Recitation 5

pthread_join()

- **pthread_join()** will suspend execution of the thread that has called it unless the target thread terminates
  - This is useful in cases when you want to wait for thread/s to terminate before further processing in main thread.

Code Demo

Project 2

Overview

- Add your own system calls by modifying some files in the kernel
- Test the system calls that you added

First, Set up the kernel source

- Before you do anything,
  - Make sure you set up the kernel source
    - This will likely take more than an hour!

Adding the system calls

1. Modify *linux-5.10.140/kernel/sys.c*

- Will contain actual implementation of your syscalls.
- *asmlinkage long sys_cs1550_send_msg(const char __user *to, const char __user *msg, const char __user *from)*
  - returns 0 for success and a negative number for error
- Use the SYSCALL_DEFINEn() macro to declare a system call with n parameters

1. Modify *linux-5.10.140/kernel/sys.c*

- *asmlinkage long sys_cs1550_get_msg(const char __user *to, char __user *msg, char __user *from)*
  - The recipient information is passed as input
  - The message and the sender are copied to the 2nd and 3rd parameter respectively
    - Copies only one message at a time
  - return values
    - 1 to indicate that there are more messages to return
    - 0 to indicate that this is the last message
    - -1 to indicate that there are no messages
    - Any other negative number to indicate an error

- Use the SYSCALL_DEFINEn() macro to declare a system call with n parameters

1. Modify *linux-5.10.140/kernel/sys.c*

- Keep track of the messages using a linked list allocated in kernel space
  - Use kmalloc() and kfree()
    - Use GFP_KERNEL for the mode
    - Allocated node to be freed after the message is copied to the recipient user

2. Modify *include/linux/syscalls.h*

- Add full prototypes of the two new system calls, without using the DEFINE macro from sys.c

3. Modify *include/uapi/asm-generic/unistd.h*

- Add system calls numbered 441 and 442.
- Increase __NR_syscalls to 443

4. Modify *arch/x86/entry/syscalls/syscall_64.tbl*

```
441     common    cs1550_send_msg     sys_cs1550_send_msg
442     common    cs1550_get_msg      sys_cs1550_get_msg
```

5. Modify *arch/x86/entry/syscalls/syscall_32.tbl*

```
441    i386      cs1550_send_msg    sys_cs1550_send_msg
442    i386      cs1550_get_msg     sys_cs1550_get_msg
```

Now, do the following

- Rebuild the Kernel
- Install the rebuilt kernel
- After this, you may only change the sys.c file as per need
  - Building kernel will be quicker when only sys.c is changed

osmsg

- Code to test the system calls you added

- You can call a system call like
  - syscall(441, to, msg, from);
    - to = name of the recipient user
    - msg = message to be sent
    - from = sending user name

How to test?

- Login with a username (eg. *user*) and password
  - Execute osmsg with –s flag for sending a message to another user (eg. *user2*)
  - You can create a shell script to send multiple messages at once
- Switch to another user (eg. *user2*) using *su* command
  - *su user2* will switch to *user2*
  - Use command *passwd* (as root) to change the password of the users
  - Execute osmsg with –r flag to read the messages that were sent to *user2*
- Use the linux command *useradd* to add more users to your system

Important points

- Use printk() only for debugging purposes in the kernel
  - Make sure to remove these in the final submission
- You can get the current user’s name using getenv()
- Make sure you go through the project description pdf thoroughly