Great Place To Work®
Certified
FEB 2023 – FEB 2024
INDIA™

UPES

UNIVERSITY OF TOMORROW

nirf
RANKING 2023

#RANKED 52
IN INDIA

#University Category

QS WORLD UNIVERSITY RANKINGS

NO.1 PVT. UNIVERSITY IN ACADEMIC REPUTATION IN INDIA

ACCREDITED WITH GRADE A NAAC

ACCREDITED **GRADE 'A'** BY NAAC

E-LEAD

PERFECT SCORE OF **150/150** AS A TESTAMENT TO EXECEPTIONAL E-LEARNING METHODS

Unit 2 : Process and Thread Management

Lecture 7

Submitted by:

Dr Khushboo Jain

School of Computer Science
UPES, Dehradun
India

Table of Contents

1. Thread Scheduling: Content Scope,
2. Pthread Scheduling
3. Threading Issues

Learning & Course Outcomes

LO1: To understand Thread Scheduling
LO2: To analyze Pthread Scheduling
LO3: To evaluate Threading Issues

CO2: Evaluate and analyze process and thread scheduling techniques, discerning their benefits and challenges.

UPES
UNIVERSITY OF TOMORROW

Thread Libraries

- **Thread library** provides programmer with API for creating and managing threads
- Two primary ways of implementing
  - Library entirely in user space
  - Kernel-level library supported by the OS

Pthread

- May be provided either as user-level or kernel-level
- A POSIX standard (IEEE 1003.1c) API for thread creation and synchronization
- ***Specification***, not ***implementation***
- API specifies behavior of the thread library, implementation is up to development of the library
- Common in UNIX operating systems (Solaris, Linux, Mac OS X)

Thread Scheduling

- Distinction between user-level and kernel-level threads
- When threads supported, threads scheduled, not processes
- Many-to-one and many-to-many models, thread library schedules user-level threads to run on LWP
  - Known as **process-contention scope** (**PCS**) since scheduling competition is within the process
  - Typically done via priority set by programmer
- Kernel thread scheduled onto available CPU is **system-contention scope** (**SCS**) – competition among all threads in system

Pthread Scheduling

- API allows specifying either PCS or SCS during thread creation
  - PTHREAD_SCOPE_PROCESS schedules threads using PCS scheduling
  - PTHREAD_SCOPE_SYSTEM schedules threads using SCS scheduling
- Can be limited by OS – Linux and Mac OS X only allow PTHREAD_SCOPE_SYSTEM

Pthread Scheduling API

```
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS 5
int main(int argc, char *argv[]) {
   int i, scope;
   pthread_t tid[NUM THREADS];
   pthread_attr_t attr;
   /* get the default attributes */
   pthread_attr_init(&attr);
   /* first inquire on the current scope */
   if (pthread_attr_getscope(&attr, &scope) != 0)
      fprintf(stderr, "Unable to get scheduling scope\n");
   else {
      if (scope == PTHREAD_SCOPE_PROCESS)
         printf("PTHREAD_SCOPE_PROCESS");
      else if (scope == PTHREAD_SCOPE_SYSTEM)
         printf("PTHREAD_SCOPE_SYSTEM");
      else
         fprintf(stderr, "Illegal scope value.\n");
   }
```

UPES
UNIVERSITY OF TOMORROW

Pthread Scheduling API

```
 /* set the scheduling algorithm to PCS or SCS */
   pthread_attr_setscope(&attr, PTHREAD_SCOPE_SYSTEM);
   /* create the threads */
   for (i = 0; i < NUM_THREADS; i++)
      pthread_create(&tid[i],&attr,runner,NULL);
   /* now join on each thread */
   for (i = 0; i < NUM_THREADS; i++)
      pthread_join(tid[i], NULL);
}
/* Each thread will begin control in this function */
void *runner(void *param)
{
   /* do some work ... */
   pthread_exit(0);
}
```

Threading Issues

- Semantics of **fork()** and **exec()** system calls
- Signal handling
  - Synchronous and asynchronous
- Thread cancellation of target thread
  - Asynchronous or deferred
- Thread-local storage
- Scheduler Activations

Semantics of fork() and exec()

- Does **fork()**duplicate only the calling thread or all threads?
  - Some UNIXes have two versions of fork
- **exec()** usually works as normal – replace the running process including all threads

Signal Handling

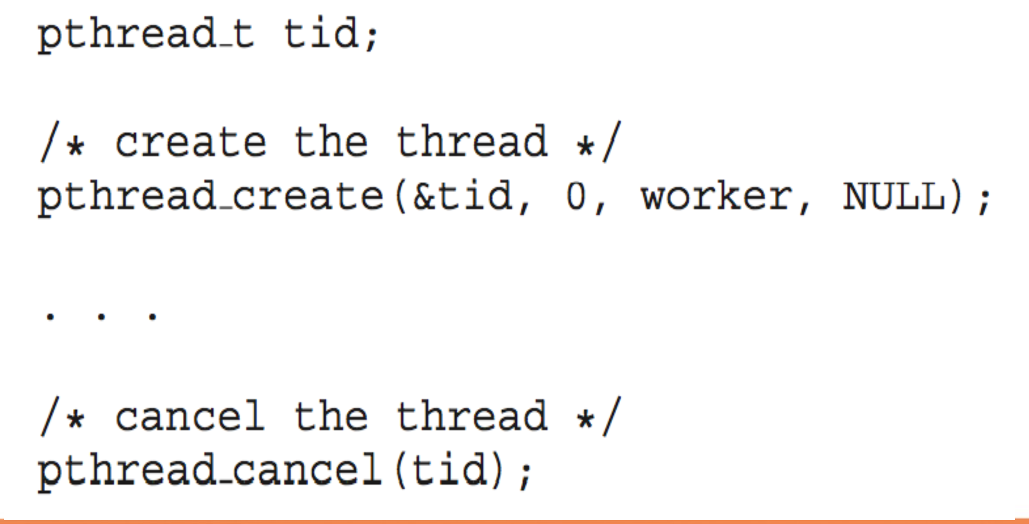
- **Signals** are used in UNIX systems to notify a process that a particular event has occurred.
- A **signal handler** is used to process signals
  - Signal is generated by particular event
  - Signal is delivered to a process
  - Signal is handled by one of two signal handlers:
    - default
    - user-defined
- Every signal has **default handler** that kernel runs when handling signal
  - **User-defined signal handler** can override default
  - For single-threaded, signal delivered to process

Signal Handling (Cont.)

- Where should a signal be delivered for multi-threaded?
  - Deliver the signal to the thread to which the signal applies
  - Deliver the signal to every thread in the process
  - Deliver the signal to certain threads in the process
  - Assign a specific thread to receive all signals for the process

Thread Cancellation

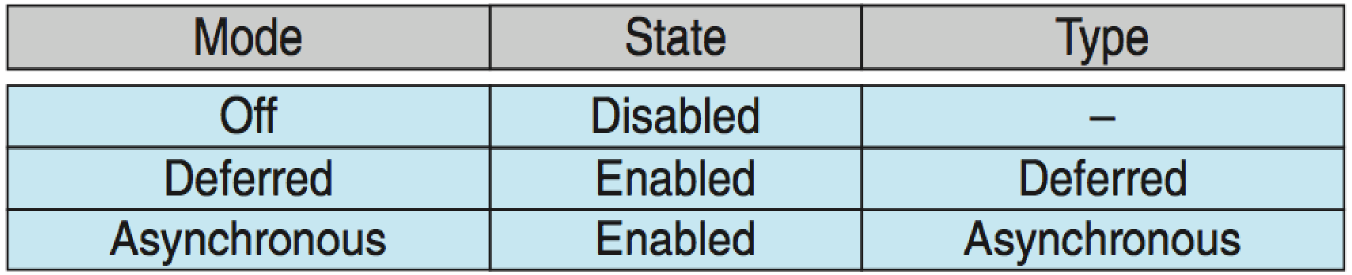
- Terminating a thread before it has finished
- Thread to be canceled is **target thread**
- Two general approaches:
  - **Asynchronous cancellation** terminates the target thread immediately
  - **Deferred cancellation** allows the target thread to periodically check if it should be cancelled
- Pthread code to create and cancel a thread:



```
pthread_t tid;

/* create the thread */
pthread_create(&tid, 0, worker, NULL);

. . .

/* cancel the thread */
pthread_cancel(tid);
```

Thread Cancellation (Cont.)

- Invoking thread cancellation requests cancellation, but actual cancellation depends on thread state



Mode	State	Type
Off	Disabled	–
Deferred	Enabled	Deferred
Asynchronous	Enabled	Asynchronous

- If thread has cancellation disabled, cancellation remains pending until thread enables it
- Default type is deferred
  - Cancellation only occurs when thread reaches **cancellation point**
    - I.e. **pthread_testcancel()**
    - Then **cleanup handler** is invoked
- On Linux systems, thread cancellation is handled through signals

Thread-Local Storage

- **Thread-local storage** (**TLS**) allows each thread to have its own copy of data
- Useful when you do not have control over the thread creation process (i.e., when using a thread pool)
- Different from local variables
  - Local variables visible only during single function invocation
  - TLS visible across function invocations
- Similar to **static** data
  - TLS is unique to each thread

Scheduler Activation

- Both M:M and Two-level models require communication to maintain the appropriate number of kernel threads allocated to the application
- Typically use an intermediate data structure between user and kernel threads – **lightweight process** (**LWP**)
  - Appears to be a virtual processor on which process can schedule user thread to run
  - Each LWP attached to kernel thread
  - How many LWPs to create?
- Scheduler activations provide **upcalls** - a communication mechanism from the kernel to the **upcall handler** in the thread library
- This communication allows an application to maintain the correct number kernel threads

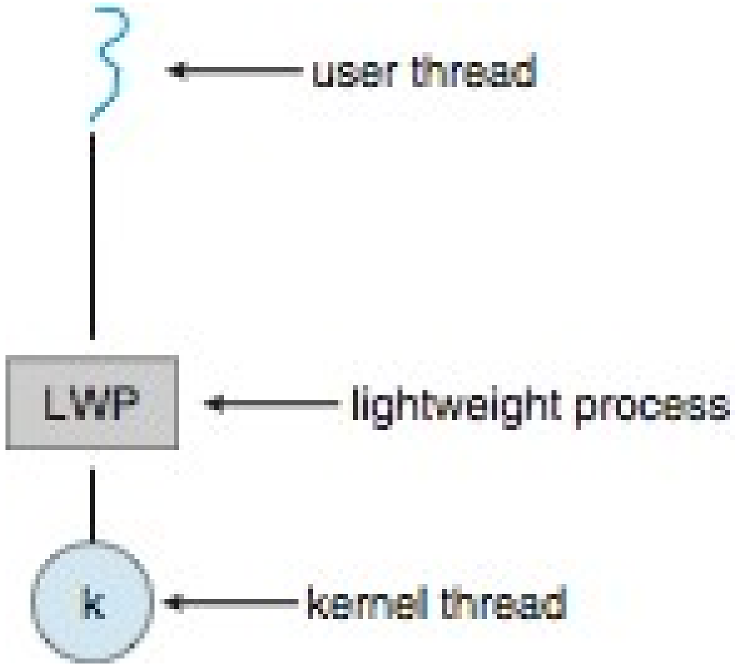


Fig 1. Scheduler Activation [1]

Summary

- Thread Scheduling Overview: Covers the principles and algorithms used to manage the execution of multiple threads within an operating system to ensure efficient and fair resource allocation.
- Pthread Scheduling: Discusses the scheduling mechanisms and policies provided by the POSIX thread (Pthread) library, which includes various priority-based scheduling options and real-time scheduling features.
- Threading Issues: Addresses common challenges in multi-threaded programming, Semantics of **fork()** and **exec()** system calls, Signal handling, Thread cancellation of target thread, Thread-local storage and scheduler Activations

Reference Material

[1]. Silberschatz, A. & Galvin, P. (2009) Operating System Concepts. 8th ed. NJ: John Wiley & Sons, Inc.

- Download Link- https://www.mbit.edu.in/wp-content/uploads/2020/05/Operating_System_Concepts_8th_EditionA4.pdf

[2]. NPTEL Video Lecture: https://onlinecourses.nptel.ac.in/noc24_cs80/preview

MCQ’s

1 What does thread scheduling primarily involve?

- a) Managing thread resources
- b) Assigning threads to specific processors
- c) Determining the order in which threads execute
- d) Handling thread synchronization

2 Which of the following is a common thread scheduling algorithm used in pthreads?

- a) First-Come, First-Served (FCFS)
- b) Round Robin
- c) Shortest Job Next (SJN)
- d) Depth-First Search (DFS)

UPES
UNIVERSITY OF TOMORROW

MCQ’s

3 What is one potential issue with thread scheduling?

a) Deadlock
b) Livelock
c) Starvation
d) Race condition

4. What is a common approach to mitigate threading issues such as race conditions?

a) Using mutual exclusion mechanisms like mutexes
b) Increasing the number of threads in the system
c) Ignoring the issue and relying on the operating system to handle it
d) Encouraging more concurrent access to shared resources

MCQ’s

5. What is the primary goal of thread scheduling?

  a) Maximizing CPU utilization
  b) Minimizing context switching overhead
  c) Ensuring fairness and responsiveness among threads
  d) Prioritizing I/O-bound threads over CPU-bound threads

6. Which threading issue occurs when a thread is perpetually denied access to resources it needs to execute?

  a) Deadlock
  b) Livelock
  c) Starvation
  d) Race condition

MCQ's

Answers

1. c) Determining the order in which threads execute
2. b) Round Robin
3. c) Starvation
4. a) Using mutual exclusion mechanisms like mutexes
5. c) Ensuring fairness and responsiveness among threads
6. c) Starvation

What’s Next

Case study: Process Management in Linux

UPES

UNIVERSITY OF TOMORROW

Thank You