Mawlana Bhashani Science and Technology University

Lab-Report

Report No: 03
Course code: ICT-3108
Course title: Operating System Lab
Date of Performance:10-09-20
Date of Submission:14-09-20

Submitted by

Name: Md.Abdullah Al Mamun
ID:IT-18040
3th year 1ndsemester
Session: 2017-2018
Dept. of ICT
MBSTU.

Submitted To

Nazrul Islam
Assistant Professor
Dept. of ICT
MBSTU.

Lab Report NO : 03

Lab Report Name : Threads on Operating System.

Theory

1. What is Thread?

A thread is a flow of execution through the process code, with its own program counter that keeps track of which instruction to execute next, system registers which hold its current working variables, and a stack which contains the execution history.
A thread shares with its peer threads few information like code segment, data segment and open files. When one thread alters a code segment memory item, all other threads see that.
A thread is also called a lightweight process. Threads provide a way to improve application performance through parallelism. Threads represent a software approach to improving performance of operating system by reducing the overhead thread is equivalent to a classical process.
Each thread belongs to exactly one process and no thread can exist outside a process. Each thread represents a separate flow of control. Threads have been successfully used in implementing network servers and web server. They also provide a suitable foundation for parallel execution of applications on shared memory multiprocessors. The following figure shows the working of a single-threaded and a multithreaded process.

2.Types of Thread

Answer:

Threads are implemented in following two ways −
User Level Threads − User managed threads.
Kernel Level Threads − Operating System managed threads acting on kernel, an operating system core.
User Level Threads
In this case, the thread management kernel is not aware of the existence

of threads. The thread library contains code for creating and destroying threads, for passing message and data between threads, for schedulingthread execution and for saving and restoring thread contexts. The application starts with a single thread.

Kernel Level Threads

In this case, thread management is done by the Kernel. There is no thread management code in the application area. Kernel threads are supported directly by the operating system. Any application can be programmed to be multithreaded. All of the threads within an application are supported within a single process.

The Kernel maintains context information for the process as a whole and for individuals threads within the process. Scheduling by the Kernel is done on a thread basis. The Kernel performs thread creation, scheduling and management in Kernel space. Kernel threads are generally slower to create and manage than the user threads.

3. Implemantation of threat

Answer:

The implementation of threads an proccess differs between operating systems, but in most cases a thread is a component of a process. Multiple threads can exist within one process, executing concurrently and sharing resources such as memory, while different processes do not share these resources. In particular, the threads of a process share its executable code and the values of its dynamically allocated variables and non-thread-local global variables at any given time.Process

Thread #2

Thread #1

fig:

A process with two threads of execution, running on one processor

kernel-level threading

Threads created by the user in a 1:1 correspondence with schedulable entities in the kernel[10]re the simplest possible threading implementation. OS/2 and Win32 used this approach from the start, while on Linux the usual C library implements this approach (via the NPTL or older LinuxThreads). This approach is also used by Solaris, NetBSD, FreeBSD, macOS, and iOS.user-level threading

An model implies that all application-level threads map to one kernel-level scheduled entity;[10] the kernel has no knowledge of the application threads. With this approach, context switching can be done very quickly and, in addition, it can be implemented even on simple kernels which do not support threading. One of the major drawbacks, however, is that it cannot benefit from the hardware acceleration on multithreaded processors or multi-processor computers: there is never more than one thread being scheduled at the same time.[10] For example: If one of the threads needs to execute an I/O request, the whole process is blocked and the threading advantage cannot be used. The GNU Portable Threads uses User-level threading, as does State Threads.
hybrid threading
M:N maps some M number of application threads onto some N number of kernel entities,[10] or "virtual processors." This is a compromise between kernel-level ("1:1") and user-level ("N:1") threading. In general, "M:N" threading systems are more complex to implement than either kernel or user threads, because changes to both kernel and user-space code are required[clarification needed]. In the M:N implementation, the threading library is responsible for scheduling user threads on the available schedulable entities; this makes context switching of threads very fast, as it avoids system calls. However, this increases complexity and the likelihood of priority inversion, as well as suboptimal scheduling without extensive (and expensive) coordination between the userland scheduler and the kernel scheduler.

Impementation of Thread

```
class ThreadDemo implements Runnable {Thread t;
ThreadDemo() {
t = new Thread(this, "Thread");
System.out.println("Child thread: " + t);
t.start();
}
public void run() {
try {
System.out.println("Child Thread");
thread.sleep(50);
} catch (InterruptedException e) {
```

```
System.out.println("The child thread is interrupted.");
}
System.out.println("Exiting the child thread");
}
}
public class Demo {
public static void main(String args[]) {
new ThreadDemo();try {
System.out.println("Main Thread");
Thread.sleep(100);
} catch (InterruptedException e) {
System.out.println("The Main thread is interrupted");
}
System.out.println("Exiting the Main thread");
}
}
```

Output

```
Child thread: Thread[Thread,5,main]
Main Thread
Child Thread
Exiting the child thread
Exiting the Main thread
```

ET2504 Computer Networking Software Tools

Introduction to lab environment

Discussion :

The purpose of this laboratory is to provide an introduction to GNU/Linux environment and to get in
practice with the basic commands that we use in GNU/Linux system.
Before you start remember 'man' command is our friend, which provides in depth information for the
topic or command typed in. 'man' pages are very helpful while learning GNU/Linux.
usage: $man <command>
For example: $man man

Start the Terminal (Applications>Accessories>Terminal)

1. Issue the following basic commands and observe the output

$ whoami

It shows who logged on this system

$ hostname It tells on which machine you are

$ pwd

shows the path of current working directory

$ ls

displays the list of files in the current working directory

2. Working with text editors

Text editors are essential items to handle many important tasks, for example to edit the configuration files, programming language source code and to create README files etc. GNU/Linux supports many number of text editors like Kate, Geany, emacs, vi, pico , Gedit, Kwrite etc. some of them are pre installed, for example “vi”.

Exercise: Use pico editor and create a text file with name 'README.txt' in your 'home' directory, enter some text and exit by saving. Now display the content of that file on screen.

3. Working with files and directories

These are the basic commands that are useful when working with files and directories.

$ mkdir <dir_name>

creates a directory with specified 'dir_name'

$ cd <dir_name>

switches to 'dir_name' directory

$ cd ..

moves one directory up

$ cd ../ ../

moves two directories up (and so on)

$ cd

brings you to the highest level of your home directory

$ rmdir <dir_name>

removes entire directory

$ rm <file_name>

removes file name

$ rm r < dir_name>

removes directory including it's contents

$ mv <name1> <name2>

renames the directories or files

$ mv <name> <path>

moves files/directories to the specified path

$ cp <name> <path>
copies file/directory as specified in path