COMP 4511- System and Kernel Programming in Linux
Programming Assignment #4

Assignment 4 **System Call Implementation**

Concatenating several files into one is a common task that many users perform in the user level. For example,

```
$ cat file1 file2 file3 > newfile
```

Your task is to create a new system call that can concatenate multiple files into a target new file. The new system call interface is as follows:

```
long sys_xmerge(void* args, size_t argslen);
```

The usage of `void*` is a trick to bypass the limitation of syscall. The above prototype of the syscall allows to you to change the number and types of arguments. You would pass a `void*` pointer to the syscall, where all the arguments will be packed. `argslen` is the length of the `void*` buffer that the kernel should access.

Arguments (packed inside void*):

```
__user const char *outfile; // name of the output file

__user const char **infiles; // names of input files (Array)

unsigned int num_files; // number of input files

int oflags;              // the open flag for the outfile

mode_t mode; // permission mode for newly created outfile

// the number of files that have been read into outfile
__user int *ofile_count;
```

Note:

`__user` is used to mark a pointer as userspace. It is a flag to indicate that the pointer exists in userspace and that it should not be dereferenced. `copy_from_user` and `copy_to_user` should be used to handle the memory access. For details, please check the Appendix section.

Return value:

On success, the syscall should return the number of bytes successfully concatenated; Otherwise, it needs to return appropriate `-errno` on failure.

Implementation Details

Your system call should open each of the files listed in infiles in order, read their content, and then concatenate their content into the file named in outfile.

The newly created file should have a permission mode as defined in the mode parameter. See open(2) or create(2) for description of this parameter.
The oflags parameter should behave exactly as the flags parameter to the open(2) syscall. Please consult the man page for more info. You should support the following flags only (the only affect the outfile):

`O_APPEND, O_CREATE, O_TRUNC, and O_EXCL.`

Verifications

The most important thing is to ensure the validity of the input arguments. You must check for ALL possible bad conditions that could occur as the result of bad inputs to the syscall. In this case, the syscall should return the proper negative errno value (e.g., -EINVAL, -EPERM, -EACCESS, and etc). Consult the system errno table and pick the right error numbers for different conditions.

The possible kinds of errors of `sys_xmerge` are:

- missing arguments passed;
- null arguments;
- pointers to bad addresses;
- buffers and their declared lengths not matching;
- invalid flags;
- file(s) cannot be opened, read, written;

As the system call, you code must be efficient. For example, you might be better read in data in chunks that are native to the system this code is compiled on, the system page size (`PAGE_CACHE_SIZE` or `PAGE_SIZE`).

Cautions

- Allocate one page as a temporary buffer.
- Do not waste extra kernel memory (dynamic or stack) for the syscall.
- **Make sure no memory leak occurs**.

Test Program

Write a C program called `test_xmerge.c` that will call your syscall (see the lab example). You need to add `argc` and `argv` in the main function as parameters.The program should output some indication of success and uses perror() to print out information about what errors occurred. For example,

```
$ ./test_xmerge [flags] outfile infile infile2 ...
```

where flags is:

- -a: append mode (O APPEND)
- -c: O CREATE
- -t: O TRUNC
- -e: O EXCL
- -m ARG: set default mode to ARG (e.g., octal 755, see chmod(2) and umask(1))
- -h: print short usage string

Submission

- `xmerge.c`
  - For the implementation of syscall
  - You should place `xmerge.c` in `/home/oslab/linux-3.10.94/os_lab`
  - We assume that a 32-bit kernel is used
  - We assume that the syscall number is 355 (should be unused)
    - `355 i386 xmerge sys_xmerge`
- `test_xmerge.c`
  - For testing the syscall

Appendix

- The following user space memory access API may be useful for this assignment. You are not required to use all of the following functions

Function	Description
`access_ok`	Check the validity of the user space memory pointer
`get_user`	Gets a simple variable from user space
`put_user`	Puts a simple variable to user space
`clear_user`	Clears, or zeros, a block in user space
`copy_to_user`	Copies a block of data from the kernel to user space
`copy_from_user`	Copies a block of data from user space to the kernel
`strnlen_user`	Gets the size of a string buffer in user space
`strncpy_from_user`	Copies a string from user space to the kernel

- You can also call other syscalls. For example: `sys_open`, `sys_read`, and `sys_close` are useful syscalls for file I/O