**DAY – 14**

**THREADS:**

Parallel Programming

The link is established through threads.

LWT – Light Weighted Threads

**NECCESSITY OF CONCURRENCY:**

Concurrency refers to the ability of a system to perform multiple tasks simunteneously or handle multiple operations within overlapping time frames.

In application it allows to manage tasks.

1**. Responsiveness:**

It allows application to remain responsive to user even when performing long tasks in the background

For e.g.: webserver can handle multiple client requests concurrently without bocking other users

**2.** **Utilizing multi core Processors:**

Execute multiple instructions at a time

**3.** **Efficient Resource utilization**

Avoid idle times to better utilize available system resources.

**4. Scalability**

Efficient and effective usage

**5. Improved throughput**

By overlapping tasks especially in systems where tasks don’t always require CPU like I/O bound

**Concurrency in Programming handled by:**

**1. Multithreading:**

· Multiple threads within a single process are created to perform different tasks concurrently.

· Threads share same memory space which allows efficient communication but requires careful synchronization to prevent race conditions

**2. Multiprocessing:**

· Separate processes are created to handle different tasks concurrently.

· Each process has its own memory space. Which is safer but requires inter process communication methods like pipes or shared memory for collaboration

E.g.: Kho Kho

· POSIX threads(pthreads) creates and manages threads within a process

**3. 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 languages like Python, Java Script

· Fork() creates separate process with individual memory spaces in multiprocessing

* Concurrency is dealing with multiple tasks at once or having multiple tasks in progress
* Parallelism is executing multiple tasks truly simultaneously achieved by multiple CPU cores

Atomic Operations: Enables simple, thread-safe operations without full mutex overhead

Semaphores: Controls access to shared resources by multiple threads

* Each thread has its own stack, registers, program counter
* Threads within the same process can easily communicate
* Threads are lighter in terms of system process than full resources

· **Traditional View of Process:**

Process= Process context +code, data and stack

· Alternate view of Process:

Process: thread+ code, data and kernel context

· A process with multiple threads

* Each thread has its own logical control flow
* Each thread shares the same code data and kernel context
* Each thread has its own thread id

**Threads and Process similarities**

* Each has its own logical control flow
* Each can run concurrently
* Each is context switched

**Threads and Process differences:**

* Threads share same data and process do not
* Threads are less expensive than process
* Shares memory while process do not
* Threads are fast to create and lightweight
* Thread terminates when function finishes by exit call while process terminate independently.
* Threads are easy to communicate while in process we need inter process communication.

**Pthreads**

* Creating and reaping threads: pthread\_create, pthread\_join
* Determining thread id: pthread\_self
* Terminating threads: pthread\_cancel, pthread\_exit
* Exit(terminates all threads), return( terminates current thread)
* Synchronizing access to shared variables: pthread\_mutex\_init, pthread\_mutex\_[un]lock
* pthread\_cond\_init, pthread\_cond\_[timed]wait

**THREAD DEMO**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <pthread.h>

#define ITNERS 1000000

void printHello();

int main()

{

pthread\_t tid;

int status=0;

printf("\nIn the Main Function\n");

printf("\nI am Main Program/Process/MainThread\n");

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

// pthread\_join(tid,NULL);

sleep(1);

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

return 0;

}

void printHello()

{

int i;

printf("\nHello World\n");

for(i=0;i<ITNERS;i++)

printf("\n\ti=%d",i);

}

**CODE2:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <pthread.h>

#include <string.h>

#include <time.h>

typedef unsigned long long ULL;

struct varg

{

ULL start;

ULL end;

};

typedef struct Employee

{

int id;

char name[20];

float sal;

}EMP;

ULL sumOdd = 0;

ULL sumEven = 0;

void \*findEven(void \*vargs)

{

ULL i;

ULL start,end;

struct varg \*ptr = (struct varg \*)vargs;

start = ptr->start;

end = ptr->end;

printf("\n%lld \n%lld\n",ptr->start,ptr->end);

for(i = start;i<=end;i++)

{

if((i & 1) == 0)

{

sumEven += i;

// sleep(1);

}

}

printf("\nIn Thread Sum Even = %lld\n", sumEven);

}

void \*func(void \*vargs)

{

struct varg \*ptr = (struct varg \*)vargs;

// printf("\nvargs = %d\n",\*ptr);

// char \*Name = (char \*)vargs;

// printf("\nvargs = %s\n",Name);

// struct varg \*ptr = (struct varg \*)vargs;

// printf("\n%d\n%d\n",ptr->a,ptr->b);

printf("\n1st arg = %llu",ptr->start);

printf("\n2nd arg = %llu",ptr->end);

}

void \*printEmp(void \*vargs)

{

EMP \*ptr = (EMP \*)vargs;

printf("\nID: %d",ptr->id);

printf("\nName: %s",ptr->name);

printf("\nSalary: %f",ptr->sal);

}

int main()

{

int a=10;

char name[20] = "Bhima";

struct varg arg;

EMP e;

pthread\_t tid;

struct timespec start, end;

double time\_taken;

// arg.a = 101;

// arg.b = 201;

arg.start = 110;

arg.end = 1900000000;

e.id=101;

strcpy(e.name,"Bhima");

e.sal = 10001.1;

clock\_gettime(CLOCK\_MONOTONIC, &start);

// pthread\_create(&tid, NULL, func,&arg);

pthread\_create(&tid, NULL, findEven,&arg);

// pthread\_create(&tid,NULL,printEmp,&e);

// Stop clock

pthread\_join(tid,NULL);

clock\_gettime(CLOCK\_MONOTONIC, &end);

// Calculate time taken in seconds

time\_taken = (end.tv\_sec - start.tv\_sec) + (end.tv\_nsec - start.tv\_nsec) / 1e9;

printf("Time taken by function: %f seconds\n", time\_taken);

printf("\nSum Even = %lld\n",sumEven);

pthread\_exit(NULL);