

Education

UNIVERSITY OF LEEDS

MS IN DATA SCIENCE/ANALYSIS

📅 Sep 2022 📍 Leeds, UK

- International Masters Excellence Scholarship
- Expecting a First class degree
- Working as PAL Mentor

MITS, GWALIOR

BTECH IN COMPUTER SCIENCE

📅 Sep 2020 📍 Gwalior, MP

Graduated with Distinction

Links

🐙 GitHub **themagicalmammal**
in LinkedIn **themagicalmammal**

Coursework

GRADUATE

Data Science

Programming for Data Science

Knowledge Representation and Reasoning

Statistical Theory and Methods

Learning Skills through Case Studies

Artificial Intelligence

Machine Learning

Statistical Learning

Dissertation: Machine learning methods for solubility prediction in chemical experiments

Skills

PROGRAMMING

Python • R • PHP • C/C++ • HTML/CSS • JavaScript • SQL

MISCELLANEOUS

Shell • \LaTeX • Tableau • Microsoft Office • Firebase • Git

Honors

EXCELLENCE SCHOLARSHIP

UNIVERSITY OF LEEDS

Awarded to international students who have achieved the equivalent of a UK first class bachelors' degree.

INDUSTRIAL VISIT

INFOSYS PUNE

I was as one of the students for industrial training, where 28 students were chosen out of 2000.

Experience

SOFTWARE ENGINEER

INDIAN INSTITUTE OF TECHNOLOGY

📅 Sep 2020 – Aug 2021 📍 Indore, MP

- Using Yolo to boost the accuracy of low light situations in mobilenetv3
- Using Bash scripts and Linux tools to automate and optimize the data handling for traffic signs.

MACHINE LEARNING INTERN

RAJA RAMANNA CENTRE FOR ADVANCED TECHNOLOGY

📅 Apr 2020 – Aug 2020 📍 Indore, MP

- Using Genetic algorithm designed a model to detect the crossover/mutation in gene which is portable for scheduling scenarios scenarios.

WEB DEVELOPMENT INTERN

RAJA RAMANNA CENTRE FOR ADVANCED TECHNOLOGY

📅 Jan 2020 – Feb 2020 📍 Indore, MP

- Designed the dynamic website using PHP for data visualization (Angular5 and NodeJs) of particle accelerator for internal usage.
- Worked on AmCharts to display live data streams and used statistical tools for computation and understanding of data.

WEB DEVELOPMENT INTERN

RAJA RAMANNA CENTRE FOR ADVANCED TECHNOLOGY

📅 Jun 2019 – Jul 2019 📍 Indore, MP

- Using Django to connect with SQL backend with SQL Server 2012.
- The webpage is made out of AngularJS with UI elements from Materialize.

Competitions

COMPLEX LEVEL

SMART INDIA HACKATHON

📅 Jan 2019 – Feb 2019 📍 Bhubaneswar, Orissa

- We had a team of 6 people under the alias 'Cicada3301' everyone had different skill sets we made a website which was connected to a blockchain network on Ethereum which was set up on the network of our personal computers.
- After passing all the three rounds, our team was declared the winner of our question set. We won a lump sum of 1lakh₹(s).

Recent Projects

IPL ANALYSIS

UNIVERSITY OF LEEDS

📅 Nov 2021 – Dec 2021 📍 Leeds, UK

A project under the University of Leeds to analyse data from 2008 to 2015 and discover patterns in the data such as trends, co-relations, probabilities, and used tools such as matplotlib, Pandas, Seaborn for data visualization.

STUDY OF THE BEHAVIOUR OF SERIAL KILLERS'


UNIVERSITY OF LEEDS


📅 Nov 2021 – Dec 2021 📍 Leeds, UK


The study of the behaviour of serial killers with different motives such as Convenience (did not want children/spouse), Enjoyment or Power and Escape or avoiding arrest.

HARSH GADGIL


Data Engineer

 (647) 221 7999

 hgadgil.com

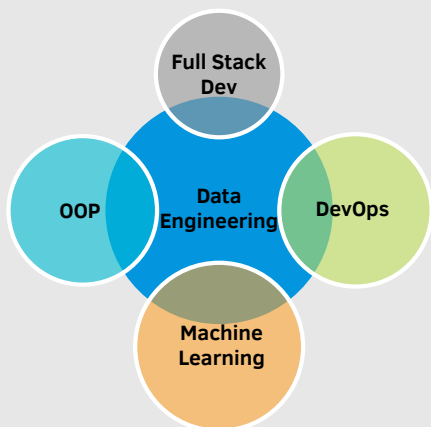
 harsh.gadgil@gmail.com

 /in/hsgadgil

 opensorceror

Technical Skills

Overview



Programming

0 LOC —————> 5000 LOC

Scala • Java • Python

JavaScript • SQL • \LaTeX

C • C++ • R

Education

MSc., Computer Science (GPA: 3.7)

Specialization: Data Mining

University of Guelph

2015 - 2017 | Guelph, Canada

BEng., Computer Engineering (GPA: 4.0)

University of Pune

2009 - 2013 | Pune, India

Experience

April 2017 - **Machine Learning Engineer**

Bell

Present

- Focused on developing machine learning models, production deployment, testing, scaling
- Projects: SMS spam detection, Robocall detection
Tools: Kafka, Flume, Spark, Hive, Solr, Jupyter Lab, Kubernetes, Docker, GitLab CI / CD
Awards: Merit Gold (3x), Merit Platinum (2x)
- Helped pioneer automated deployments of data pipelines using CI / CD, enabling faster and more portable deployments
- Presented several talks on Machine Learning and Big Data at university hackathons and Bell internal conferences
- Conducted interviews for junior and senior data engineering roles

Sep 2015 -
May 2016

Co-founder & Full Stack Developer

LocalXChange Inc.

- In a team of 2, raised \$8,000 in funding from The Hub incubator at the University of Guelph, to develop a hyperlocal content platform, aimed at delivering local news and events to local users in realtime
- In a team of 3, built hybrid mobile & web apps with Ionic, Angular.js and MongoDB, surpassing 1,000 users within a month since launch
- Met with the Mayor of Guelph and University of Guelph officials to discuss how the app can help boost Guelph Tourism

Sep 2015 -
Dec 2016

Graduate Teaching Assistant

University of Guelph

- TA for CIS*2430 (OOP), CIS*4150 (Software Reliability & Testing) and CIS*3530 (Database Systems & Concepts) courses
- Delivered several lectures for CIS*3530 when professor was away at a conference

Dec 2013 -
Apr 2015

Test Automation Engineer

Synechron

- Primarily developed test automation libraries using Java and C#, scripts and CI / CD pipelines
- Lead development of a Keyword Driven & Behavior Driven test framework for Microsoft Dynamics CRM. Earned monetary award & client appreciation.
- Demonstrated that rewriting an in-house test framework for Microsoft Dynamics AX, using an open source library (White) instead of a proprietary one (Coded UI), would help the team save \$4K annually by downgrading Microsoft Visual Studio

Research

2015 - 2017 **MSc. Candidate, Graduate Research Assistant**

University of Guelph

Thesis: Data Integration from Multiple Historical Sources to Study Canadian Casualties of WWI







- Proposed a stepwise deterministic method to integrate datasets without labeled data. The method performs comparably with a method that incorporates a Support Vector Machine
- Prepared a longitudinal dataset to enable comprehensive analyses about WWI Canadian society and military, seeding further research
- **Tools:** R, Python, scikit-learn, pandas

Publications

L. Antonie, H. Gadgil, G. Grewal, and K. Inwood, "Historical Data Integration - A Study of WWI Canadian Soldiers," in 2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW), pp. 186-193, IEEE, 2016.

MARISSA MAYER

Business Woman & Proud Geek

 mmayer@yahoo-inc.com  Address, Street, 00000 County  Sunnyvale, CA
 marissamayr.tumblr.com  @marissamayer  marissamayer



EXPERIENCE

President & CEO

Yahoo!

 July 2012 – Ongoing  Sunnyvale, CA

- Led the \$5 billion acquisition of the company with Verizon – the entity which believed most in the immense value Yahoo! has created
- Acquired Tumblr for \$1.1 billion and moved the company's blog there
- Built Yahoo's mobile, video and social businesses from nothing in 2011 to \$1.6 billion in GAAP revenue in 2015
- Tripled the company's mobile base to over 600 million monthly active users and generated over \$1 billion of mobile advertising revenue last year

Vice President of Location & Services

Google

 Oct 2010 – July 2012  Palo Alto, CA

- Position Google Maps as the world leader in mobile apps and navigation
- Oversaw 1000+ engineers and product managers working on Google Maps, Google Places and Google Earth

Vice President of Search Products & UX

Google

 2005 – 2010  Palo Alto, CA

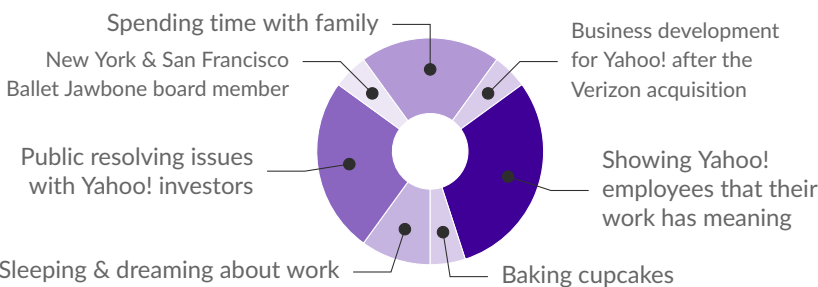
Product Manager & UI Lead

Google

 Oct 2001 – July 2005  Palo Alto, CA

- Appointed by the founder Larry Page in 2001 to lead the Product Management and User Interaction teams
- Optimized Google's homepage and A/B tested every minor detail to increase usability (incl. spacing between words, color schemes and pixel-by-pixel element alignment)


A DAY OF MY LIFE





LIFE PHILOSOPHY


"If you don't have any shadows, you're not standing in the light."

MOST PROUD OF

 **Courage I had**
to take a sinking ship and try to make it float

 **Persistence & Loyalty**
I showed despite the hard moments and my willingness to stay with Yahoo after the acquisition

 **Google's Growth**
from a hundred thousand searches per day to over a billion

 **Inspiring women in tech**
Youngest CEO on Fortune's list of 50 most powerful women

STRENGTHS

Hard-working (18/24) Persuasive
Motivator & Leader

UX Mobile Devices & Applications
Product Management & Marketing

LANGUAGES

English
Spanish
German



EDUCATION

M.S. in Computer Science
Stanford University

 Sept 1997 – June 1999

B.S. in Symbolic Systems
Stanford University

 Sept 1993 – June 1997

PUBLICATIONS

Books

- E. Someone and **T. Lim**, *A Fictional Research*. Somewhere, Some Place, 2010.

Journal Articles

- **L. T. Wong** and E. Someone, "A non-existant paper," *Journal of Carrying On*, vol. 12, 2011.
- **L. T. Lim**, E. Someone, and A. Other, "A study into fireside story-telling," *Journal of Carrying On*, vol. 7, 2008.

Conference Proceedings

- E. Someone and **L. T. Lim**, "Another paper something something," in *Proceedings of the 72nd AmaZing Conference*, Far Far Away, 2013.

REFEREES

Prof. Alpha Beta

@ Institute

✉ a.beta@university.edu

Address Line 1

Address line 2

Prof. Gamma Delta

@ Institute

✉ g.delta@university.edu

Address Line 1

Address line 2

THREADS

4.1 Processes and Threads

- Multithreading
- Thread Functionality

4.2 Types of Threads

- User-Level and Kernel-Level Threads
- Other Arrangements

4.3 Multicore and Multithreading

- Performance of Software on Multicore
- Application Example: Valve Game Software

4.4 Windows Process and Thread Management

- Management of Background Tasks and Application Lifecycles
- The Windows Process
- Process and Thread Objects
- Multithreading
- Thread States
- Support for OS Subsystems

4.5 Solaris Thread and SMP Management

- Multithreaded Architecture
- Motivation
- Process Structure
- Thread Execution
- Interrupts as Threads

4.6 Linux Process and Thread Management

- Linux Tasks
- Linux Threads
- Linux Namespaces

4.7 Android Process and Thread Management

- Android Applications
- Activities
- Processes and Threads

4.8 Mac OS X Grand Central Dispatch

4.9 Summary

4.10 Key Terms, Review Questions, and Problems

LEARNING OBJECTIVES

After studying this chapter, you should be able to:

- Understand the distinction between process and thread.
- Describe the basic design issues for threads.
- Explain the difference between user-level threads and kernel-level threads.
- Describe the thread management facility in Windows.
- Describe the thread management facility in Solaris.
- Describe the thread management facility in Linux.

This chapter examines some more advanced concepts related to process management, which are found in a number of contemporary operating systems. We show that the concept of process is more complex and subtle than presented so far and in fact embodies two separate and potentially independent concepts: one relating to resource ownership, and another relating to execution. This distinction has led to the development, in many operating systems, of a construct known as the **thread**.

4.1 PROCESSES AND THREADS

The discussion so far has presented the concept of a process as embodying two characteristics:

- 1. Resource ownership:** A process includes a virtual address space to hold the process image; recall from Chapter 3 that the process image is the collection of program, data, stack, and attributes defined in the process control block. From time to time, a process may be allocated control or ownership of resources, such as main memory, I/O channels, I/O devices, and files. The OS performs a protection function to prevent unwanted interference between processes with respect to resources.
- 2. Scheduling/execution:** The execution of a process follows an execution path (trace) through one or more programs (e.g., Figure 1.5). This execution may be interleaved with that of other processes. Thus, a process has an execution state (Running, Ready, etc.) and a dispatching priority, and is the entity that is scheduled and dispatched by the OS.

Some thought should convince the reader that these two characteristics are independent and could be treated independently by the OS. This is done in a number of operating systems, particularly recently developed systems. To distinguish the two characteristics, the unit of dispatching is usually referred to as a thread or

lightweight process, while the unit of resource ownership is usually referred to as a **process** or **task**.¹

Multithreading

Multithreading refers to the ability of an OS to support multiple, concurrent paths of execution within a single process. The traditional approach of a single thread of execution per process, in which the concept of a thread is not recognized, is referred to as a single-threaded approach. The two arrangements shown in the left half of Figure 4.1 are single-threaded approaches. MS-DOS is an example of an OS that supports a single-user process and a single thread. Other operating systems, such as some variants of UNIX, support multiple user processes, but only support one thread per process. The right half of Figure 4.1 depicts multithreaded approaches. A Java runtime environment is an example of a system of one process with multiple threads. Of interest in this section is the use of multiple processes, each of which supports multiple threads. This approach is taken in Windows, Solaris, and many

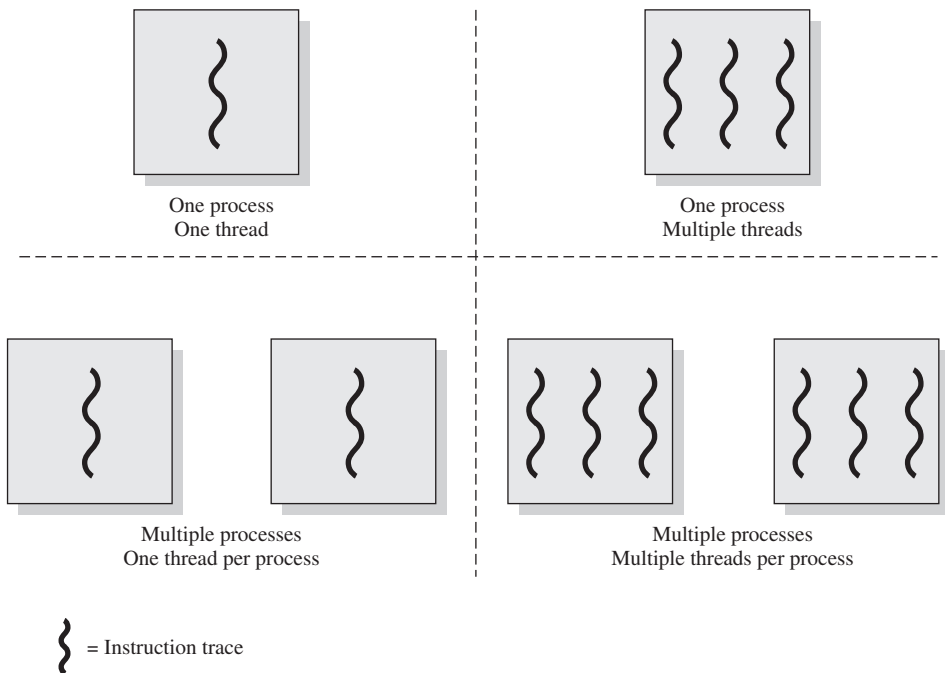


Figure 4.1 Threads and Processes

¹Alas, even this degree of consistency is not maintained. In IBM's mainframe operating systems, the concepts of address space and task, respectively, correspond roughly to the concepts of process and thread that we describe in this section. Also, in the literature, the term *lightweight process* is used as either (1) equivalent to the term *thread*, (2) a particular type of thread known as a kernel-level thread, or (3) in the case of Solaris, an entity that maps user-level threads to kernel-level threads.

Shubhi Rani

Linkedin: <https://www.linkedin.com/in/shubhir/>

Github: <https://github.com/shubhi28>

Email : shubhi2808@gmail.com

Mobile : +1-631-645-8315

EDUCATION

- **Stony Brook University** Stony Brook, NY
Masters in Computer Science; GPA: 3.54 *Aug 2015 - Dec 2016*
Courses: Operating Systems, Analysis Of Algorithms, Artificial Intelligence, Machine Learning, Probability and Statistics and Network Security.
- **Birla Institute of Technology** Mesra, India
Bachelor of Computer Science; GPA: 3.9 (8.54/10.0 - First in class of 60) *Aug 2008 - May 2012*

SKILLS SUMMARY

- **Languages:** Java, C++, Python, C, SQL, Unix scripting
- **Tools:** Kubernetes, Docker, Springboot, GIT, JIRA, Matlab, XCode, Postgres

EXPERIENCE

- **VMware** Palo Alto, CA
Member Of Technical Staff *Feb 2017 - Current*
 - **Events and Alert Manager:** Network Fabric Controller is a logically centralized software controller to manage a distributed physical network fabric or a physical network underlay. Designed and developed a library which can be used by any services within Network Fabric Controller to generate events and raise alerts for NFC managed objects. The events and alerts are displayed on the NFC dashboard.
 - **Upgrade NFC:** Designed and developed an over-the-air and air-gapped upgrade mechanism that is used to upgrade the single node Network Fabric Controller cluster.
 - **Health Monitoring System:** Designed and developed a monitoring service which is responsible for monitoring the health of all the micro services running inside NFC cluster.
 - **CLI framework:** Developed an internal command line interface tool which provides a set of commands specific to Network Fabric Controller projects to get the system health, logs and current resource utilization. It can be easily extended to perform various other actions.
 - **Bootstrap NFC:** Network Fabric Controller is composed of several micro services deployed on the Kubernetes pods on a single-node cluster. Designed and implemented the bootstrapping mechanism to package all the services and deploy on the Kubernetes environment.
 - **Install/Upgrade/Uninstall NSX agent:** Worked on install, upgrade and uninstall mechanism of NSX agent on workload VMs deployed on NSX cross cloud environment.
 - **AppDiscovery:** Worked on application profiling feature which provides visualization and details of which processes inside a workload VM are communicating on the network.
- **Stony Brook University** Stony Brook, NY
Research Assistant - Prof. Erez Zadok *May 2016 - August 2016*
 - **System Call Trace Record/Replay:** Worked on building a trace replayer at system call level to reproduce system call operations that were captured during a specific workload using C, C++, DataSeries. Developed a wrapper class that makes C++ functions callable by strace C code.
- **Samsung Research Institute** Noida, India
Software Developer Engineer *Jun 2012 - July 2015*
 - **Android File System:**
 - Involved in board bring-up activities for Android Smart phones based on Exynos and Broadcom chipsets on Android version 4.3 Jelly Bean to Android 5.0 Lollipop.
 - Experienced in porting of File System (FAT, EXFAT, SDCARDFS, EXT4) on Samsung mobiles proprietary platform.
 - Enhanced performance of smart phones having low RAM by analyzing performance using blktrace and tuning kernel parameters. The code was merged in around 15 smart phones.

ACADEMIC PROJECTS

- **Plug board Proxy (Networking):** Developed a plug board proxy that adds an extra layer of encryption to connections towards TCP services. Clients running on same server connect to pbproxy, which then relays all traffic to actual services. (Mar '16)
- **Asynchronous Work Queue Manager (Kernel Programming):** Developed a kernel module to serve as an asynchronous work queue manager with configurable worker threads. Implemented netlink sockets to propagate callbacks from kernel to user land and throttling to improve job extraction latency. (Nov '15)
- **Anti-Malware Stackable File System (Kernel Programming):** Implemented a stackable, anti-malware Linux file system that prevents the existing file system from being corrupted by malware by detecting virus pattern while attempting to open, read and write a file. (Oct '15)

- **File Encryption System Call (Kernel Programming):** Implemented a system call in Linux kernel, which supports multiple ciphers to encrypt or decrypt an input file.(Sep '15)
- **Peg- Solitaire, Connect Four, Sudoku (Game Development):** Designed a Peg Solitaire, Connect Four and Sudoku using Iterative Deepening Search, Alpha-beta pruning and Backtracking, MRV and Forward Chaining Artificial Intelligence Algorithms respectively in Python. (Aug '15)

HONORS AND AWARDS

- Selected in top 20 students for the Code House event organized by VMware in August15 - August17, 2016.
- Ranked first among batch of 60 students in my Computer Science Engineering Branch.
- Ranked fifth among batch of 500 students at High School Level A.I.S.S.E 2005

modern versions of UNIX, among others. In this section, we give a general description of multithreading; the details of the Windows, Solaris, and Linux approaches will be discussed later in this chapter.

In a multithreaded environment, a process is defined as the unit of resource allocation and a unit of protection. The following are associated with processes:

- A virtual address space that holds the process image
- Protected access to processors, other processes (for interprocess communication), files, and I/O resources (devices and channels)

Within a process, there may be one or more threads, each with the following:

- A thread execution state (Running, Ready, etc.)
- A saved thread context when not running; one way to view a thread is as an independent program counter operating within a process
- An execution stack
- Some per-thread static storage for local variables
- Access to the memory and resources of its process, shared with all other threads in that process

Figure 4.2 illustrates the distinction between threads and processes from the point of view of process management. In a single-threaded process model (i.e., there is no distinct concept of thread), the representation of a process includes its process control block and user address space, as well as user and kernel stacks to manage the call/return behavior of the execution of the process. While the process is running, it controls the processor registers. The contents of these registers are saved when the process is not running. In a multithreaded environment, there is still a single process

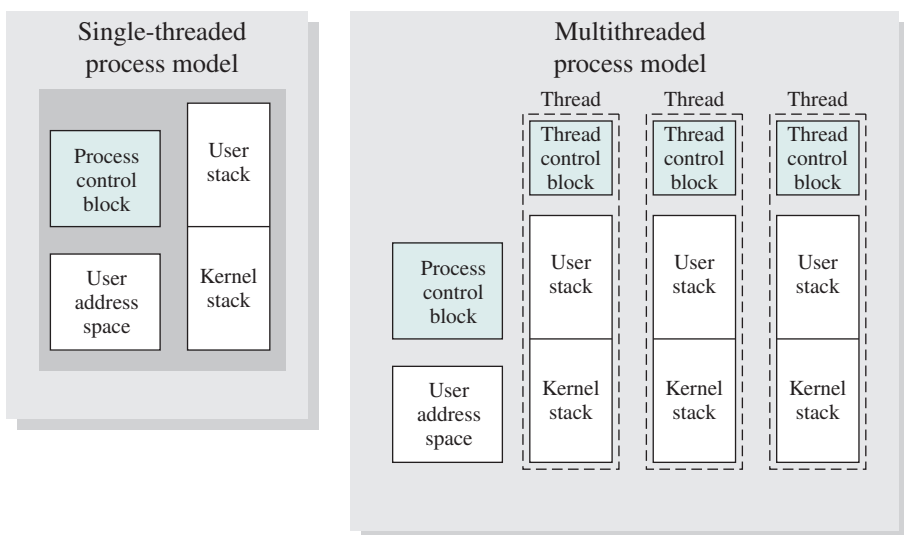


Figure 4.2 Single-Threaded and Multithreaded Process Models

control block and user address space associated with the process, but now there are separate stacks for each thread, as well as a separate control block for each thread containing register values, priority, and other thread-related state information.

Thus, all of the threads of a process share the state and resources of that process. They reside in the same address space and have access to the same data. When one thread alters an item of data in memory, other threads see the results if and when they access that item. If one thread opens a file with read privileges, other threads in the same process can also read from that file.

The key benefits of threads derive from the performance implications:

1. It takes far less time to create a new thread in an existing process, than to create a brand-new process. Studies done by the Mach developers show that thread creation is ten times faster than process creation in UNIX [TEVA87].
2. It takes less time to terminate a thread than a process.
3. It takes less time to switch between two threads within the same process than to switch between processes.
4. Threads enhance efficiency in communication between different executing programs. In most operating systems, communication between independent processes requires the intervention of the kernel to provide protection and the mechanisms needed for communication. However, because threads within the same process share memory and files, they can communicate with each other without invoking the kernel.

Thus, if there is an application or function that should be implemented as a set of related units of execution, it is far more efficient to do so as a collection of threads, rather than a collection of separate processes.

An example of an application that could make use of threads is a file server. As each new file request comes in, a new thread can be spawned for the file management program. Because a server will handle many requests, many threads will be created and destroyed in a short period. If the server runs on a multiprocessor computer, then multiple threads within the same process can be executing simultaneously on different processors. Further, because processes or threads in a file server must share file data and therefore coordinate their actions, it is faster to use threads and shared memory than processes and message passing for this coordination.

The thread construct is also useful on a single processor to simplify the structure of a program that is logically doing several different functions.

[LETW88] gives four examples of the uses of threads in a single-user multiprocessing system:

1. **Foreground and background work:** For example, in a spreadsheet program, one thread could display menus and read user input, while another thread executes user commands and updates the spreadsheet. This arrangement often increases the perceived speed of the application by allowing the program to prompt for the next command before the previous command is complete.
2. **Asynchronous processing:** Asynchronous elements in the program can be implemented as threads. For example, as a protection against power failure, one can design a word processor to write its random access memory (RAM)

buffer to disk once every minute. A thread can be created whose sole job is periodic backup and that schedules itself directly with the OS; there is no need for fancy code in the main program to provide for time checks or to coordinate input and output.

3. **Speed of execution:** A multithreaded process can compute one batch of data while reading the next batch from a device. On a multiprocessor system, multiple threads from the same process may be able to execute simultaneously. Thus, even though one thread may be blocked for an I/O operation to read in a batch of data, another thread may be executing.
4. **Modular program structure:** Programs that involve a variety of activities or a variety of sources and destinations of input and output may be easier to design and implement using threads.

In an OS that supports threads, scheduling and dispatching is done on a thread basis; hence, most of the state information dealing with execution is maintained in thread-level data structures. There are, however, several actions that affect all of the threads in a process, and that the OS must manage at the process level. For example, suspension involves swapping the address space of one process out of main memory to make room for the address space of another process. Because all threads in a process share the same address space, all threads are suspended at the same time. Similarly, termination of a process terminates all threads within that process.

Thread Functionality

Like processes, threads have execution states and may synchronize with one another. We look at these two aspects of thread functionality in turn.

THREAD STATES As with processes, the key states for a thread are Running, Ready, and Blocked. Generally, it does not make sense to associate suspend states with threads because such states are process-level concepts. In particular, if a process is swapped out, all of its threads are necessarily swapped out because they all share the address space of the process.

There are four basic thread operations associated with a change in thread state [ANDE04]:

1. **Spawn:** Typically, when a new process is spawned, a thread for that process is also spawned. Subsequently, a thread within a process may spawn another thread within the same process, providing an instruction pointer and arguments for the new thread. The new thread is provided with its own register context and stack space and placed on the Ready queue.
2. **Block:** When a thread needs to wait for an event, it will block (saving its user registers, program counter, and stack pointers). The processor may then turn to the execution of another ready thread in the same or a different process.
3. **Unblock:** When the event for which a thread is blocked occurs, the thread is moved to the Ready queue.
4. **Finish:** When a thread completes, its register context and stacks are deallocated.

A significant issue is whether the blocking of a thread results in the blocking of the entire process. In other words, if one thread in a process is blocked, does this prevent the running of any other thread in the same process, even if that other thread is in a ready state? Clearly, some of the flexibility and power of threads is lost if the one blocked thread blocks an entire process.

We will return to this issue subsequently in our discussion of user-level versus kernel-level threads, but for now, let us consider the performance benefits of threads that do not block an entire process. Figure 4.3 (based on one in [KLEI96]) shows a program that performs two remote procedure calls (RPCs)² to two different hosts to obtain a combined result. In a single-threaded program, the results are obtained in sequence, so the program has to wait for a response from each server in turn. Rewriting the program to use a separate thread for each RPC results in a substantial speedup. Note if this program operates on a uniprocessor, the requests must be generated sequentially and the results processed in sequence; however, the program waits concurrently for the two replies.

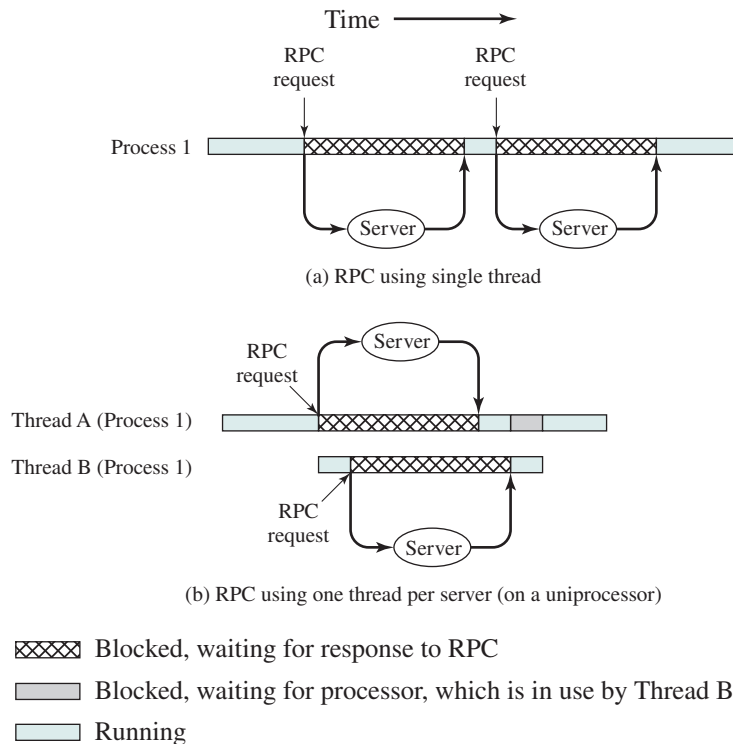


Figure 4.3 Remote Procedure Call (RPC) Using Threads

²An RPC is a technique by which two programs, which may execute on different machines, interact using procedure call/return syntax and semantics. Both the called and calling programs behave as if the partner program were running on the same machine. RPCs are often used for client/server applications and will be discussed in Chapter 16.

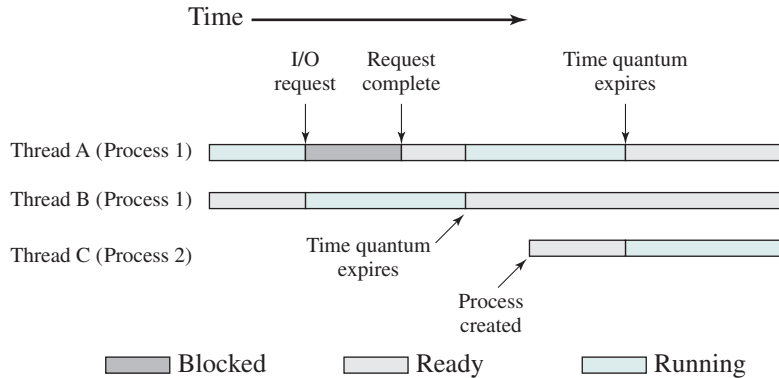


Figure 4.4 Multithreading Example on a Uniprocessor

On a uniprocessor, multiprogramming enables the interleaving of multiple threads within multiple processes. In the example of Figure 4.4, three threads in two processes are interleaved on the processor. Execution passes from one thread to another either when the currently running thread is blocked or when its time slice is exhausted.³

THREAD SYNCHRONIZATION All of the threads of a process share the same address space and other resources, such as open files. Any alteration of a resource by one thread affects the environment of the other threads in the same process. It is therefore necessary to synchronize the activities of the various threads so that they do not interfere with each other or corrupt data structures. For example, if two threads each try to add an element to a doubly linked list at the same time, one element may be lost or the list may end up malformed.

The issues raised and the techniques used in the synchronization of threads are, in general, the same as for the synchronization of processes. These issues and techniques will be the subject of Chapters 5 and 6.

4.2 TYPES OF THREADS

User-Level and Kernel-Level Threads

There are two broad categories of thread implementation: user-level threads (ULTs) and kernel-level threads (KLTs).⁴ The latter are also referred to in the literature as *kernel-supported threads* or *lightweight processes*.

USER-LEVEL THREADS In a pure ULT facility, all of the work of thread management is done by the application and the kernel is not aware of the existence of threads.

³In this example, thread C begins to run after thread A exhausts its time quantum, even though thread B is also ready to run. The choice between B and C is a scheduling decision, a topic covered in Part Four.

⁴The acronyms ULT and KLT are not widely used, but are introduced for conciseness.

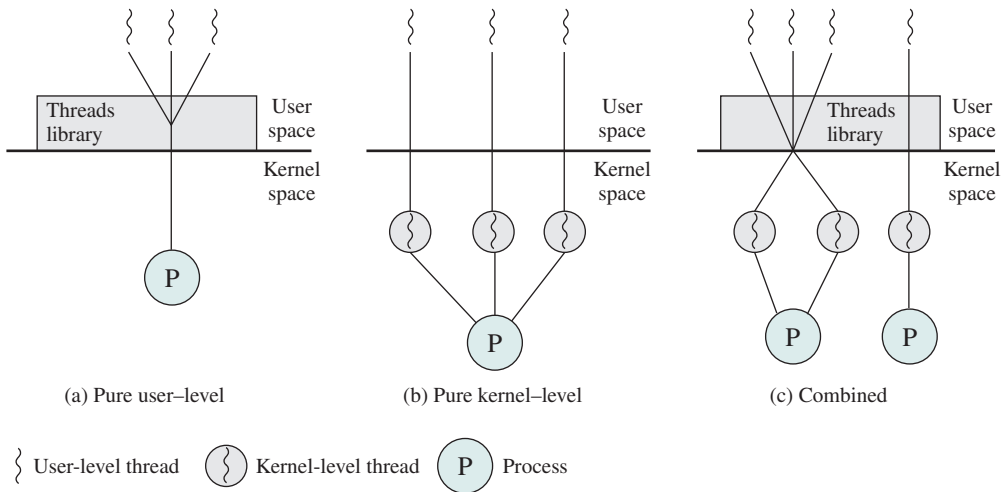


Figure 4.5 User-Level and Kernel-Level Threads

Figure 4.5a illustrates the pure ULT approach. Any application can be programmed to be multithreaded by using a threads library, which is a package of routines for ULT management. The threads library contains code for creating and destroying threads, for passing messages and data between threads, for scheduling thread execution, and for saving and restoring thread contexts.

By default, an application begins with a single thread and begins running in that thread. This application and its thread are allocated to a single process managed by the kernel. At any time that the application is running (the process is in the Running state), the application may spawn a new thread to run within the same process. Spawning is done by invoking the spawn utility in the threads library. Control is passed to that utility by a procedure call. The threads library creates a data structure for the new thread and then passes control to one of the threads within this process that is in the Ready state, using some scheduling algorithm. When control is passed to the library, the context of the current thread is saved, and when control is passed from the library to a thread, the context of that thread is restored. The context essentially consists of the contents of user registers, the program counter, and stack pointers.

All of the activity described in the preceding paragraph takes place in user space and within a single process. The kernel is unaware of this activity. The kernel continues to schedule the process as a unit and assigns a single execution state (Ready, Running, Blocked, etc.) to that process. The following examples should clarify the relationship between thread scheduling and process scheduling. Suppose process B is executing in its thread 2; the states of the process and two ULTs that are part of the process are shown in Figure 4.6a. Each of the following is a possible occurrence:

1. The application executing in thread 2 makes a system call that blocks B. For example, an I/O call is made. This causes control to transfer to the kernel. The kernel invokes the I/O action, places process B in the Blocked state, and

Doe John

YOUR ADDRESS

000000000000 | ✉️ thisis@fakeemail.com | 🐙 github.com/themagicalmammal | 🔗 linkedin.com/in/themagicalmammal

Personal Profile

A University of Leeds graduate student who is enrolled in the Data Science and Analysis programme. Dedicated to software engineering, having two or more years of expertise, and specialising in full-stack web development, algorithms, and machine learning. Searching mostly for Machine Learning, Data Engineer, and Python Development positions.

Education

University of Leeds

Leeds, UK

MSc in Data Science and Analysis

Sept 2021 - Current

- Head of School International Excellence Scholar
- Working as a PAL Mentor
- **Courses:** Data Science, Programming for Data Science, Knowledge Representation and Reasoning, Statistical Theory and Methods, Learning Skills through Case Studies, Artificial Intelligence, Machine Learning, Statistical Learning

Madhav Institute of Technology and Science

Gwalior, India

BTech in Information Technology

May 2016 - May 2020

- Graduated with Distinction
- Selected for Industrial Visit among the top of the class

Atomic Energy Central School

Indore, India

High School

Apr 2014 - Apr 2016

- Passed with Distinction
- Specialised in Physics, Chemistry, and Maths with Computer Science

Work Experience

Indian Institute of Technology

Indore, India

Software Engineer

Sept 2020 - Aug 2021

- Collaborated with a four-person team to develop a CNN model that utilised YOLO as a foundation to improve the accuracy of ambient lighting conditions in the Mobile net architecture.
- Automated and optimised the data handling process for traffic signs, working with Ubuntu 20.04 did shell scripting, and employed other Linux tools.
- Significantly boosted the model's accuracy by 60%, which was yielding an accuracy of 70-78% under Indian street light. The older models, had an accuracy of 10-15%.
- **Technical Skills:** Python with PyTorch, NumPy, Matplotlib, Pandas, Scikit-learn, C++, Ubuntu Linux, Linux tools, Apt, Scripting, Git.
- **Soft Skills:** Teamwork, Time Management, Communication, Presentation skills.

University Projects

The Relation between GDP and IMR

Leeds, UK

University of Leeds

Feb 2022 - Apr 2022

- Analysing data from 1960 to 2020 and discover patterns in the data that show trends between Gross domestic product per capita, Infant Mortality rate, Literacy Rate, and Industrial Development.
- Study cross-sectional, longitudinal, and among various countries to find variations in trends and get the historical data to back up the change in the trend of our data.
- Specific case studies were traced back to trends seen in the UK and how they affected the GDP and IMR.
- **Technical Skills:** Tableau, Overleaf, LaTeX.
- **Soft Skills:** Time Management, Teamwork, Presentation skills, Report writing.

IPL Analysis

Leeds, UK

University of Leeds

Nov 2021 - Dec 2021

- Analysing data from 2008 to 2015 to discover patterns such as trends, correlations, and probabilities.
- Finding the differences between the best teams and players in various fields, as well as their performance on the field and the opponents.
- **Technical Skills:** Python with Pandas, matplotlib, Seaborn.
- **Soft Skills:** Presentation skills, Leadership, Teamwork, Logical Thinking.

Study of the behavior of Serial Killers

Leeds, UK

University of Leeds

Oct 2021 - Dec 2021

- Study of serial killers behavior with various motives such as convenience (did not want children or spouse), enjoyment, power, escape, or avoiding arrest.
- Finding patterns in their starting and ending ages (when they are caught) and other factors of killing, as well as how it varies with different motives.
- **Technical Skills:** R with ggplot2, tidy, R Markdown.
- **Soft Skills:** Report writing, Logical Thinking, Critical Thinking.

Skills

Programming Python (Pandas, PyTorch, NumPy, Scikit-learn. etc.), R(ggplot2), PHP, C/C++, HTML/CSS, JavaScript, SQL.
Miscellaneous Linux, Shell (Bash/Zsh), \LaTeX (Overleaf/R Markdown), Tableau, Microsoft Office, Firebase, Git.
Soft Skills Time Management, Teamwork, Problem-solving, Documentation, Engaging Presentation.

Achievements

2019	319/340 , Graduate Record Examinations (GRE)	India
2018	Elite , DBMS NPTEL Exam	India
2018	Elite , C NPTEL Exam	India
2019	Winner , Smart India Hackathon (SIH)	India
2014	Level 3 , TablaNiketan Exam	India
2014	Level 1 , National Talent Search Exam (NTSE)	India

Publications

JOURNAL ARTICLES

Incentive-based resource assignment and regulation for collaborative cloud services in community networks

Amin M Khan, Ümit C. Büyüksahin, Felix Freitag

Journal of Computer and System Sciences 81.8 (Dec. 2015) pp. 1479–1495. 2015

Cloud services in the Guifi.net community network

Mennan Selimi, Amin M Khan, Emmanouil Dimogerontakis, Felix Freitag, Roger Pueyo Centelles

Computer Networks 93.P2 (Dec. 2015) pp. 373–388. 2015

CONFERENCE PROCEEDINGS

Prototyping Incentive-Based Resource Assignment for Clouds in Community Networks

Amin M Khan, Umit Cavus Buyuksahin, Felix Freitag

28th IEEE International Conference on Advanced Information Networking and Applications (AINA 2014), 2014, Victoria, Canada

Interests

Cooking I love cooking. I am an expert in most Indian-style cooking, enjoy baking and making my own pizza.

Linux Since 2017, I have been in love with Linux. I recently switched to Mac OS, which feels like a premium of Linux.

Technical Writing I write descriptive blogs about Linux some of them are on my GitHub and Medium.

Art I have always enjoyed drawing since I was a child. Recently, I have shifted to digital art mostly on Deviantart.

Video Games I always had the gaming gene. I mostly play on my phone and occasionally on my switch.

Swimming I started swimming when I was 12. Going through submerged things while holding my breath was the most exciting part.

Languages

English Professional proficiency

Hindi Native proficiency

References available upon request.