Thread Scheduling

Last Updated : 22 Sep, 2023

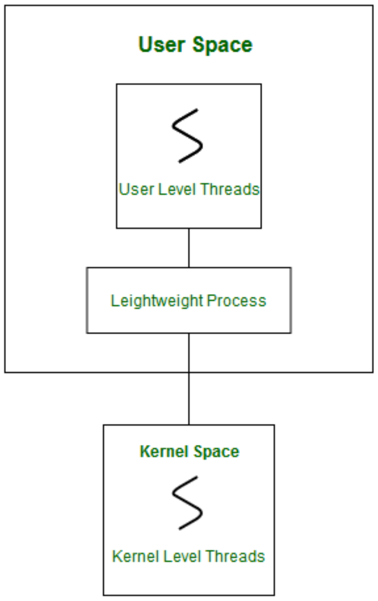
There is a component in Java that basically decides which thread should execute or get a resource in the operating system.

Scheduling of threads involves two boundary scheduling.

1. Scheduling of user-level threads (ULT) to kernel-level threads (KLT) via lightweight process (LWP) by the application developer.
2. Scheduling of kernel-level threads by the system scheduler to perform different unique OS functions.

Lightweight Process (LWP)

Light-weight process are threads in the user space that acts as an interface for the ULT to access the physical CPU resources. Thread library schedules which thread of a process to run on which LWP and how long. The number of LWPs created by the thread library depends on the type of application. In the case of an I/O bound application, the number of LWPs depends on the number of user-level threads. This is because when an LWP is blocked on an I/O operation, then to invoke the other ULT the thread library needs to create and schedule another LWP. Thus, in an I/O bound application, the number of LWP is equal to the number of the ULT. In the case of a CPU-bound application, it depends only on the application. Each LWP is attached to a separate kernel-level thread.



In real-time, the first boundary of thread scheduling is beyond specifying the scheduling policy and the priority. It requires two controls to be specified for the User level threads: Contention scope, and Allocation domain. These are explained as following below.

Contention Scope

The word contention here refers to the competition or fight among the User level threads to access the kernel resources. Thus, this control defines the extent to which contention takes place. It is defined by the application developer using the thread library.

Depending upon the extent of contention it is classified as-

- **Process Contention Scope (PCS) :**
  The contention takes place among threads **within a same process**.
  The thread library schedules the high-prioritized PCS thread to access the resources via available LWPs (priority as specified by the application developer during thread creation).
- **System Contention Scope (SCS) :**
  The contention takes place among **all threads in the system**. In this case, every SCS thread is associated to each LWP by the thread library and are scheduled by the system scheduler to access the kernel resources.
  In LINUX and UNIX operating systems, the POSIX Pthread library provides a function *Pthread_attr_setscope* to define the type of contention scope for a thread during its creation.

```
int Pthread_attr_setscope(pthread_attr_t *attr, int scope)
```

The first parameter denotes to which thread within the process the scope is defined.
The second parameter defines the scope of contention for the thread pointed. It takes two values.

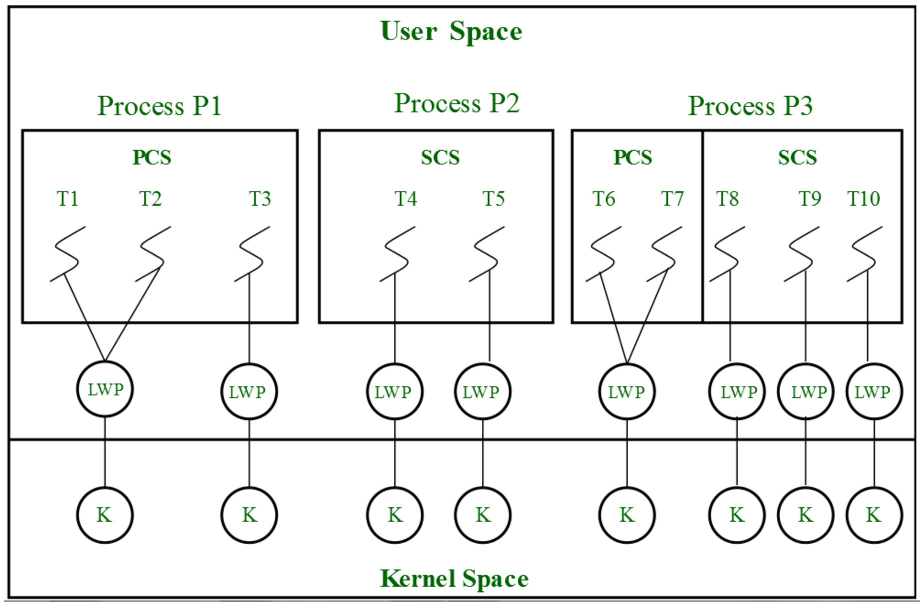
```
PTHREAD_SCOPE_SYSTEM
PTHREAD_SCOPE_PROCESS
```

If the scope value specified is not supported by the system, then the function returns *ENOTSUP*.

Allocation Domain

The allocation domain is **a set of one or more resources** for which a thread is competing. In a multicore system, there may be one or more allocation domains where each consists of one or more cores. One ULT can be a part of one or more allocation domain. Due to this high complexity in dealing with hardware and software architectural interfaces, this control is not specified. But by default, the multicore system will have an interface that affects the allocation domain of a thread.

Consider a scenario, an operating system with three process P1, P2, P3 and 10 user level threads (T1 to T10) with a single allocation domain. 100% of CPU resources will be distributed among all the three processes. The amount of CPU resources allocated to each process and to each thread depends on the contention scope, scheduling policy and priority of each thread defined by the application developer using thread library and also depends on the system scheduler. These User level threads are of a different contention scope.



In this case, the contention for allocation domain takes place as follows:

Process P1

All PCS threads T1, T2, T3 of Process P1 will compete among themselves. The PCS threads of the same process can share one or more LWP. T1 and T2 share an LWP and T3 are allocated to a separate LWP. Between T1 and T2 allocation of kernel resources via LWP is based on preemptive priority scheduling by the thread library. A Thread with a high priority will preempt low priority threads. Whereas, thread T1 of process p1 cannot preempt thread T3 of process p3 even if the priority of T1 is greater than the priority of T3. If the priority is equal, then the allocation of ULT to available LWPs is based on the scheduling policy of threads by the system scheduler(not by thread library, in this case).

Process P2

Both SCS threads T4 and T5 of process P2 will compete with processes P1 as a whole and with SCS threads T8, T9, T10 of process P3. The system scheduler will schedule the kernel resources among P1, T4, T5, T8, T9, T10, and PCS threads (T6, T7) of process P3 considering each as a separate process. Here, the Thread library has no control of scheduling the ULT to the kernel resources.

Process P3

Combination of PCS and SCS threads. Consider if the system scheduler allocates 50% of CPU resources to process P3, then 25% of resources is for process scoped threads and the remaining 25% for system scoped threads. The PCS threads T6 and T7 will be allocated to access the 25% resources based on the priority by the thread library. The SCS threads T8, T9, T10 will divide the 25% resources among themselves and access the kernel resources via separate LWP and KLT. The SCS scheduling is by the system scheduler.

Note:

```
For every system call to access the kernel resources, a Kernel
Level thread is created
and associated to separate LWP by the system scheduler.


Number of Kernel Level Threads = Total Number of LWP
Total Number of LWP = Number of LWP for SCS + Number of LWP for PCS
Number of LWP for SCS = Number of SCS threads
Number of LWP for PCS = Depends on application developer
```

Here,

```
Number of SCS threads = 5
Number of LWP for PCS = 3
Number of SCS threads = 5
Number of LWP for SCS = 5
Total Number of LWP   = 8 (=5+3)
Number of Kernel Level Threads = 8
```

Advantages of PCS over SCS

- If all threads are PCS, then context switching, synchronization, scheduling everything takes place within the userspace. This reduces system calls and achieves better performance.
- PCS is cheaper than SCS.
- PCS threads share one or more available LWPs. For every SCS thread, a separate LWP is associated.For every system call, a separate KLT is created.
- The number of KLT and LWPs created highly depends on the number of SCS threads created. This increases the kernel complexity of handling scheduling and synchronization. Thereby, results in a limitation over SCS thread creation, stating that, the number of SCS threads to be smaller than the number of PCS threads.
- If the system has more than one allocation domain, then scheduling and synchronization of resources becomes more tedious. Issues arise

when an SCS thread is a part of more than one allocation domain, the system has to handle n number of interfaces.

The second boundary of thread scheduling involves CPU scheduling by the system scheduler. The scheduler considers each kernel-level thread as a separate process and provides access to the kernel resources.

FAQs on Thread Scheduling

1. What is a time slice or quantum in thread scheduling?

A time slice, also known as a quantum, is the maximum amount of time a thread is allowed to execute before the scheduler can potentially switch to another thread. Preemptive scheduling algorithms use time slices to ensure fairness and prevent threads from monopolizing the CPU.

2. How do priority levels affect thread scheduling?

Thread scheduling algorithms often use priority levels to determine which thread should run next. Higher-priority threads are given preference over lower-priority threads. Priority-based scheduling helps manage the urgency and importance of different threads.

Are you a student in Computer Science or an employed professional looking to take up the **GATE 2025 Test**? Of course, you can get a good score in it but to get the best score our **GATE CS/IT 2025 - Self-Paced Course** is available on GeeksforGeeks to help you with its preparation.

M **maha...** 29

Previous Article

Multiple-Processor Scheduling in Operating System

Next Article

Thread in Operating System

Similar Reads

Relationship between User level thread and Kernel level thread

Introduction: User-level threads and kernel-level threads are two different approaches to implementing thread management in an operating system....

9 min read

Difference Between Thread ID and Thread Handle

Prerequisite: Thread in Operating System Thread Id is a long positive integer that is created when the thread was created. During the entire...

3 min read

Difference between User Level thread and Kernel Level thread

User-level threads are threads that are managed entirely by the user-level thread library, without any direct intervention from the operating system's...

4 min read

Difference between Priority Scheduling and Round Robin (RR) CPU...

1. Priority Scheduling Algorithm : Priority scheduling algorithm executes the processes depending upon their priority. Each process is allocated a...

3 min read

Difference between Priority scheduling and Shortest Job First (SJF)...

1. Priority Scheduling Algorithm : Priority scheduling algorithm executes the processes depending upon their priority. Each process is allocated a...

3 min read

View More Articles

Article Tags : Computer Subject Operating Systems Write From Home

All Cheat Sheets

NLP Tutorial
Deep Learning Tutorial

Web Technologies
HTML
CSS
JavaScript
TypeScript
ReactJS
NextJS
Bootstrap
Web Design

Python Tutorial
Python Programming Examples
Python Projects
Python Tkinter
Web Scraping
OpenCV Tutorial
Python Interview Question
Django

Computer Science
Operating Systems
Computer Network
Database Management System
Software Engineering
Digital Logic Design
Engineering Maths
Software Development
Software Testing

DevOps
Git
Linux
AWS
Docker
Kubernetes
Azure
GCP
DevOps Roadmap

System Design
High Level Design
Low Level Design
UML Diagrams
Interview Guide
Design Patterns
OOAD
System Design Bootcamp
Interview Questions

Inteview Preparation
Competitive Programming
Top DS or Algo for CP
Company-Wise Recruitment Process
Company-Wise Preparation
Aptitude Preparation
Puzzles

School Subjects
Mathematics
Physics
Chemistry
Biology
Social Science
English Grammar
Commerce
World GK

GeeksforGeeks Videos
DSA
Python
Java
C++
Web Development
Data Science
CS Subjects