README.md

AOS Assignment-1 Part-B

Implementation of a Loadable Kernel Module (LKM) for a Set with Concurrency Support

Bratin Mondal (21CS10016)
Datta Ksheeraj (21CS30037)

Contents:

21CS10016_21CS30037.c : kernel module implementation
Makefile: For loading the module and running tests on it
Tests: For testing the implemented kernel module

Execution Instructions

1. For building the kernel module use **make**
2. For installing or uninstalling the kernel module use **make install** or **make uninstall** accordingly
3. For running the various tests for testing purpose use **make tests**
4. For cleaning use **make clean**

Module Operations

Building the Kernel Module

When building the kernel module, the following steps occur:

- **Compilation**:
    - The make command compiles the source code into a kernel module binary (.ko file).
    - This involves invoking the kernel build system to process the source files and generate the module.
- **Object File Creation**:
    - During the build process, object files are created for the kernel module source code (21CS10016_21CS30037.c).
- **Module Binary:**
    - The output of the build process is a kernel module binary file (partb_21CS10016_21CS30037.ko) that can be loaded into the kernel.

Loading the Kernel Module

Upon loading the module using insmod the following events take place:

- Initialization **(LKM_Init function)**:
    - The module's initialization function is called.
    - A proc file named /proc/partb_21CS10016_21CS30037 is created.
    - A mutex (procFileMutex) is initialized to synchronize access to the proc file.
- Proc File Creation:
    - The proc file is set up to handle operations such as open, read, write, and release through the proc_fops structure.

Interacting with the Kernel Module

The proc file allows interaction through the following operations:

- Opening the Proc File (process_file_open function):
    - When a process opens the proc file, a process node is created and inserted into a Red-Black Tree (for logarithmic time complexity) if it does not already exist.
    - The process node keeps track of the process ID, associated set, state(set initialized/not), rb_node.
- Writing to the Proc File (process_file_write function):
    - Writing data to the proc file triggers the process_file_write function.
    - If the set is not initialized, the data is interpreted as the capacity for the set. The set is then initialized with this capacity.
    - If the set is already initialized, the data is treated as a value to be inserted into the set.
- Reading from the Proc File (process_file_read function):
    - Reading data from the proc file triggers the process_file_read function.
    - The function retrieves elements from the set and copies them to the user-space buffer.
- Releasing the Proc File (process_file_release function):
    - When a process closes the proc file, the corresponding process node is deleted from the Red-Black Tree (for logarithmic time complexity).
    - The associated set, if any, is also deleted.

Unloading the Kernel Module

When unloading the module using rmmod (make uninstall), the following events occur:

- Cleanup (LKM_Exit function):
    - The module's exit function is called.
    - All process nodes in the Red-Black Tree are deleted, and any associated sets are freed.
    - The process file is removed.
    - The mutex is destroyed.
    - An informational message is logged indicating that the module has been unloaded and resources have been cleaned up.
- Resource Deallocation:
    - All dynamically allocated memory (e.g., process nodes, sets) is freed to prevent memory leaks.

`Interaction with the LKM by user space process`

- **Opening the Process File**
    - File Path: /proc/partb_21CS10016_21CS30037
    - Mode: Read-write (0666)
    - Behavior:
        - The user-space process opens the proc file, initiating interaction with the LKM.
        - Each process receives its own unique set associated with the proc file.
- **Initializing the Set**
    - System Call: write()
    - Data: 1 byte (maximum capacity N of the set)
    - Behavior:
        - The user-space process writes a single byte containing the maximum capacity N of the set to the proc file.
        - Valid Range: 1 <= N <= 100

        - If N is within this range, the LKM initializes the set with the specified capacity.
        - If N is outside this range, the LKM produces an EINVAL error and leaves the set uninitialized.
- **Inserting Integers into the Set**
    - System Call: write()
    - Data: 4 bytes (32-bit integer)
    - Behavior:
        - The user-space process writes a 32-bit integer to the proc file.
        - Error Handling:
            - Invalid Argument: If the data is not a 32-bit integer, the LKM produces an EINVAL error.
            - Capacity Exceeded: If the set is full (number of elements reaches set capacity), the LKM produces an EACCES error and returns -EACCES.
        - On successful insertion, the LKM returns the number of bytes written (4 bytes).
- **Reading Integers from the Set**
    - System Call: read()
    - Data: 4 * N bytes (N 32-bit integers), where N is the number of elements in the set.
    - Behavior:
        - The user-space process reads an array of 32-bit integers from the proc file in sorted order.
        - Read Rules:
            - Element Retrieval: The integers are retrieved in ascending order.
            - Error Handling:
                - In case of error LKM produces an EACCES error and returns -EACCES.
        - On successful read, the LKM returns the number of bytes read (4 * N bytes).
        - In case the buffer is too small to hold all elements, the LKM copies as many elements as possible but returns the size of the buffer required to hold all elements, so that the user-space process can allocate a larger buffer and try again.
- **Closing the Proc File**
    - System Call: close()
    - Behavior:
        - The user-space process closes the proc file, terminating interaction with the current set.
        - The process can reopen the file to create and interact with a new set.
- **Concurrent Access**
    - Multiple processes can access the proc file simultaneously.
    - Each process will have its own unique set, managed independently by the LKM.