MANIPAL UNIVERSITY JAIPUR

Operating Systems Lab – AIM2231 (AY : 2024-25)

Upendra Singh
Assistant Professor
Department of AIML, School of Computer Science and Engineering
Manipal University Jaipur.

Week 4 : System Calls

Demonstrate the use of file system calls.

Outline

- What is System Calls
- Features of System Calls
- Creating, Writing, and Reading a File
- Appending Data to a File and using Its Metadata
- Key File System Calls
- Examples Programs
- Exercises

What is System Calls

- A system call is a programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on.
- A computer program makes a system call when it requests the operating system’s kernel.
- System call provides the services of the operating system to the user programs via the Application Program Interface(API).
- System calls are the only entry points into the kernel system and are executed in kernel mode.

Features of System Calls

- **Interface:** System calls provide a well-defined interface between user programs and the operating system.
- **Protection:** System calls are used to access privileged operations that are not available to normal user programs.
- **Kernel Mode:** When a system call is made, the program is temporarily switched from user mode to kernel mode.
- **Context Switching:** A system call requires a context switch, which involves saving the state of the current process and switching to the kernel mode to execute the requested service.

Features of System Calls

- **Error Handling:** System calls can return error codes to indicate problems with the requested service. Programs must check for these errors and handle them appropriately.
- **Synchronization:** System calls can be used to synchronize access to shared resources, such as files or network connections. The operating system provides synchronization mechanisms, such as locks or semaphores, to ensure that multiple programs can access these resources safely.

Creating, Writing, and Reading a File

1. Creating/Opening a File:

- `open("example.txt", O_CREAT | O_WRONLY, 0644)`:
  - If example.txt does not exist, it will create the file because of the O_CREAT flag.
  - Opens the file for writing only (`O_WRONLY`).
  - 0644 sets the file permissions (read/write for owner, read-only for others).
- If the file cannot be opened, `open() returns -1, and perror()` is called to display the error.

Creating, Writing, and Reading a File

2. Writing to the File:

- **`write(fd, writeData, sizeof(writeData))`**:
  - Writes the content of the **`writeData`** buffer into the file.
  - **`sizeof(writeData)`** specifies the number of bytes to write (length of the string in this case).
- If write() fails, it returns -1.

Creating, Writing, and Reading a File

3. **Closing the File**:

- `close(fd)` closes the file descriptor, ensuring that changes are saved and resources are freed.

4. **Reopening the File for Reading**:

- `open("example.txt", O_RDONLY)` opens the file for reading only `(O_RDONLY)`.
- If the file cannot be opened, `perror()` reports the error.

Creating, Writing, and Reading a File

5. Reading from the File:

- `read(fd, readBuffer, sizeof(readBuffer))` reads up to `sizeof(readBuffer)` bytes from the file and stores them in the `readBuffer`.

Appending Data to a File and Using Its Metadata

1. **Opening the File in Append Mode**:

- `open("example.txt", O_WRONLY | O_APPEND)`:
    - Opens the file for writing (O_WRONLY) and ensures that all writes go to the end of the file (O_APPEND).
- If the file does not exist or fails to open, `perror()` is called.

Appending Data to a File and Using Its Metadata

2. **Appending Data**:

- `write(fd, appendData, sizeof(appendData))` appends the content of `appendData` to the file.
- If write() fails, it returns -1.

3. **Closing the File**:

- `close(fd)` ensures the changes are saved.

Appending Data to a File and Using Its Metadata

4. **Retrieving File Information**:

- `stat("example.txt", &fileStat)`:
  - Gathers metadata about the file and stores it in the fileStat structure.
  - `fileStat.st_size` contains the file size in bytes.

Key File System Calls

1. open():

- Opens or creates a file and returns a file descriptor (a unique integer identifying the open file).
- Flags determine the file's behavior:
  - **O_CREAT**: Create the file if it doesn't exist.
  - **O_WRONLY**: Open the file for writing only.
  - **O_APPEND**: Ensure writes go to the end of the file.

2. write():

- Writes data from a buffer to the file.
- Requires the file descriptor, the data buffer, and the number of bytes to write.

Key File System Calls

3. read():

- Reads data from the file into a buffer.
- Requires the file descriptor, the buffer, and the number of bytes to read.

4. close():

- Closes the file descriptor, ensuring changes are flushed to the file and freeing system resources.

5. stat():

- Retrieves metadata about a file (e.g., size, permissions, last modified time).
- The st_size field in the stat structure stores the file's size in bytes.

Examples Programs

- Creating, Writing, and Reading a File
- Appending Data to a File and Displaying Its Size

Source Code Folder Link

https://mujcampus-my.sharepoint.com/:f:/g/personal/upendra_singh_jaipur_manipal_edu/EjL1pWMmp-tNjcaKHMTjfdYBJ_sIRyew-7CBaRQ08D5k6g?e=HLB5xg

Exercises:

1. Write a program to check if a file exists. If it exists, print "File found", otherwise create the file.
2. Write a program to copy the contents of one file into another using read() and write() system calls.
3. Modify the first program (`FileReadWrite.c`) to handle errors during each file operation (e.g., open(), read(), write()).

Thank you