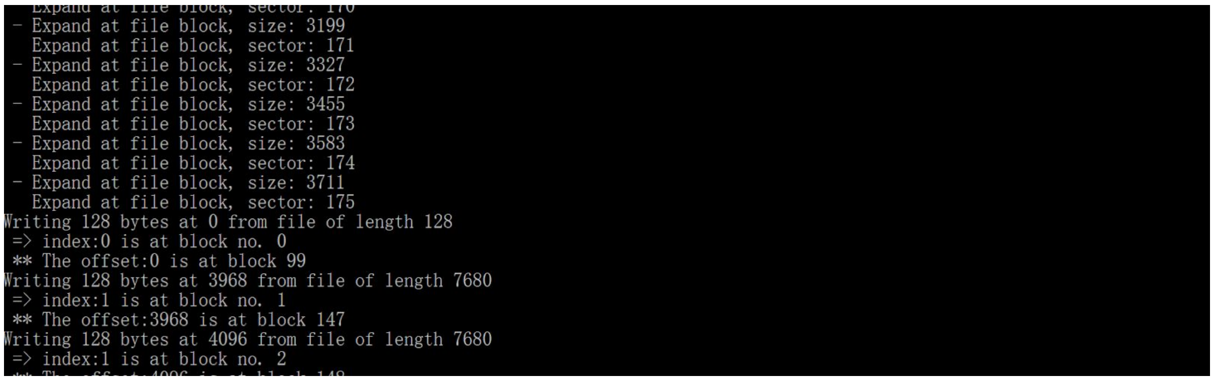
The space is allocated.

I ran matmult, making the file expanded fast.



```
Allocating, size: 128
[First sector] 114[Last sector] 145[Previous numbyte] 3969[Required]: 128[Allocated]: 0
 - Expand at file block, size: 3711
 - remain: 127
 - Expand at file block, size: 255
   Expand at file block, sector: 148
 - Expand at file block, size: 383
   Expand at file block, sector: 149
 - Expand at file block, size: 511
   Expand at file block, sector: 150
 - Expand at file block, size: 639
   Expand at file block, sector: 151
 - Expand at file block, size: 767
   Expand at file block, sector: 152
 - Expand at file block, size: 895
   Expand at file block, sector: 153
 - Expand at file block, size: 1023
   Expand at file block, sector: 154
 - Expand at file block, size: 1151
   Expand at file block, sector: 155
 - Expand at file block, size: 1279
   Expand at file block, sector: 156
 - Expand at file block, size: 1407
   Expand at file block, sector: 157
 - Expand at file block, size: 1535
   Expand at file block, sector: 158
 - Expand at file block, size: 1663
   Expand at file block, sector: 159
```

…

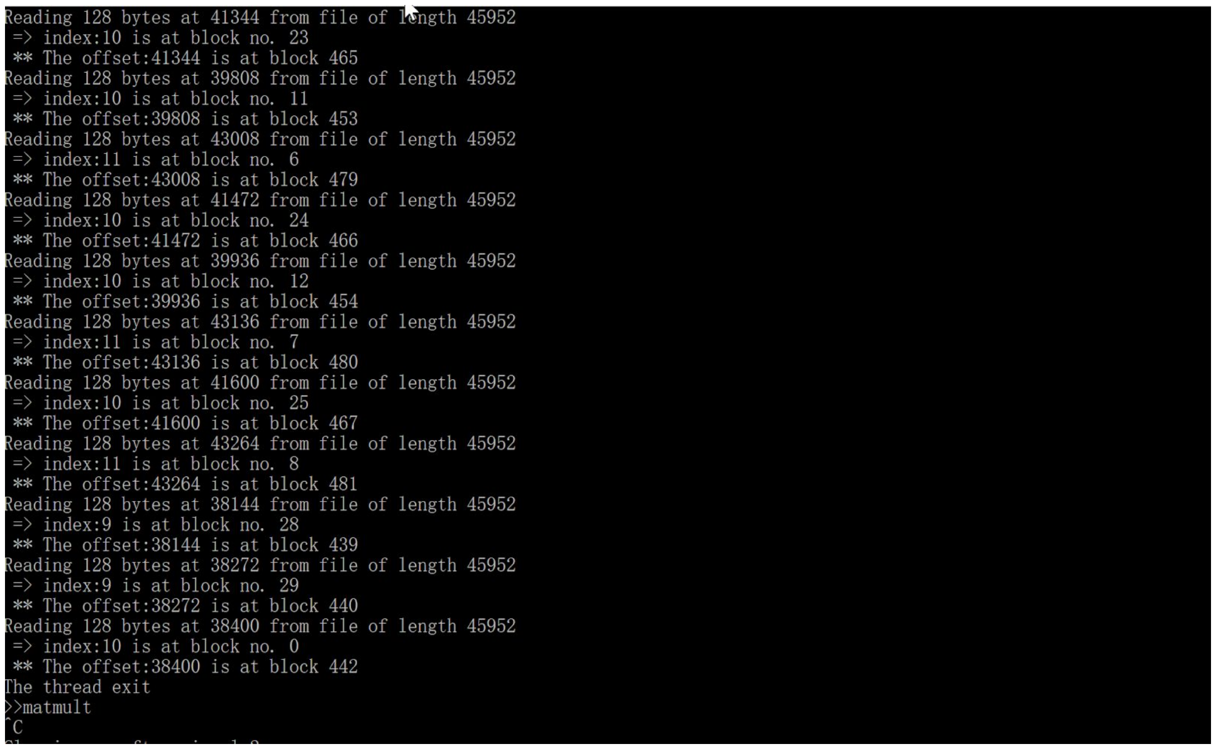


```
   Expand at file block, sector: 170
 - Expand at file block, size: 3199
   Expand at file block, sector: 171
 - Expand at file block, size: 3327
   Expand at file block, sector: 172
 - Expand at file block, size: 3455
   Expand at file block, sector: 173
 - Expand at file block, size: 3583
   Expand at file block, sector: 174
 - Expand at file block, size: 3711
   Expand at file block, sector: 175
Writing 128 bytes at 0 from file of length 128
 => index:0 is at block no. 0
 ** The offset:0 is at block 99
Writing 128 bytes at 3968 from file of length 7680
 => index:1 is at block no. 1
 ** The offset:3968 is at block 147
Writing 128 bytes at 4096 from file of length 7680
 => index:1 is at block no. 2
 ** The offset:4096 is at block 148
```

When the requred space is larger than 1 fileblock’s maximum size 30 * 128 = 3840, it will expand to more blocks.

The swapspace now is expanded to 7680.

After running matmult multiple times, the file size is much larger.

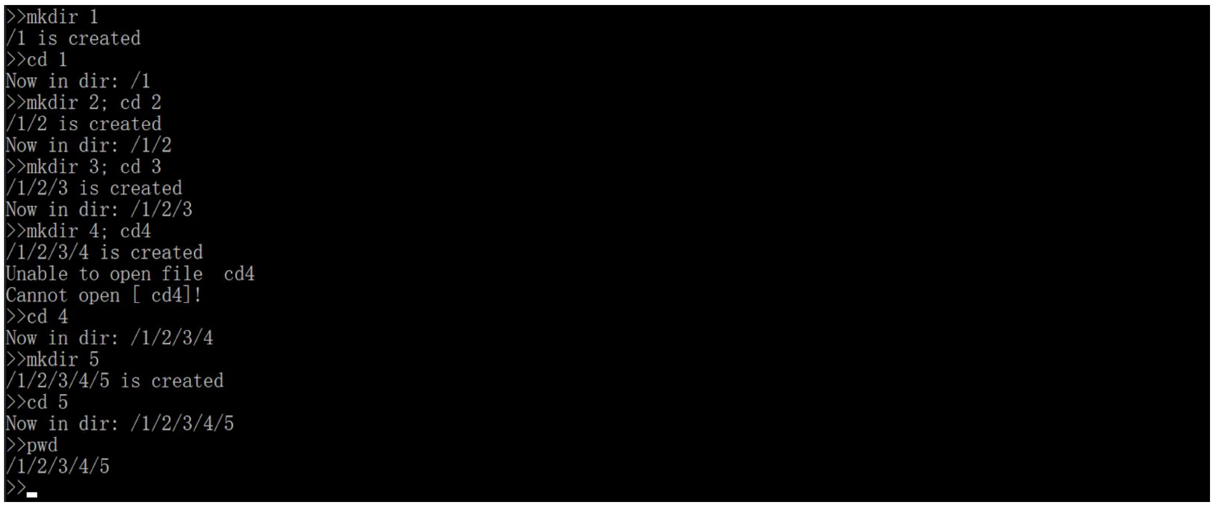


```
Reading 128 bytes at 41344 from file of length 45952
 => index:10 is at block no. 23
 ** The offset:41344 is at block 465
Reading 128 bytes at 39808 from file of length 45952
 => index:10 is at block no. 11
 ** The offset:39808 is at block 453
Reading 128 bytes at 43008 from file of length 45952
 => index:11 is at block no. 6
 ** The offset:43008 is at block 479
Reading 128 bytes at 41472 from file of length 45952
 => index:10 is at block no. 24
 ** The offset:41472 is at block 466
Reading 128 bytes at 39936 from file of length 45952
 => index:10 is at block no. 12
 ** The offset:39936 is at block 454
Reading 128 bytes at 43136 from file of length 45952
 => index:11 is at block no. 7
 ** The offset:43136 is at block 480
Reading 128 bytes at 41600 from file of length 45952
 => index:10 is at block no. 25
 ** The offset:41600 is at block 467
Reading 128 bytes at 43264 from file of length 45952
 => index:11 is at block no. 8
 ** The offset:43264 is at block 481
Reading 128 bytes at 38144 from file of length 45952
 => index:9 is at block no. 28
 ** The offset:38144 is at block 439
Reading 128 bytes at 38272 from file of length 45952
 => index:9 is at block no. 29
 ** The offset:38272 is at block 440
Reading 128 bytes at 38400 from file of length 45952
 => index:10 is at block no. 0
 ** The offset:38400 is at block 442
The thread exit
>>matmult
^C
```

As seen above, the file is far more larger than 4 KByte.

Hierarchical Directory

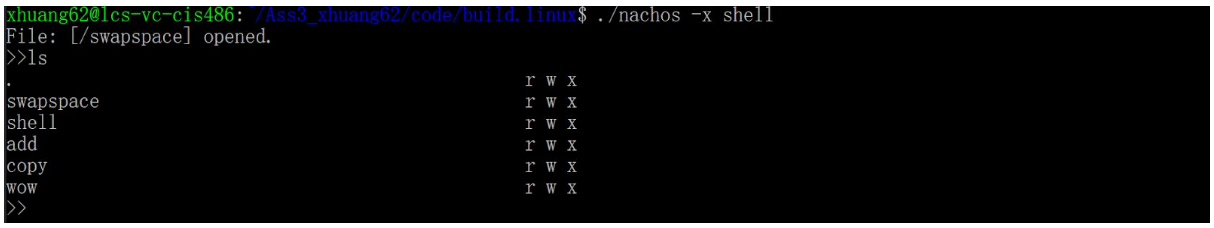
I created multiple child directory. It works well.



```
>>mkdir 1
/1 is created
>>cd 1
Now in dir: /1
>>mkdir 2; cd 2
/1/2 is created
Now in dir: /1/2
>>mkdir 3; cd 3
/1/2/3 is created
Now in dir: /1/2/3
>>mkdir 4; cd4
/1/2/3/4 is created
Unable to open file  cd4
Cannot open [ cd4]!
>>cd 4
Now in dir: /1/2/3/4
>>mkdir 5
/1/2/3/4/5 is created
>>cd 5
Now in dir: /1/2/3/4/5
>>pwd
/1/2/3/4/5
>>
```

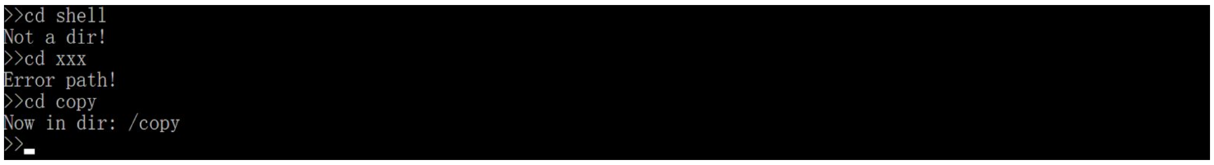
Shell Application and Command Line Test

>> ls



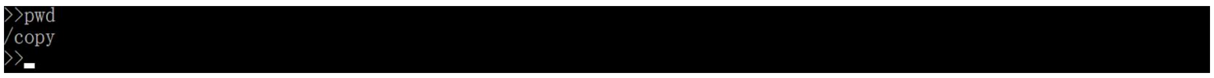
```
xhuang62@lcs-vc-cis486:~/Ass3_xhuang62/code/build.linux$ ./nachos -x shell
File: [/swapspace] opened.
>>ls
.                                                    r w x
swapspace                                            r w x
shell                                                r w x
add                                                  r w x
copy                                                 r w x
wow                                                  r w x
>>
```

>> cd [dir name]



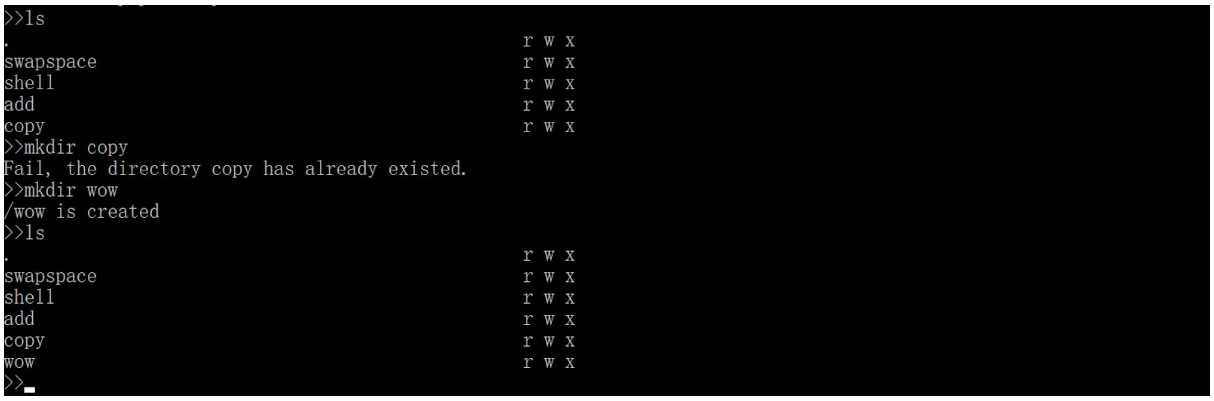
```
>>cd shell
Not a dir!
>>cd xxx
Error path!
>>cd copy
Now in dir: /copy
>>
```

>> pwd



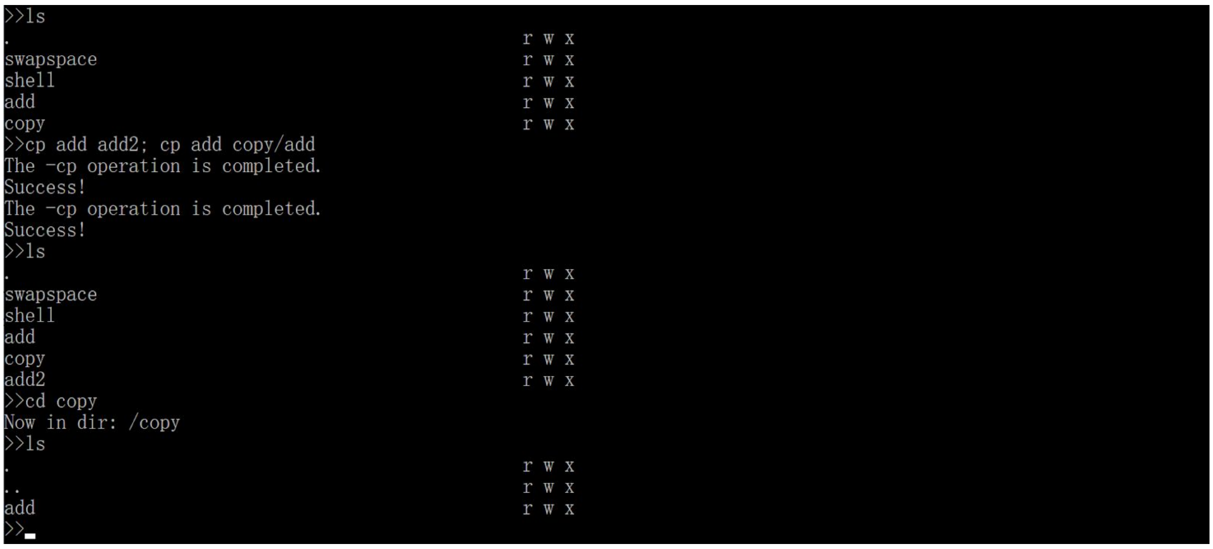
```
>>pwd
/copy
>>
```

>> mkdir [dir name]



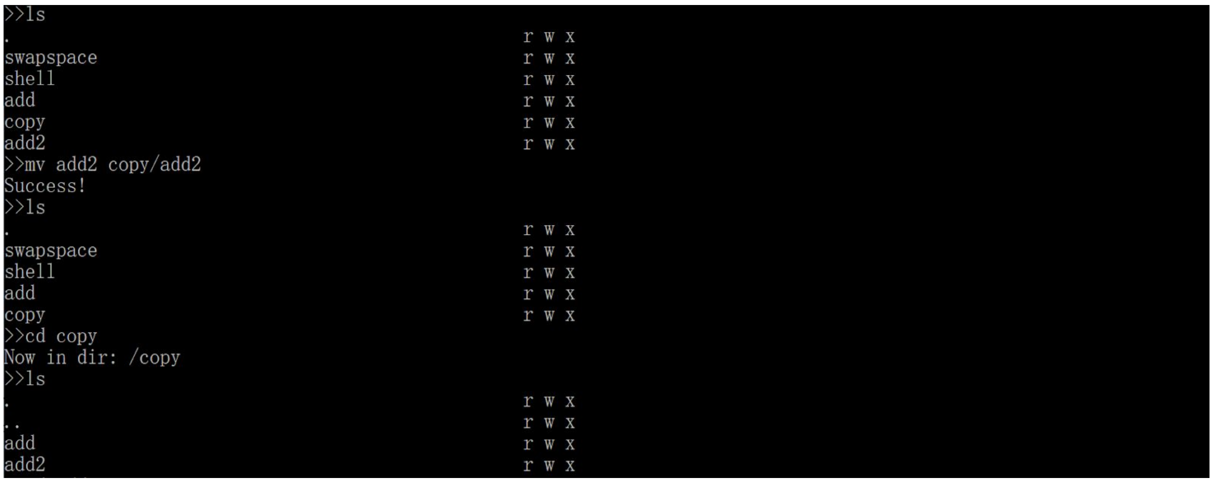
```
>>ls
.                                                    r w x
swapspace                                            r w x
shell                                                r w x
add                                                  r w x
copy                                                 r w x
>>mkdir copy
Fail, the directory copy has already existed.
>>mkdir wow
/wow is created
>>ls
.                                                    r w x
swapspace                                            r w x
shell                                                r w x
add                                                  r w x
copy                                                 r w x
wow                                                  r w x
>>
```

>> cp [source file name] [target file name]

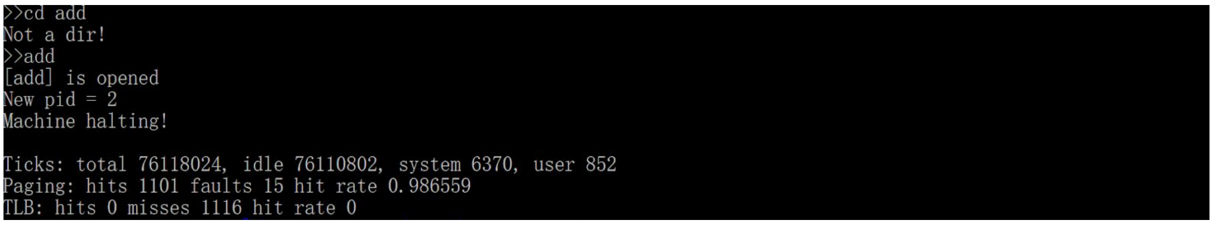


```
>>ls
.                                                  r w x
swapspace                                          r w x
shell                                              r w x
add                                                r w x
copy                                               r w x
>>cp add add2; cp add copy/add
The -cp operation is completed.
Success!
The -cp operation is completed.
Success!
>>ls
.                                                  r w x
swapspace                                          r w x
shell                                              r w x
add                                                r w x
copy                                               r w x
add2                                               r w x
>>cd copy
Now in dir: /copy
>>ls
.                                                  r w x
..                                                 r w x
add                                                r w x
>>
```

>> mv [source file name] [target file name]

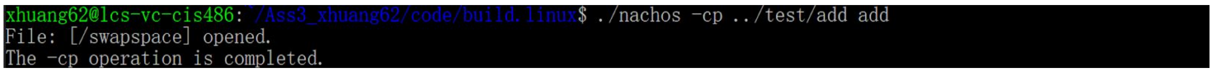


```
>>ls
.                                                  r w x
swapspace                                          r w x
shell                                              r w x
add                                                r w x
copy                                               r w x
add2                                               r w x
>>mv add2 copy/add2
Success!
>>ls
.                                                  r w x
swapspace                                          r w x
shell                                              r w x
add                                                r w x
copy                                               r w x
>>cd copy
Now in dir: /copy
>>ls
.                                                  r w x
..                                                 r w x
add                                                r w x
add2                                               r w x
```

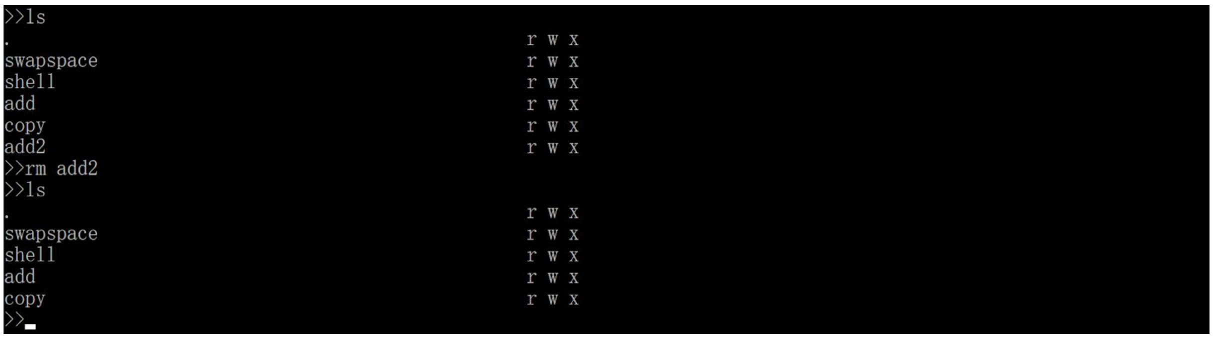


```
>>cd add
Not a dir!
>>add
[add] is opened
New pid = 2
Machine halting!

Ticks: total 76118024, idle 76110802, system 6370, user 852
Paging: hits 1101 faults 15 hit rate 0.986559
TLB: hits 0 misses 1116 hit rate 0
```

>> rm [file name]

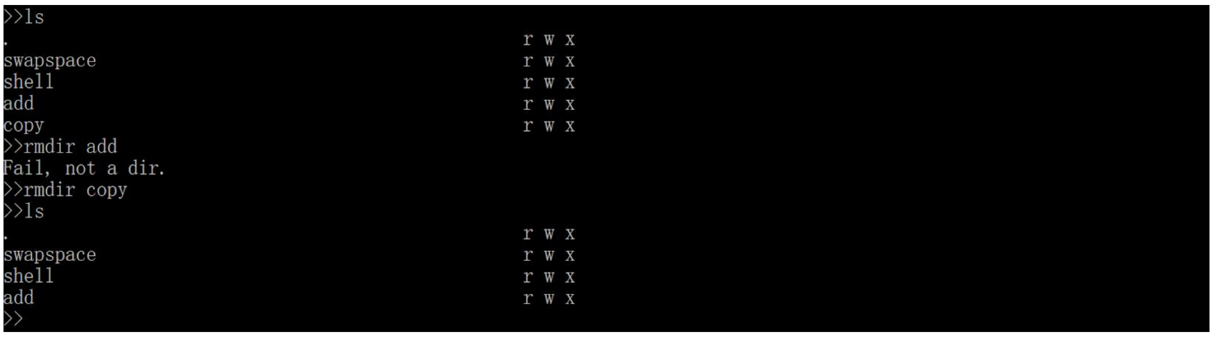


```
xhuang62@lcs-vc-cis486:~/Ass3_xhuang62/code/build.linux$ ./nachos -cp ../test/add add
File: [/swapspace] opened.
The -cp operation is completed.
```

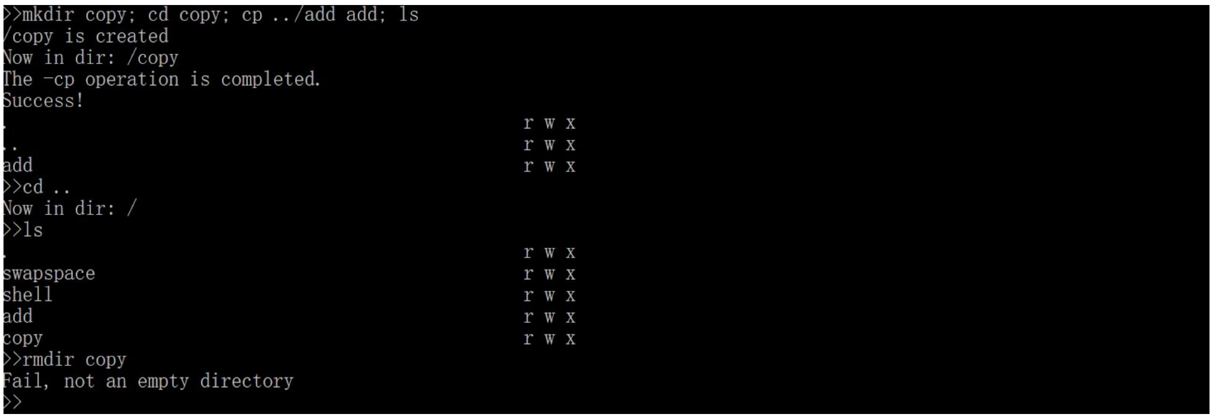


```
>>ls
.                                                  r w x
swapspace                                          r w x
shell                                              r w x
add                                                r w x
copy                                               r w x
add2                                               r w x
>>rm add2
>>ls
.                                                  r w x
swapspace                                          r w x
shell                                              r w x
add                                                r w x
copy                                               r w x
>>
```

>> rmdir [dir name]

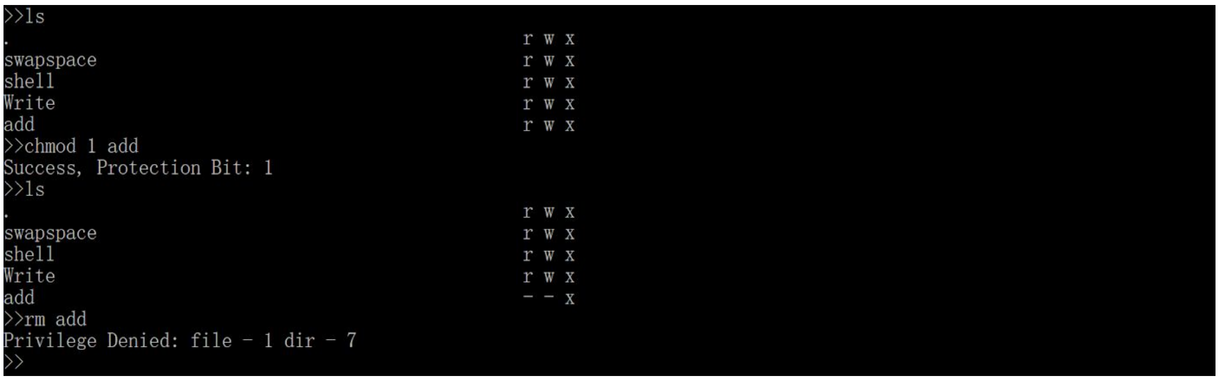


```
>>ls
.                                                   r w x
swapspace                                           r w x
shell                                               r w x
add                                                 r w x
copy                                                r w x
>>rmdir add
Fail, not a dir.
>>rmdir copy
>>ls
.                                                   r w x
swapspace                                           r w x
shell                                               r w x
add                                                 r w x
>>
```

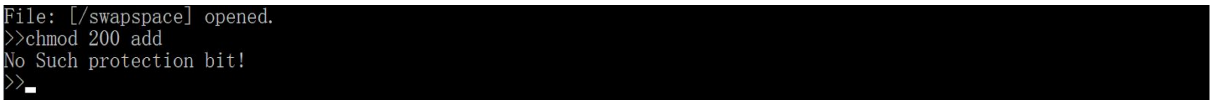


```
>>mkdir copy; cd copy; cp ../add add; ls
/copy is created
Now in dir: /copy
The -cp operation is completed.
Success!
.                                                   r w x
..                                                  r w x
add                                                 r w x
>>cd ..
Now in dir: /
>>ls
.                                                   r w x
swapspace                                           r w x
shell                                               r w x
add                                                 r w x
copy                                                r w x
>>rmdir copy
Fail, not an empty directory
>>
```

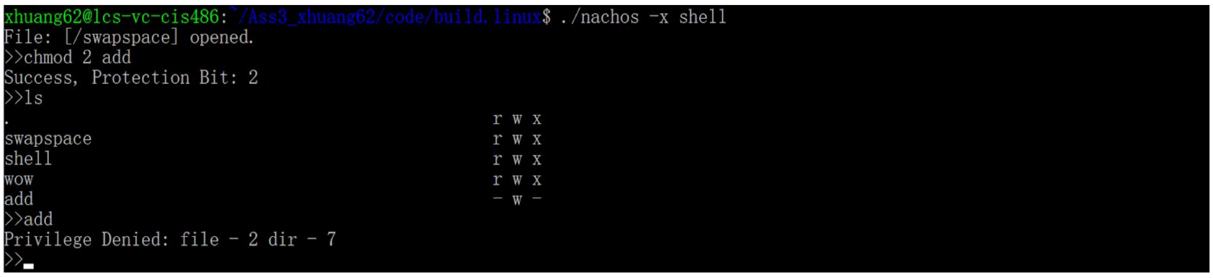
>> chmod [mode(0-7)] [file name]



```
>>ls
.                                                   r w x
swapspace                                           r w x
shell                                               r w x
Write                                               r w x
add                                                 r w x
>>chmod 1 add
Success, Protection Bit: 1
>>ls
.                                                   r w x
swapspace                                           r w x
shell                                               r w x
Write                                               r w x
add                                                 - - x
>>rm add
Privilege Denied: file - 1 dir - 7
>>
```



```
File: [/swapspace] opened.
>>chmod 200 add
No Such protection bit!
>>
```



```
xhuang62@lcs-vc-cis486:~/Ass3_xhuang62/code/build.linux$ ./nachos -x shell
File: [/swapspace] opened.
>>chmod 2 add
Success, Protection Bit: 2
>>ls
.                                                   r w x
swapspace                                           r w x
shell                                               r w x
wow                                                 r w x
add                                                 - w -
>>add
Privilege Denied: file - 2 dir - 7
>>
```