Task 1: File System Calls

Introduction

Task 1 implements the file system calls creat, open, read, write, close, and unlink in UserProcess.java

Pseudocode – `handleCreat`

Method attempts to open a specified file and returns the new file descriptor used to access the file. If the file doesn't exist, it will create the file with the specified name. If there is an error, it will return -1

```
PROCEDURE handleCreat (fileDescriptor) :

      //argument validation
      if (fileDescriptor < 0 or > 15)
            return -1;

      //get filename from fileDescriptor and attempt to open file
      fileName = readVirtualMemoryString(fileDescriptor);
      openFile = UserKernel.filesystem.open(fileName, true);

      if (openFile success) {
            for (int i = 2; i < localArrayLength; ++i) {
                  if (no space)
                        return -1;
                  else if (file not in local table)
                        //put file into available space and return new
                          file descriptor
                  else if (file already in local table)
                        //return new file descriptor
            }
      }

      else if(openFile fail)
            return -1;

      if(create fail)
            return -1;

END PROCEDURE
```

Pseudocode – `handleOpen`

Method attempts to open a specified file and returns the new file descriptor used to access the file. If there is an error, it will return -1

```
PROCEDURE handleOpen (fileDescriptor) :

      //argument validation
      if (fileDescriptor < 0 or > 15)
            return -1;

      // get filename from fileDescriptor and attempt to open file
      fileName = readVirtualMemoryString(fileDescriptor);
      openFile = UserKernel.filesystem.open(fileName, false);

      if (openFile success) {
            //search local file table for the file
            for (int i = 2; i < localArrayLength; ++i) {
                  if (no space)
                        return -1;
                  else if (file not in local table)
                        //put file into available space and return new
                          file descriptor
                  else if (file already in local table)
                        //return new file descriptor
            }
      }

      else if(openFile fail)
            return -1;

END PROCEDURE
```

Pseudocode – `handleRead()`

Method attempts to read the specified bytes into the buffer from the file or stream referred to by the descriptor and returns the number of bytes read. If there is an error, it returns -1.

```
PROCEDURE handleRead (fileDescriptor, bufferAddr, bufferSize) :

     //argument validation
     if (fileDescriptor < 0 || > 15, bufferAddr < 0 || bufferSize < 0)
          return -1;

     if (bufferSize == 0) // no bytes to read
          return 0;

     //create array size bufferSize to store the bytes read
     stor = new byte[bufferSize];

     //read file and store into created array
     readBytes = file.read(stor, bufferSize);

     //error if readBytes < 0
     if (readBytes < 0)
          return -1;

     //write from array to virtual memory
     writtenBytes = writeVirtualMemory(bufferAddr, stor, readBytes);

     //return number of bytes read
     return readBytes;

END PROCEDURE
```

Pseudocode – `handleWrite()`

Method attempts to write the specified bytes into the buffer from the file or stream referred to by the descriptor and returns the number of bytes written. If there is an error, it returns -1.

```
PROCEDURE handleWrite (fileDescriptor, bufferAddr, bufferSize) :

      //argument validation
      if (fileDescriptor < 0 || > 15, bufferAddr < 0 || bufferSize < 0)
            return -1;

      //create temp array size bufferSize to store the bytes
      stor = new byte[bufferSize];

      //read from buffer and store into created array
      readBytes = readVirtualMemory(bufferAddr, stor, bufferSize);

      //error if readBytes < 0
      if (readBytes < 0)
            return -1;

      //write from array to file
      writtenBytes = file.write(stor, readBytes);

      if(writtenBytes < 0) //error if bytes written < 0
            return -1;
      else
            return writtenBytes; //return number of bytes written


END PROCEDURE
```

Pseudocode – `handleClose()`

Method closes the specified file descriptor so it no longer refers to any file or stream and resources allocated with the file are released. Returns 0 for success and -1 if it encounters an error.

```
PROCEDURE handleClose (fileDescriptor) :

      //argument validation
      if (fileDescriptor < 0 || > 15)
            return -1;

      //get filename from fileDescriptor
      fileName = readVirtualMemoryString(fileDescriptor);

      //Search the file table for file
      for (int i = 2; i < localArrayLength; ++i) {
            if(fileName not found)
                  return -1;
            else
                  // set value to null
      }

      boolean close = false;

      close = UserKernel.fileSystem.remove(fileName); //remove file

      if(close == false) //if file remove failed
            return -1;
      else
            return 0;
END PROCEDURE
```

Pseudocode – `handleUnlink()`

Method deletes a file from the file system. If the file is open, it waits until the last file descriptor reference is closed before deletion. If the file is closed, the space the file occupied becomes free and can be reused. Returns 0 for success and -1 if it encounters an error.

```
PROCEDURE handleUnlink (fileDescriptor) :

      //argument validation
      if (fileDescriptor < 0 || > 15)
            return -1;

      //get filename from fileDescriptor
      fileName = readVirtualMemoryString(fileDescriptor);

      if(no process has the file open) {
            //delete file and free resources
            return 0;
      }
      else if(process has the file open) {
            //wait until the last file descriptor reference is closed
            //delete file and free resources
            return 0;
      }
      else //error occurred
            return -1;


END PROCEDURE
```

Pseudocode – `handleHalt()`

Modified to ensure only the root process can call it. If another process tries to call it, it will be ignored and returned.

```
PROCEDURE handleHalt() :

      //if process is root
      if(processID == 0) {
            Machine.halt();
            return 0;
      }
      else
            return 1;


END PROCEDURE
```

Test Cases

Test 1 `handleCreat()`**:** Ensure that the file is opened properly if the file exists. If the file doesn't exist, it should be created without errors.

Test 2 `handleOpen()`**:** Ensure that the file is opened properly if it the file exists. Also to check that the method properly handles the 16 file limit and ensures it is not exceeded.

Test 3 `handRead()`**:** Ensure that it will read bytes from a file or exit gracefully if the file cannot be read or if there are invalid arguments.

Test 4 `handleWrite()`**:** Ensure that it will write the data to the file, or fails gracefully if given invalid arguments.

Test 5 `handleClose()`**:** Ensure that it will close the file descriptor for reuse and release the resources allocated to it.

Test 6 `handleUnlink()`**:** Ensure that it removes the files without errors. The file must be closed and the file descriptor must be free and resources associated to it released. There must be no remaining references to the file.

Test 7 `handleHalt()`**:** Ensure only root can invoke the method. Call with a non root process to return without running.