Lab – 04

Objectives:

1. Understanding the concept of Inter Process Communication (IPC)

2. Understanding the concept of Signals and their working

3. Understanding the concepts of Threads

Resources:

1. Video Lecture 09: https://youtu.be/q9lsekGzh0A?list=PL7B2bn3GwfBuJWtHADcXC44piWLRzr8

2. Video Lecture 10: https://youtu.be/lKSNjpGxt5o?list=PL7B2bn3GwfBuJWtHADcXC44piWLRzr8

3. Lecture Slides: https://www.arifbutt.me/wp-content/uploads/2024/08/Lecture-08-ThreadManagement.pdf

---

Task 1:

I. What is the purpose of interprocess communication (IPC) in Unix systems?

Interprocess Communication (IPC) enables processes to communicate and synchronize their actions. Its primary purposes include:

1. Data Sharing: Allows processes to exchange information.
2. Synchronization: Coordinates execution to prevent conflicts in resource access.
3. Resource Management: Facilitates efficient allocation of shared resources.
4. Distributed Communication: Enables processes on different machines to communicate.

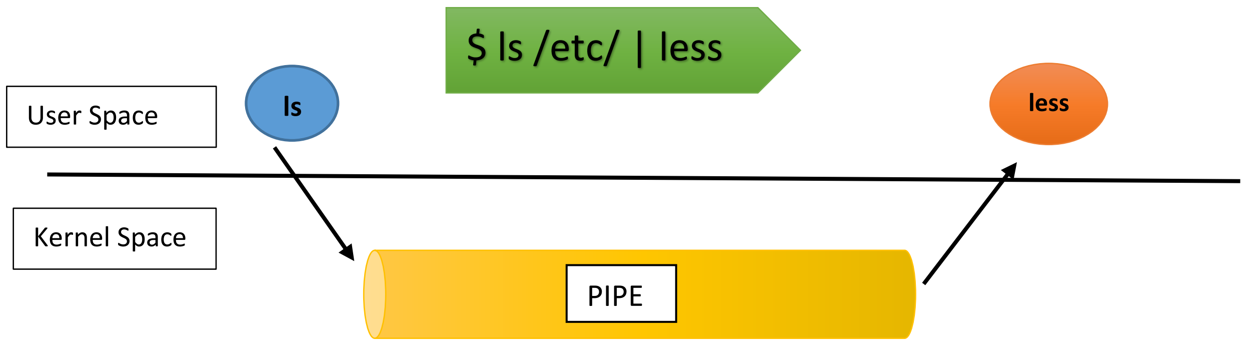
Common IPC mechanisms include pipes, FIFOs, shared memory, and sockets.

II. Explain the difference between pipes and FIFOs (named pipes).

Pipes are unnamed, temporary communication channels accessible only by the creating processes and exist only while those processes are running. In contrast, FIFOs are named, persistent, and visible in the filesystem, allowing any process to access them. Pipes enable one-way communication between related processes, while FIFOs facilitate one-way communication between unrelated processes. Furthermore, access to pipes is limited to parent-child relationships, whereas any process with the necessary permissions can access FIFOs.

III. Draw a diagram to explain the working of pipes.

- **Process A** writes data to the pipe, which **Process B** reads.
- Pipes provide unidirectional communication between processes.



Task 2: Write a command which create a named pipe with a name "fifo1". Use this pipe to transfer some data from one process on one terminal to other process in any other terminal and display that data on screen.

- Create the named pipe: **mkfifo fifo1**

Creates a named pipe called fifo1 for interprocess communication.

- Send data to the pipe (in Terminal 1): **echo "Hello from Terminal 1" > fifo1**

Purpose: Sends the message "Hello from Terminal 1" into the named pipe fifo1.

- Read data from the pipe (in Terminal 2): **cat fifo1**

Reads and displays the message from the named pipe fifo1 on the screen.

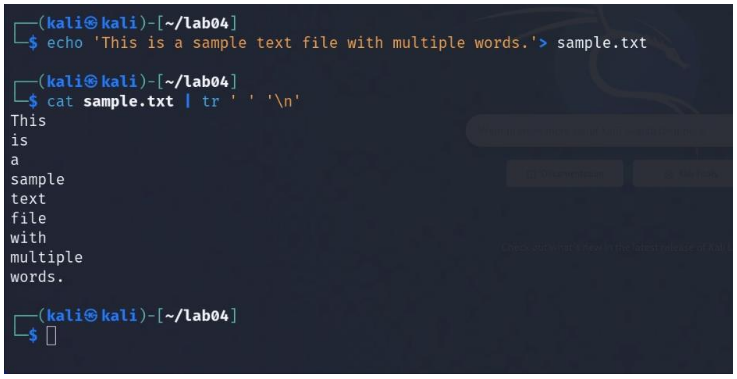
Task 3: Create a file that contains the text "This is, yes really! a text with?&* too many str$ange# characters ;-)". Now write a set of command(s) that prints the above but do not print non-letters from above text. (Hint: use pipe)

 Create a File**:** Creates a file named textfile.txt with the specified text.

echo "This is, yes really! a text with?&* too many str$ange# characters ;-)" > textfile.txt

 Print Only Letters: Extracts sequences of letters from textfile.txt and removes newline characters, outputting the result on a single line. **grep -o '[a-zA-Z]*' textfile.txt | tr -d '\n'**

Task 4: Write set of command(s) that receives a text file, and outputs all words on a separate line



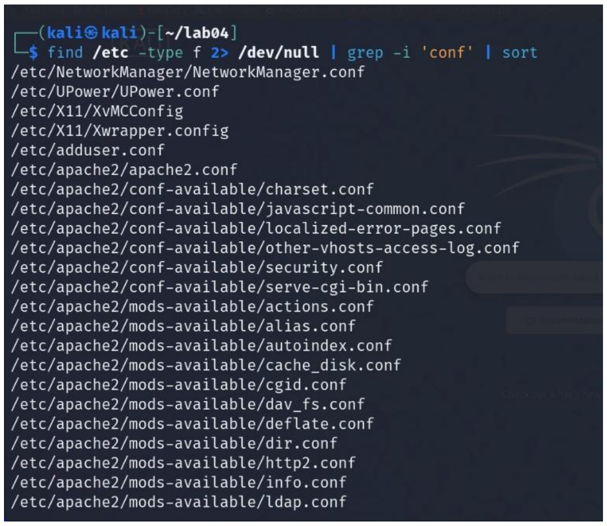
```
┌──(kali㉿kali)-[~/lab04]
└─$ echo 'This is a sample text file with multiple words.'> sample.txt

┌──(kali㉿kali)-[~/lab04]
└─$ cat sample.txt | tr ' ' '\n'
This
is
a
sample
text
file
with
multiple
words.

┌──(kali㉿kali)-[~/lab04]
└─$ 
```

echo "This is a sample text file with multiple words." > sample.txt

cat sample.txt | tr ' '   '\n'

Task 5: Write down set of command(s) to make a sorted list of all files in /etc that contain the case insensitive string conf in their file name.

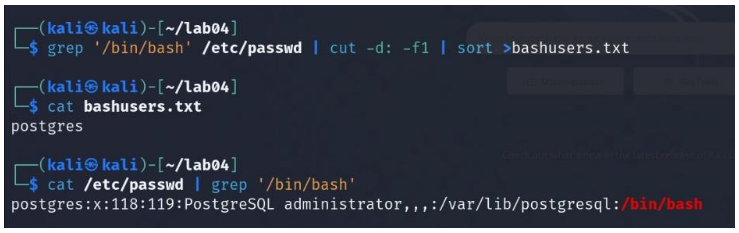
find /etc -type f | grep -i 'conf' | sort



```
┌──(kali㉿kali)-[~/lab04]
└─$ find /etc -type f 2> /dev/null | grep -i 'conf' | sort
/etc/NetworkManager/NetworkManager.conf
/etc/UPower/UPower.conf
/etc/X11/XvMCConfig
/etc/X11/Xwrapper.config
/etc/adduser.conf
/etc/apache2/apache2.conf
/etc/apache2/conf-available/charset.conf
/etc/apache2/conf-available/javascript-common.conf
/etc/apache2/conf-available/localized-error-pages.conf
/etc/apache2/conf-available/other-vhosts-access-log.conf
/etc/apache2/conf-available/security.conf
/etc/apache2/conf-available/serve-cgi-bin.conf
/etc/apache2/mods-available/actions.conf
/etc/apache2/mods-available/alias.conf
/etc/apache2/mods-available/autoindex.conf
/etc/apache2/mods-available/cache_disk.conf
/etc/apache2/mods-available/cgid.conf
/etc/apache2/mods-available/dav_fs.conf
/etc/apache2/mods-available/deflate.conf
/etc/apache2/mods-available/dir.conf
/etc/apache2/mods-available/http2.conf
/etc/apache2/mods-available/info.conf
/etc/apache2/mods-available/ldap.conf
```

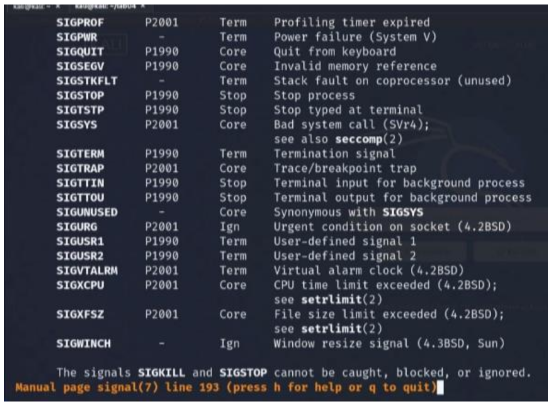
Task 6: Write down set of command(s) to put a sorted list of all bash users in bashusers.txt

The command **grep '/bin/bash' /etc/passwd | cut -d: -f1 | sort > bashusers.txt** extracts usernames from the /etc/passwd file where the shell is set to Bash, using grep to filter lines and cut to extract the first field (username) based on the colon delimiter. The sorted list of usernames is then saved to bashusers.txt.

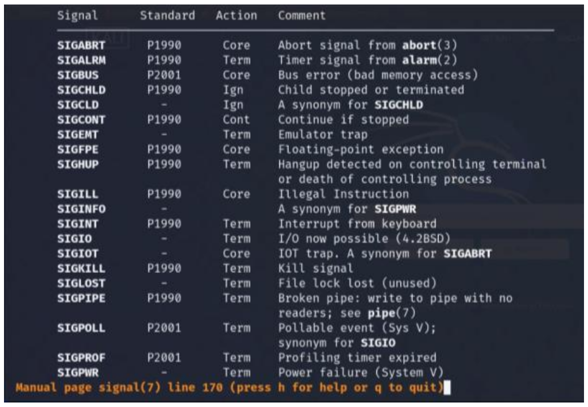


```
┌──(kali㉿kali)-[~/lab04]
└─$ grep '/bin/bash' /etc/passwd | cut -d: -f1 | sort >bashusers.txt

┌──(kali㉿kali)-[~/lab04]
└─$ cat bashusers.txt
postgres

┌──(kali㉿kali)-[~/lab04]
└─$ cat /etc/passwd | grep '/bin/bash'
postgres:x:118:119:PostgreSQL administrator,,,:/var/lib/postgresql:/bin/bash
```

Task 7: What signals user can send to a process from keyboard give their name, number and their default actions.



```
       SIGPROF      P2001     Term    Profiling timer expired
       SIGPWR         -       Term    Power failure (System V)
       SIGQUIT      P1990     Core    Quit from keyboard
       SIGSEGV      P1990     Core    Invalid memory reference
       SIGSTKFLT      -       Term    Stack fault on coprocessor (unused)
       SIGSTOP      P1990     Stop    Stop process
       SIGTSTP      P1990     Stop    Stop typed at terminal
       SIGSYS       P2001     Core    Bad system call (SVr4);
                                      see also seccomp(2)
       SIGTERM      P1990     Term    Termination signal
       SIGTRAP      P2001     Core    Trace/breakpoint trap
       SIGTTIN      P1990     Stop    Terminal input for background process
       SIGTTOU      P1990     Stop    Terminal output for background process
       SIGUNUSED      -       Core    Synonymous with SIGSYS
       SIGURG       P2001     Ign     Urgent condition on socket (4.2BSD)
       SIGUSR1      P1990     Term    User-defined signal 1
       SIGUSR2      P1990     Term    User-defined signal 2
       SIGVTALRM    P2001     Term    Virtual alarm clock (4.2BSD)
       SIGXCPU      P2001     Core    CPU time limit exceeded (4.2BSD);
                                      see setrlimit(2)
       SIGXFSZ      P2001     Core    File size limit exceeded (4.2BSD);
                                      see setrlimit(2)
       SIGWINCH       -       Ign     Window resize signal (4.3BSD, Sun)

       The signals SIGKILL and SIGSTOP cannot be caught, blocked, or ignored.
 Manual page signal(7) line 193 (press h for help or q to quit)
```



```
       Signal      Standard   Action   Comment
       ──────────────────────────────────────────────────────────────
       SIGABRT      P1990      Core    Abort signal from abort(3)
       SIGALRM      P1990      Term    Timer signal from alarm(2)
       SIGBUS       P2001      Core    Bus error (bad memory access)
       SIGCHLD      P1990      Ign     Child stopped or terminated
       SIGCLD         -        Ign     A synonym for SIGCHLD
       SIGCONT      P1990      Cont    Continue if stopped
       SIGEMT         -        Term    Emulator trap
       SIGFPE       P1990      Core    Floating-point exception
       SIGHUP       P1990      Term    Hangup detected on controlling terminal
                                       or death of controlling process
       SIGILL       P1990      Core    Illegal Instruction
       SIGINFO        -                A synonym for SIGPWR
       SIGINT       P1990      Term    Interrupt from keyboard
       SIGIO          -        Term    I/O now possible (4.2BSD)
       SIGIOT         -        Core    IOT trap. A synonym for SIGABRT
       SIGKILL      P1990      Term    Kill signal
       SIGLOST        -        Term    File lock lost (unused)
       SIGPIPE      P1990      Term    Broken pipe: write to pipe with no
                                       readers; see pipe(7)
       SIGPOLL      P2001      Term    Pollable event (Sys V);
                                       synonym for SIGIO
       SIGPROF      P2001      Term    Profiling timer expired
       SIGPWR         -        Term    Power failure (System V)
 Manual page signal(7) line 170 (press h for help or q to quit)
```

Common Keyboard Signals

Signal Name	Signal Number	Default Action
SIGINT	2	Terminate the process (generated by Ctrl + C)
SIGQUIT	3	Terminate the process and create a core dump (generated by Ctrl + \)
SIGHUP	1	Terminate the process or reload configuration (often sent when a terminal closes)
SIGTERM	15	Terminate the process gracefully (default signal for kill)
SIGKILL	9	Forcefully terminate the process (cannot be caught or ignored)
SIGSTOP	19	Stop (pause) the process (cannot be caught or ignored)

Brief Explanation of Each Signal

- **SIGINT**: Sent when the user types Ctrl + C. The process is terminated immediately.
- **SIGQUIT**: Sent when the user types Ctrl + \. This signal also terminates the process but creates a core dump for debugging purposes.
- **SIGHUP**: Typically sent when a terminal connection is lost. It can also be used to instruct processes to reload their configuration files.
- **SIGTERM**: This is a polite way to ask a process to terminate, allowing it to perform cleanup operations before exiting.
- **SIGKILL**: This signal immediately stops the process without cleanup. It cannot be handled or ignored by the process.
- **SIGSTOP**: This signal pauses the process, which can be resumed later with the SIGCONT signal.

Task 8: What is the difference b/w "$ kill pid" and "$ kill -15 pid." What does "$ kill -9 pid" will do? (Note: pid could be any process id).

$ kill pid: Sends the default SIGTERM signal (15) to the specified process, requesting it to terminate gracefully, allowing cleanup operations.
$ kill -15 pid: Explicitly sends the SIGTERM signal to the process, with the same effect as $ kill pid; the process can choose to handle or ignore it.
$ kill -9 pid: Sends the SIGKILL signal (9) to the process, forcefully terminating it immediately without cleanup or the ability to ignore the signal.

Task 9: See the man page and mention the different types of dispositions a signal can have by giving two examples signals for each.

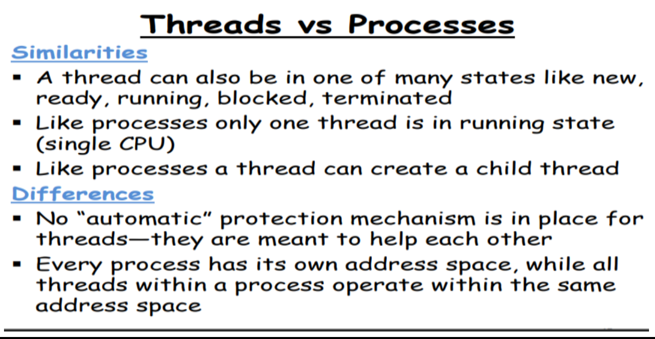
In Unix-like systems, signals can have different dispositions that determine how a process responds when it receives a signal.

1. **Terminate (Term)**:
   - **Examples**:
     - **SIGINT**: Interrupt from keyboard (Ctrl+C).
     - **SIGHUP**: Hangup detected on controlling terminal.
2. **Ignore (Ign)**:
   - **Examples**:
     - **SIGCHLD**: Child process stopped or terminated.
     - **SIGURG**: Urgent condition on socket.
3. **Core Dump (Core)**:
   - **Examples**:
     - **SIGABRT**: Abort signal from abort(3).
     - **SIGSEGV**: Invalid memory reference.
4. **Stop (Stop)**:
   - **Examples**:
     - **SIGSTOP**: Stop process (cannot be caught).
     - **SIGTSTP**: Stop typed at terminal (Ctrl+Z).
5. **Continue (Cont)**:
   - **Examples**:
     - **SIGCONT**: Continue if stopped.
     - **SIGTTIN**: Terminal input for background process.

Task 10: What are similarities and differences in a process and a thread? Give 2 use cases of muti-threading.

Use Cases of Multi-threading:

1. **Web Servers**: Handling multiple client requests simultaneously, allowing for concurrent processing and improved performance.
2. **Real-time Data Processing**: Applications that require continuous data input (e.g., stock trading platforms) can use multi-threading to process incoming data and perform calculations in parallel.



Threads vs Processes

Similarities
- A thread can also be in one of many states like new, ready, running, blocked, terminated
- Like processes only one thread is in running state (single CPU)
- Like processes a thread can create a child thread

Differences
- No "automatic" protection mechanism is in place for threads—they are meant to help each other
- Every process has its own address space, while all threads within a process operate within the same address space

---

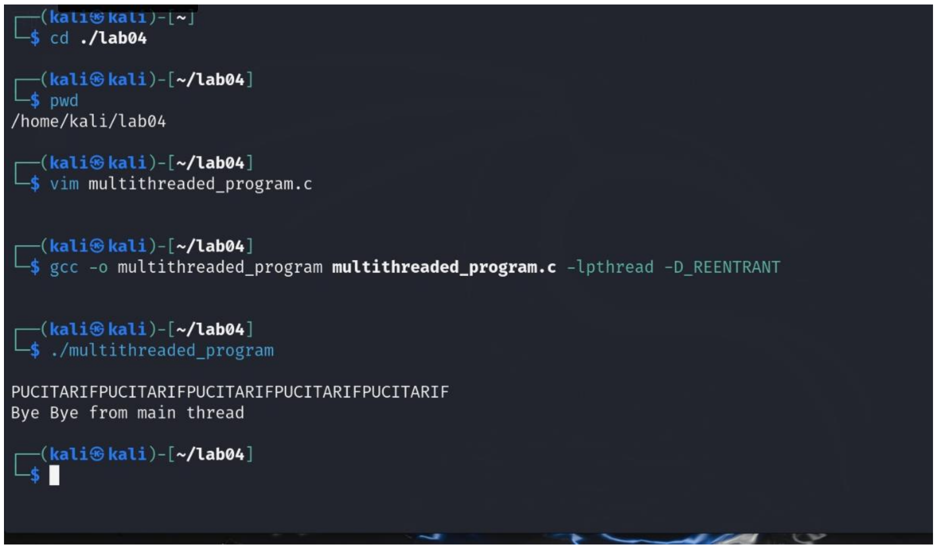
Task 11: What is difference between kernel and user level threads?

1. **Kernel-Level Threads**:
   - Managed by the operating system kernel, which can schedule and manage threads directly.
   - Allows the OS to perform better scheduling and utilize multiple processors effectively.
   - Example: If one kernel thread is blocked (e.g., waiting for I/O), the kernel can schedule another thread from the same process to run.
2. **User-Level Threads**:
   - Managed by user-level libraries, not the operating system; the kernel is unaware of their existence.
   - Faster context switching since the kernel does not need to be involved in thread management.
   - Example: If a user-level thread blocks, the entire process can become unresponsive, as the kernel cannot schedule another thread from that process.

Summary

Kernel-level threads provide better concurrency and allow better use of multi-core systems, while user-level threads offer quicker context switching and are more lightweight.

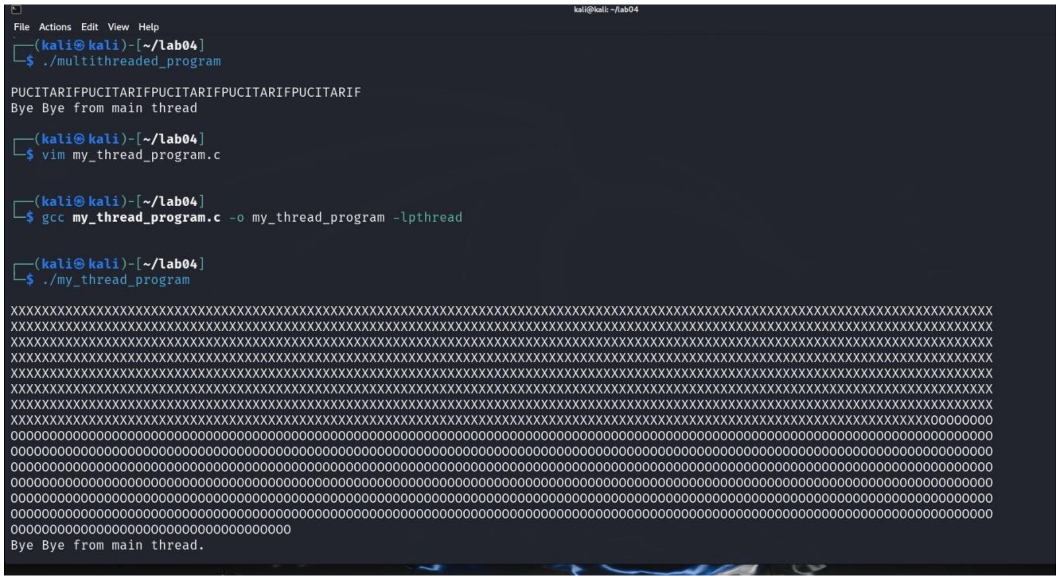
Task 12: Compile and execute the following code. Explain the output. And what are the advantages of compiling multithreaded programs using the –D_REENTRANT flag?



```
┌──(kali㉿kali)-[~]
└─$ cd ./lab04

┌──(kali㉿kali)-[~/lab04]
└─$ pwd
/home/kali/lab04

┌──(kali㉿kali)-[~/lab04]
└─$ vim multithreaded_program.c


┌──(kali㉿kali)-[~/lab04]
└─$ gcc -o multithreaded_program multithreaded_program.c -lpthread -D_REENTRANT


┌──(kali㉿kali)-[~/lab04]
└─$ ./multithreaded_program

PUCITARIFPUCITARIFPUCITARIFPUCITARIFPUCITARIF
Bye Bye from main thread

┌──(kali㉿kali)-[~/lab04]
└─$ 
```

Compiling with the -D_REENTRANT flag makes your multithreaded programs safer and more reliable by ensuring functions can run safely across multiple threads. It also helps avoid bugs and ensures compatibility with standard libraries, making the code easier to debug and portable across different systems.

Task 13: What will be the output of the following code snippet?



```
┌──(kali㉿kali)-[~/lab04]
└─$ ./multithreaded_program

PUCITARIFPUCITARIFPUCITARIFPUCITARIFPUCITARIF
Bye Bye from main thread

┌──(kali㉿kali)-[~/lab04]
└─$ vim my_thread_program.c


┌──(kali㉿kali)-[~/lab04]
└─$ gcc my_thread_program.c -o my_thread_program -lpthread


┌──(kali㉿kali)-[~/lab04]
└─$ ./my_thread_program

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXOOOOOOOO
OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
Bye Bye from main thread.
```