International Journal of Electrical, Electronics and Computer Engineering **13**(1&2): 71-73(2024)

ISSN No. (Online): 2277-2626

Role of Threads in Operating System

Saurabh Sharma*, Sidharth Sharma, Lotus Student
Department of School of Computer Science and Engineering
Govt. P.G College Dharamshala, Himachal Pradesh Technical University (HPTU), India.

(Corresponding author: Saurabh Sharma)*

(Received: 13 February 2024, Accepted: 23 April 2024)
(Published by Research Trend, Website: www.researchtrend.net)

ABSTRACT: **This paper deals with threads used in an operating system. We discuss about working of multithreading system, motivation to implement thread concept. The first part of paper explain you background to implement thread. And the second part emphasizes the concept of thread is an operating system.**

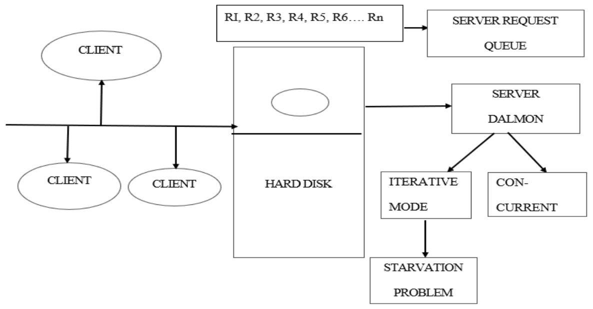
Keywords: Threads, fork, multithreading, effective, user mode, concurrency, OS.

INTRODUCTION

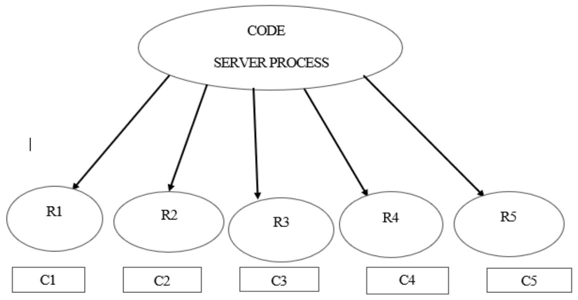
Thread is a light weight process (LWP) and we say that thread is also a process or every process have its own Process control block (PCB). By default -each process will have a single thread and we have two types of threads kernel level or user level thread. Thread will do smart work it will have less resources with it. A thread is like another copy of a process that executes independently. Each thread may run over different part of a program or each thread has a separate stack for independent function calls.

BACKGROUND OF STUDY

In client server environment client will send the requests and then request come into the queue and there is server process in operating system which is suppose to satisfy these Request initially these server or processes were design to operate in iterative mode (sequential mode). It Satisfy the request one after another that it will take first satisfy the request totally by reserving the data from the hard disk and formulating the packet on PDU protocol data unit and sending over communication protocol then take second request do same work again and again and if you satisfy the request sequentially one after another naturally you will uncounted the problem of (starvation). To remove the problem of starvation people went for (CONCURRENT SERVER) (Silberschatz *et al.,* 2018)



Concurrency in server design was first approach by using (MULTIPROSESS APPROACH). When there is five request pending in the queue you want to currently satisfy those 5 requests by create 5 independent process called child processes.



FORK is system call used to create a child process these child processes are an exact replica (copy) of original process. We have 5 child process and server will assign request to the process and we have one CPU and we use round robin may be some Time quantum=6 sec and operating system first schedule C1 because all the process are in ready queue after 6 sec C1 permit and C2 schedule. We schedule in a such a way that everybody will be get impression my request been served. Initially using multi-process approach through creating by multiple child process using fork system call all child process have same structure. Every child have its own code section, own data segment, own heap section, own stack. So we have 5 (copy of code, data, heap, stack) separately in memory. Independent for every process it is traditional concurrency in server design. This is how concurrency was achieved initially before the concept of thread. This approach of concurrency was also for certain amount of time but when network scale increase, load increase then people realizing the concurrency is used by multi-process approach are a big drawback. The code instruction of server process are copied in all child processes. If code have 100 instruction so every child process having same 100 instruction when instruction have same functionality this is worry point to developer/researcher when functionality is same why should be have 5 copy of code section. It is wastage of memory so we introduced the concept of multithreading (Khaleel, 2022).

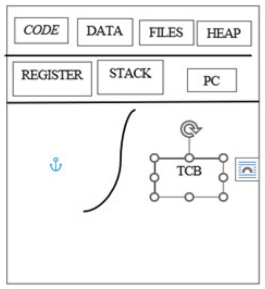
MULTITHREADING CONCEPT

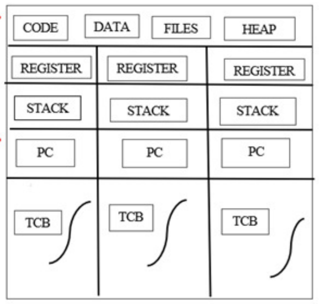
We implement the concept of multithreading to overcome the drawback why should we have replicas multiple copy and waste the resources (memory). Let us have only one copy of address space, one copy of code, one copy of data and all other resources which we can have one copy those which we cannot have we will separate it (Butenhof, 1997). This was the motivation this to introduced the concept of thread. Every thread of the process or sub process with in the main process will basically share all the common resources then it called light weight. There is no independent code section of thread no common data. The benefit of multithreaded programming can be broken down into five major categories (Galvin, 2015).

a) Responsiveness
b) Resource sharing
c) Economy
d) Scalability
e) Improved performance due to less context switch overhead |TCB| < |PCB|

Main motivation for having multithreading

1. Resource Sharing





Why thread shown like a curvy line why not straight line:

The line indicate in our code section when we execute instruction 1 then 2 but now this could be possibility that my instruction 2 may be a branch instruction go to like a function call this instruction 2 is a branch instruction means flow of control go somewhere else then sequentially then branch during the execution instruction of a program their may be a branch that branch indicate as a curve. If we say there is no branch in my code section so thread of control are straight line this line indicate the order in which the instruction are executed in code section (Tanenbaum and Bod 2014).

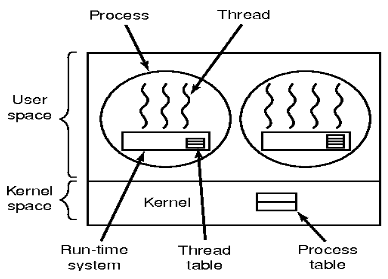
1.User -level-threads -Threads that are created and managed by the user level without any support of operating system. Benefit of user level thread-

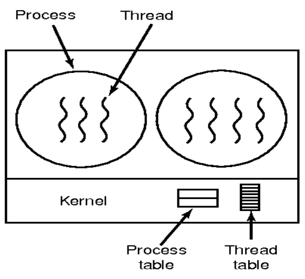
1. Transparency
2. Flexibility

2. Kernal level threads- scheduler can decide to give more time to a process having small number of threads. Kernel-level threads are especially good for application that frequently block.

Disadvantages- The kernel-level are show(they involve kernel invocation).

Overheads in the kernel (since kernel must manage and schedule threads as well as processes. It required a full thread control block (TCB) for each thread to maintain information about threads),





CONCLUSION

This paper has provided a comprehensive overview of threads in operating systems, exploring their functionality, motivation for implementation, and the transition from traditional multi-process concurrency to the concept of multithreading. Beginning with the background of implementing threads, we discussed the challenges faced in a client-server environment, particularly the issue of starvation, and how concurrency was initially achieved through the multiprocess approach using system calls like fork. However, as networks scaled and loads increased, it became evident that this approach had drawbacks, particularly in terms of resource utilization and memory wastage due to redundant code sections in each child process. Hence, the concept of multithreading was introduced to address these challenges by allowing threads within a process to share common resources, leading to more efficient resource utilization and improved economy. The main motivations behind adopting multithreading were resource sharing and cost-effectiveness. By having threads share common resources while maintaining separate execution paths, multithreading offered a more lightweight alternative to traditional multi-process concurrency, leading to better performance and scalability. In conclusion, the adoption of multithreading has significantly contributed to the efficiency and performance of modern operating systems, allowing for better resource utilization, improved concurrency, and scalability. However, it's essential to consider the appropriate use of threads based on the specific requirements and characteristics of the application or system in question. Overall, multithreading remains a fundamental concept in operating system design and continues to play a crucial role in enhancing system performance and responsiveness.

REFERENCES

Butenhof, D. R. (1997). Programming with POSIX Threads. Addison-Wesley.

Khaleel (2022) https://www.pw.live/study/batches/study

Galvin (2015). Operating System. Willy Publication, 2015.

Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). Operating System Concepts (10th ed.). Wiley.

Tanenbaum, A. S., & Bos, H. (2014). Modern Operating Systems (4th ed.). Pearson.