EEL5737 Assignemnt 2

Sagnik Ghosh (sagnik.ghosh@ufl.edu)

October 16, 2020

Question 1.

Have implemented the algorithm provided in the book for PathToInodeNumber and GeneralPathToInodeNumber. Both these functions provides Inode Number given a path. The basic difference is GeneralPathToIndodeNumber supports absolute path whereas PathToInodeNumber works relative to the current directory. NameToInodeNumber just refers the arguments to Lookup and fetches the inode for the file.

Algorithm 1 PathToInodeNumber($path, dir$)

if $PlainName(path)$ **then**
  return NameToInodeNumber($path, dir$)
else
  $dir \leftarrow$ Lookup(First($path$),dir)
  $path \leftarrow$ Rest($path$)
  return PathToInodeNumber($path, dir$)
end

Algorithm 2 GeneralPathToInodeNumber($path, cwd$)

if $path[0] ==$ "/" **then**
  return PathToInodeNumber($path, 1$)
else
  return PathToInodeNumber($path, cwd$)
end

The functions required for these, have been implemented as required. These are First, Rest and PlainName. provide the pseduocode for this below. The idea here is to split the path string on "/" and use accordingly. The PlainName returns a boolean True/False after checking whether the path contains any "/" symbol.

Listing 1: Implementation of required functions

```
def First(self,path):
  delimetedPath = path.split("/")
  logging.debug('HelperGetDirectoryPathFirst: ' + delimetedPath[0])
  return delimetedPath[0]

def Rest(self,path):
  delimetedPath = path.split("/", 1)
  logging.debug('DirectoryPathRest: ' + delimetedPath[1])
  return delimetedPath[1]

def PlainName(self, path):
  if "/" not in path:
    return True
  return False
```

Question 2.

Link is a function added to the file layer. The goal is to link a "name" to a fully defined path "target", which is a file. This is using the same file inode in a different directory. This is creating a hard link. For capturing different validation errors in the inputs, I have used several return codes. Below is the implementation:

Algorithm 3 Link($target, name, cwd$)

if *name contains "/"* **then**
  return -1
else if $Lookup(name, cwd) \neq -1$ **then**
  return -2
else
  $i \leftarrow GeneralPathToInodeNumber(target, cwd)$
  $inobj \leftarrow InodeNumber(i)$
  if $inobj.type \neq file$ **then**
    return -3
  else
    $dirinode \leftarrow InodeNumber(cwd)$
    InsertFleNameInodeNumber($dirinode, name, i$)
    $inobj.refcnt$ ←inopbj.refcnt+1
    return 0
  end
end

Question 3.

mkdir dirname: I have added a condition check for *dirname* not contain a "/", which will throw an error to user. Else, this will call FileName's Create method with dir = *cwd*, name = *dirname*, type = $INODE\_TYPE\_DIR$. If the call to create fails to create (return value -1), user has to check the debug to find the error message.

append filename string: First, the filename is looked up in the current directory. If the file isn't available, throws error in command line. If the inode number i is found, i we call FileName's Write method with file_inode_number = i, offset = $inobj.size$ and data = *string*. *inobj* refers to the inode.

Listing 2: Implementation of append command

```
def append(self, filename, string):
  i = self.FileObject.Lookup(filename,self.cwd)
  if i == -1:
    print("Error: File not found\n")
    return -1
  inobj = InodeNumber(self.FileObject.RawBlocks, i)
  inobj.InodeNumberToInode()
  count = self.FileObject.Write(i, inobj.inode.size, bytearray(string, 'utf-8'))
  if count == -1:
    print("Error: String could not be appended\n")
    return -1
  print("Bytes Written: " + str(count) + "\n")
  return 0
```

create filename: I have added a condition check for $filename$ not contain a "/", which will throw an error to user. Else, this will call FileName's create method with dir = *cwd*, name = $filename$, type =

INODE_TYPE_FILE. If the call to create fails to create (return value -1), user has to check the debug to find the error message.

ln target linkname: Here, we use the Link funtion of the file layer. Here, depending on the different error code, a different error message is populated in the command line.

Listing 3: Implementation of ln command

```
def ln(self, target, linkname):
  res = self.FileObject.Link(target,linkname,self.cwd)
  if res == -1 :
    print("Error: Specified name to link to is a directory type\n")
    return -1
  elif res == -2:
    print("Error: Specified name already exist in current directory\n")
    return -1
  elif res == -3:
    print("Error: Specified target is not a file\n")
    return -1
  return 0
```

ls: This was a minute change, just needed to change the if-else block where the the directories and files were getting printed to command line.

Listing 4: Implementation of ls command

```
def ls(self):
  inobj = InodeNumber(self.FileObject.RawBlocks, self.cwd)
  inobj.InodeNumberToInode()
  block_index = 0
  while block_index <= (inobj.inode.size // BLOCK_SIZE):
    block = self.FileObject.RawBlocks.Get(inobj.inode.block_numbers[block_index])
    if block_index == (inobj.inode.size // BLOCK_SIZE):
      end_position = inobj.inode.size % BLOCK_SIZE
    else:
      end_position = BLOCK_SIZE
    current_position = 0
    while current_position < end_position:
      entryname = block[current_position:current_position+MAX_FILENAME]
      entryinode = block[current_position+MAX_FILENAME:current_position+FILE_NAME_DIRENTRY_SIZE]
      entryinodenumber = int.from_bytes(entryinode, byteorder='big')
      inobj2 = InodeNumber(self.FileObject.RawBlocks, entryinodenumber)
      inobj2.InodeNumberToInode()
      if inobj2.inode.type == INODE_TYPE_DIR:
        print ("["+ str(inobj2.inode.refcnt) +"]:"+entryname.decode() + "/")
      else:
        print ("["+ str(inobj2.inode.refcnt) +"]:"+entryname.decode())
      current_position += FILE_NAME_DIRENTRY_SIZE
    block_index += 1
  return 0
```

I have added a line at the last of memoryfs_shell.py to store the information of every run. When we reran, it starts from where the file system was left off. This helped in testing.