**MULTI THREADING**

Parallel Programming

Necessity and Explanation of Concurrency in Programming

Introduction to concurrency – Running certain things parallelly. It refers to the ability of a system to perform multiple tasks simultaneously or handle multiple operations within overlapping time frames.

In programming, it allows an application to manage numerous tasks or execute multiple operations concurrently, rather than strictly at one time.

Necessity and benefits of concurrency

1) Responsiveness: Concurrency allows applications to remain responsive to user actions even when performing long-running tasks in the background.

2) Utilizing Multi-Core processors: Most modern MC allowing them to execute multiple instructions at once

3) Efficient resource Utilization: Concurrency allows programs to better utilize available system resources by avoiding idle times

4) Scalability: Concurrency enables applications to scale better, especially in server environments where multiple clients or tasks must be handled simultaneously.

5) Improved throughput: Throughput or the amount of application can process in a given amount of time.

Note : Light weighted Threads [LWT] – Different processes are attached using the thread

What is concurrency programming?

- Multithreading: Multiple threads within a single process are created to perform different parts of a task or handle multiple tasks concurrently.

Threads share the same memory space, which allows for efficient communication but requires careful synchronization to prevent race conditions.

Note : If it is acting on the same data then we need to avoid race condition

- Multiprocessing: Separate processes are created to handle different tasks concurrently.

Each process has its own memory space, which is safer but require inter-process communication methods like pipes or shared memory for collaboration.

IPC – Inter Process Communication

- Asynchronous programming: Non-blocking functions are used to handle tasks such as I/O operations without blocking the main thread, allowing other operations to proceed concurrently.

This is common in languages like JavaScript and Python

Concurrency vs Parallelism

- They are related but distinct

- C is about dealing with multiple tasks at once or having multiple tasks in progress

- P is about executing the tasks truly simultaneously

Methods to achieve concurrency in C

1) Multithreading using POSIX threads(pthreads):

- Thread is a smallest process of execution

- Each thread has its own stack, registers and program counters

- Threads within the same process can easily communicate and share data since they share the same memory space.

- Process = Process context + code, data and stack

- Alternate view of process :

Process = thread + code, data, kernel context

- A process with multiple threads

Each thread has its own logical control flow(sequence of PC values)

Each thread shares the same code, data and kernel context

Each thread has its own thread id (TID)

2) Using fork()

3) Asynchronous I/O

4) Atomic Operations

5) Semaphores

6) Message queues

Threads vs Processes

Similarity:

1) Each has its own logical control flow

2) Each can run concurrently

3) Each is context switched

Difference:

1) Threads share code and data, processes (typically) do not

2) Threads are somewhat expensive

POSIX thread interface

1) Creating and reaping threads

Pthread\_create, pthread\_join

2) Determining your thread ID

Pthread\_self

3) Terminating threads

Pthread\_cancel, pthread\_exit

Exit [terminates all threads], return [terminates current threads]

4) Synchronizing access to shared variables

Pthread\_mutex\_init, pthread\_mutex\_[un]lock

Pthraed\_cond\_init, …….

Pthread on success return 0, on error returns error number

gcc threads.c -lpthread – For execution( include the library – shared library)

/\*

thread demo

\*/

#include <stdio.h>

#include <stdlib.h>

#include <unistd>

#include <pthread.h>

void printHello();

int main()

{

pthread\_t tid;

int status = 0;

printf("\nIn the main function\n");

printf("\mI am in Program/Process,Main thread\n");

status = pthread\_create(&tid,NULL,printHello,NULL);

pthread\_join(tid,NULL);

printf("\nAfter completing (Main)\n");

return 0;

}

void printHello()

{

printf("\nHello World\n");

}

O/P

user50@trainux01:~/practice/try/day14$ ./a.out

In the main function

mI am in Program/Process,Main thread

Hello World

After completing (Main)

---------If pthread is not used then Hello world is not printed

NOTE : Column size should be known in an array

If we want to do array as reference then in main method it should be taken as func(a) while invoking func method and in this method we need to catch in the format of func(int a)

If we wnt the base address of the base address then at main method while invoking it should be func(&arr) and in this method we need to catch in the format of func( \*\*arr).

So that & and\* are nullified by leaving only one \* with arr which results in the base address

Why semicolon at the end of do while loop and structures --- To execute it only for one time

Why pthread\_mutexattr\_destroy() ----- to set the object referenced by attr to an invalid value.