21 Jan 2025 - Operating Systems - II - Week 03

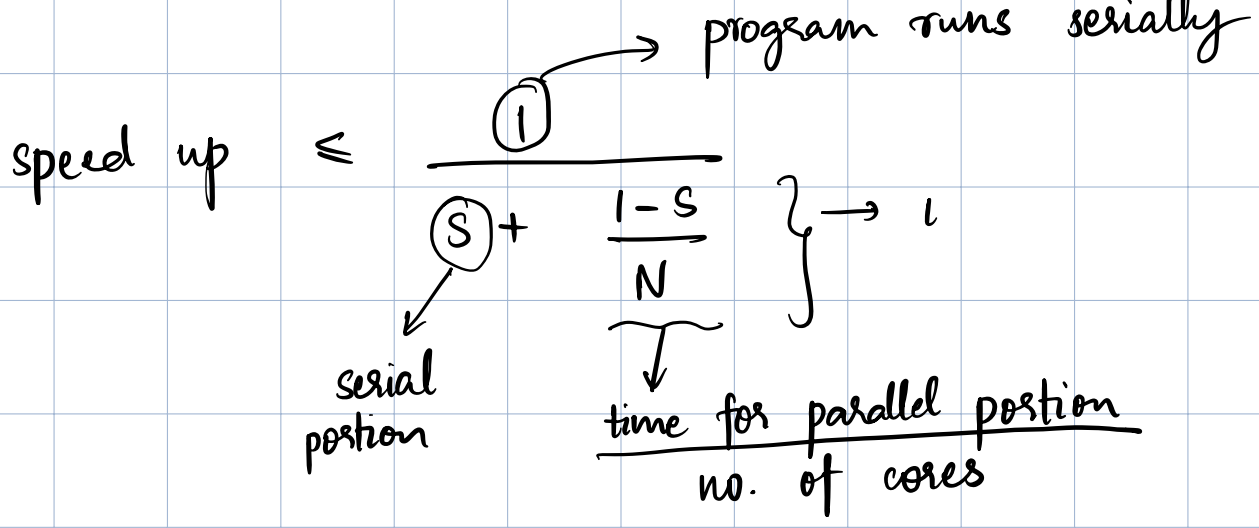
Data and Task parallelism ⇝ distribution among cores

→ a combination is used most of times

→ data parallelism ⟶ good for GPUs (task same)

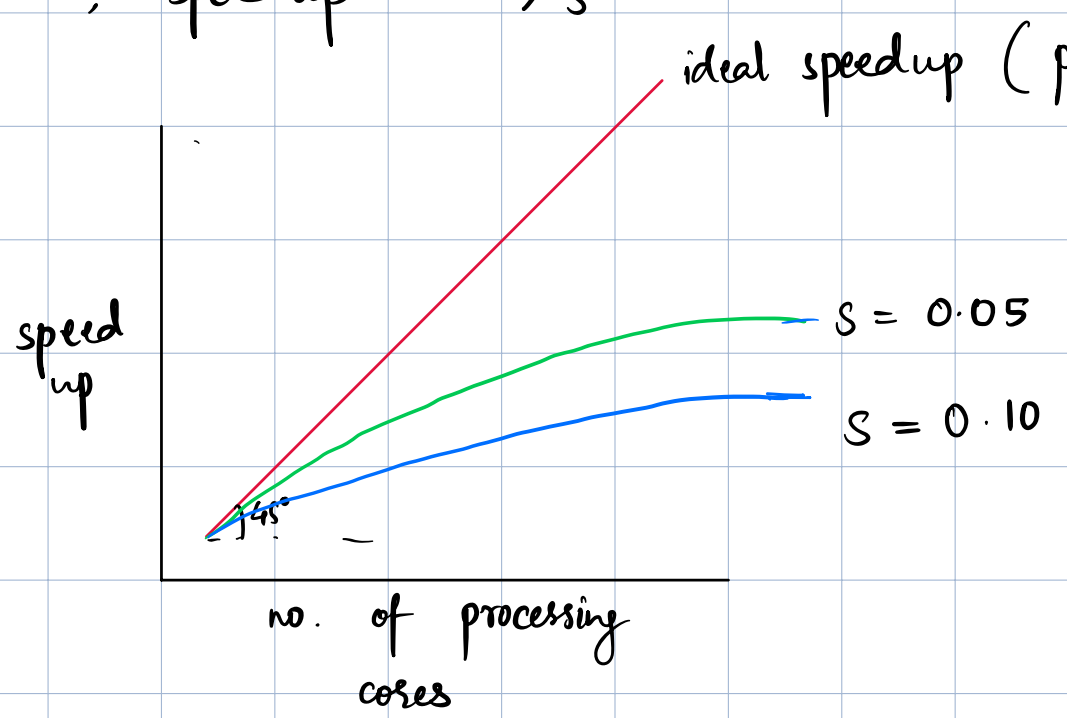
logical cores later

→ Amdahl's law



→ serial portion creates a disproportionate effect on perf gained by adding aditional cores.

→ $N \to \infty$, speedup → $1/s$



→ applies for CPUs (not GPUs)

o Sudoku assignment

core of O.S → makes system calls on behalf of application programs

User threads and Kernel threads

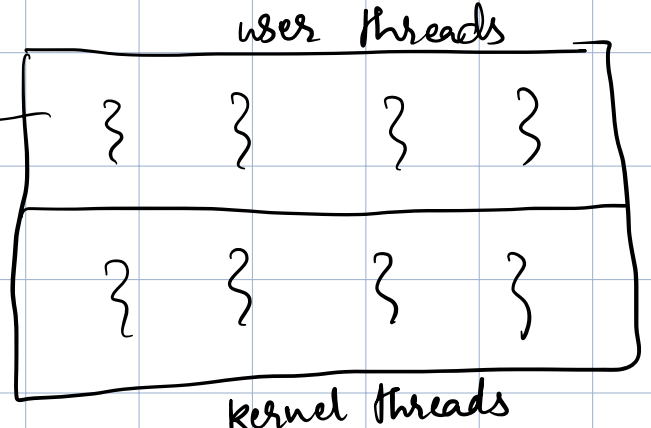
↓

supported by Kernel

different thread libraries for diff O.Ses

→ pthreads → Linux

→ C++ threads



user space

can only access user space memory

for accessing kernel, make system call

e.g printf internally makes system call

→ Why?

→ security

→ bug in user code can affect others

Multithreading models

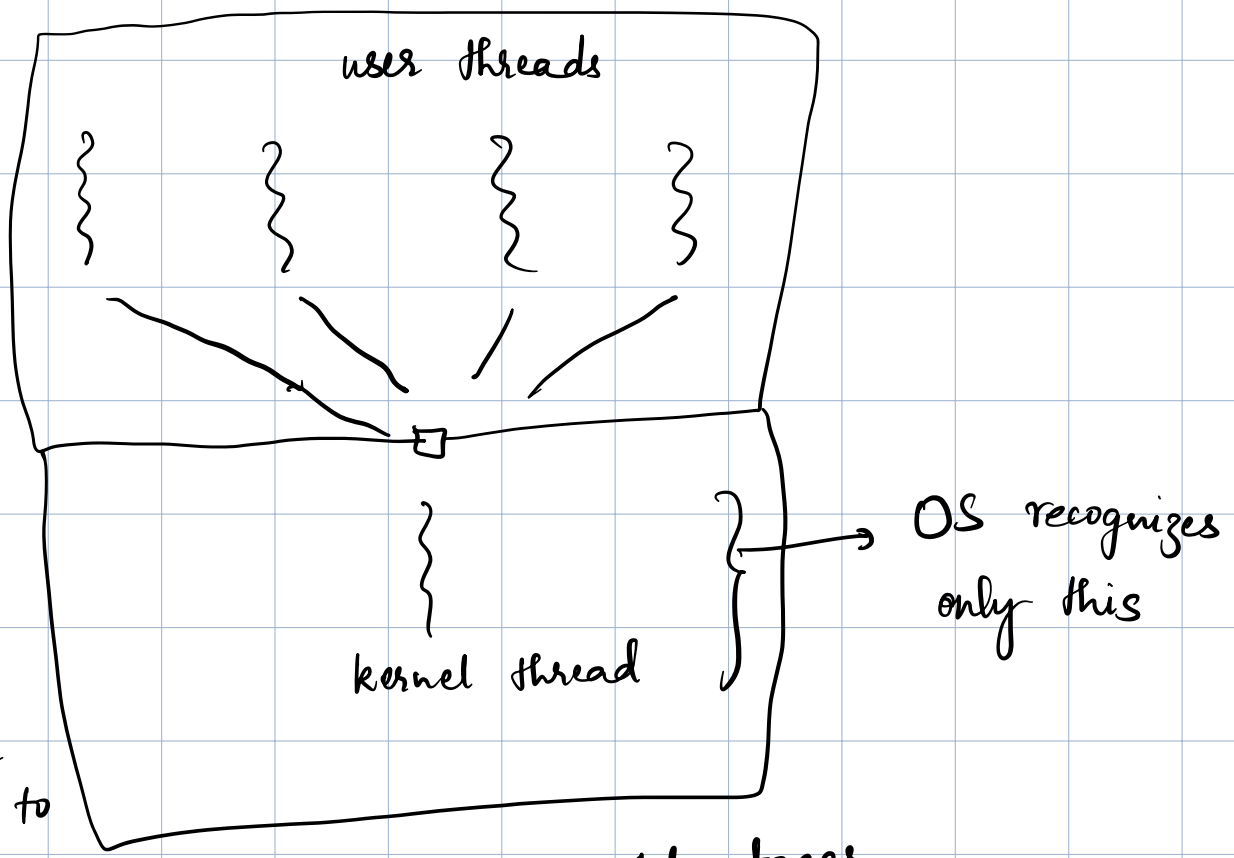
Many - to - one :

→ no parallelism concurrent } can run even on single core

→ Many user-level threads mapped to single kernel thread

→ One thread blocking causes all to block

not able to execute further with due to lack of data or due to scheduling algorithm



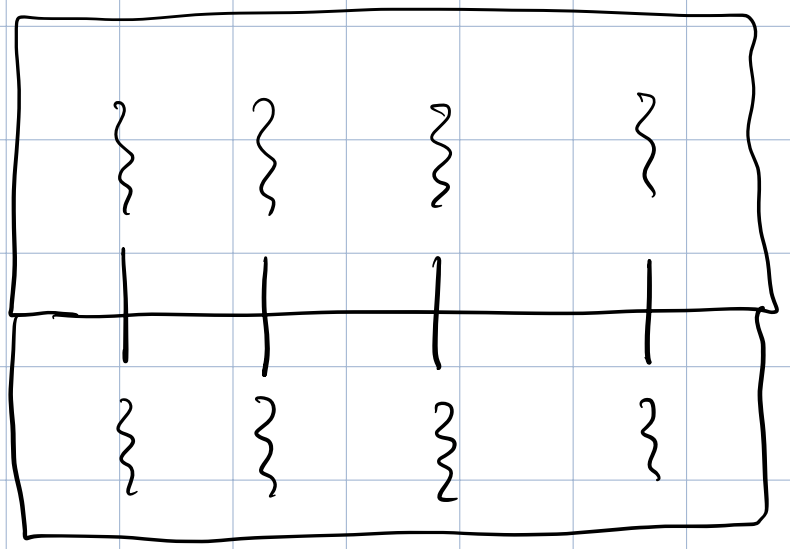
Advantages

??

One - to - one ⇝ pthread_create

→ Windows, Linux

→ No. of threads per process sometimes restricted due to overhead (Not in Linux)

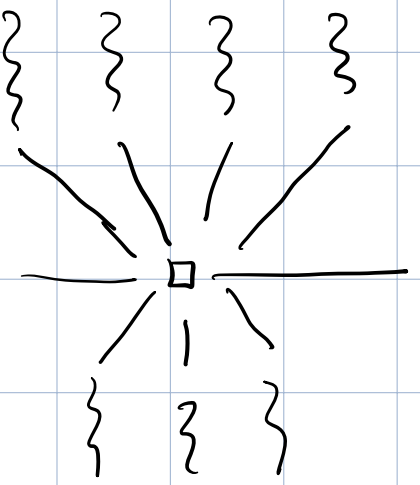


→ disadvantage: one process can create many threads } → advantage for many to one

Many-to-Many

best of both but
not very common

→ overhead in
thread management



later
scheduling
across multiple
cores

Two-level model

→ always mapped to same
kernel thread

→ You can create kernel threads (kthread) directly → but you cannot test it without an application program

→ drivers as programs attached to kernel
↓
interact with kernel

scanf → kernel → USB interface of keyboard → keyboard
← ← ← key press

Thread libraries → slides

pthreads

→ specification → like documentation → ANSI standard
— states behaviour
gcc supports

not implementation

Java, C#, JS, python } → platform independent

C → a.out, .exe

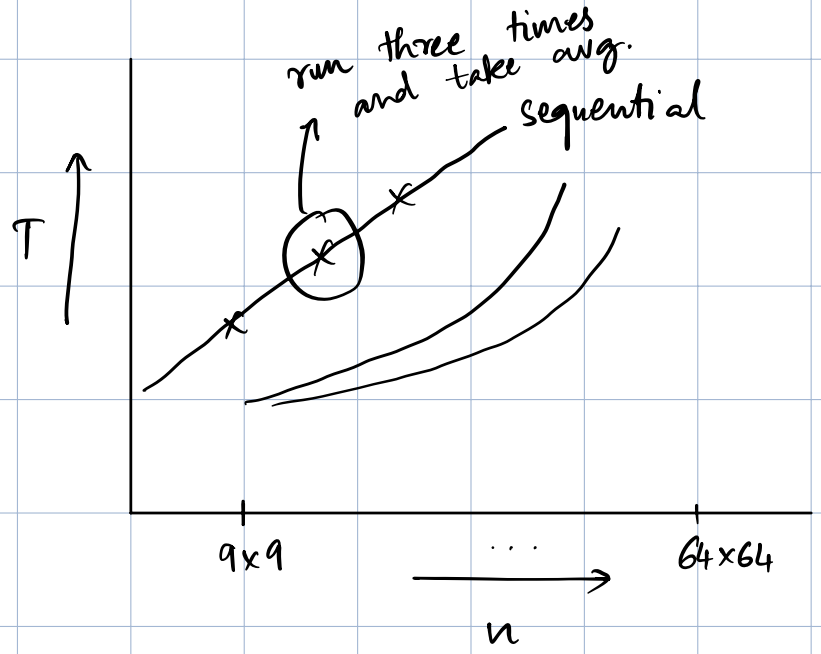
Java → not an executable intermediate state (byte code)

22 Jan 2025

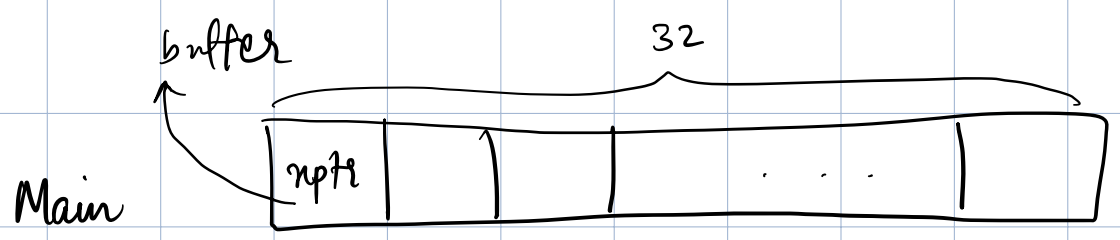
Assignment

2 plots → each plot → 3 curves
mixed, chunk, sequential

Expt 1 :



use
cin, cout



— Thr 1

— Thr 2

:

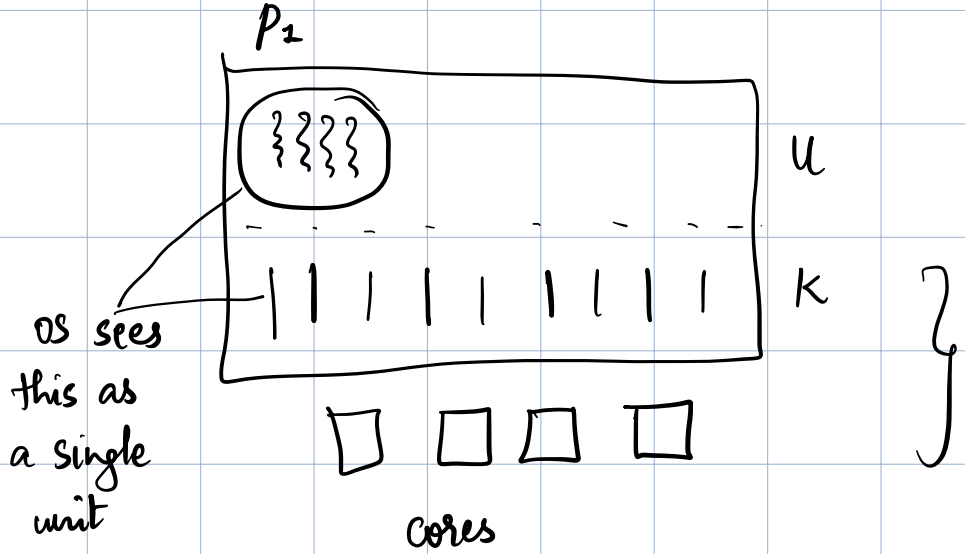
32

} create using for loop
p_thread_create

timer : start bef pthread_create
stop after join

Many-to-one : how does one thread getting blocked blocks others?

— OS sees only kernel level threads



Thr 1 : scanf

time multiplexing

no. of threads ↑

⟹ context switching, etc.

perf ↓

Implicit Threading ⟶ not in C++, provided by other languages

→ Create threads before ⟶ they wait in a pool (sleep)

→ faster ; creating threads comes with an overhead

Java threadpool → threads remain idle when task is done.

→ tasks to be performed independently (separately) from creation

→ easy for programming

→ flexibility reduces

Fork – join parallelism

Open MP → set of compiler directives

→ compiler manages the thread

```
#include <omp.h>
#include <stdio.h>
   C code sequential

#pragma omp parallel
{  printf("moo")
}
...
```

```
# pragma omp parallel for
for (i = 0 ; i < N ; i++) {
    c[i] = a[i] + b[i]
}
```

caching ??

Grand central dispatch → macOS, iOS

→ ^ { printf (...); }

^{}

Blocks ↙ assigned to a thread pool

→ Two types of dispatch queue
- serial
- concurrent

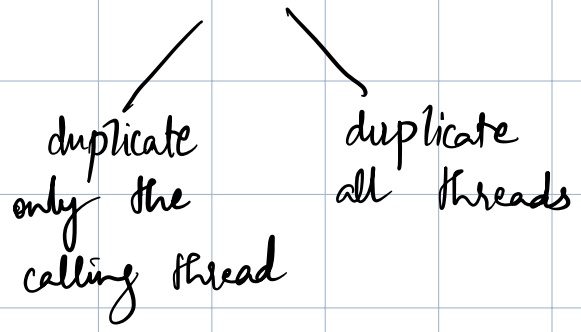
Intel Thread Building Block (TBB) → opensourced

→ threads are implicitly created

→ very helpful for p_threads

Threading issues

Semantics of fork( ) and exec( )



Signal handling (like interrupts)

→ default sigle

→ user defined signal handler

standard UNIX fn for delivering a signal

→ kill

int kill (pid_t p1, int sig)

if p_2 is multithreaded

which thread should receive

* pthread_kill

→ windows does not explicitly provide support for signals → APC

* thread - cancellation
  - searching done in 1st thread
  - quitting a tab in browser while it's loading

- asynchronous → deprecated → can leave the system in an inconsistent state
  - immediate termination
- differred
  - target thread checks periodically
  - better
  - bank transaction
  - file moving
  - default cancellation occurs only when thread reaches cancellation points → system defined
  - flag is raised ⇒ do clean up and terminate

→ You can also disable thread cancellation

- Linux implements cancel internally using signals
  kill

→ cleanup handler ⇝ user defined

Thread Local Storage

↳ similar to static

static — within function — local — global

```
Thr_1(){          | Thr_2(){          | Main {
  TLS e1;         |   e2              |   ...
  ...i;           | }                 |   thr_1();
  fn1()           |                   |   ...
}                 |                   |   thr_2();
                  |                   |   ...
  fn 2()          |   fn 2()          | }
```

local to thread 1 and all its functions

l1 → visible to fn 1, not to fn 2, main

i → not visible to fn 1

Why? Thread local functionalities

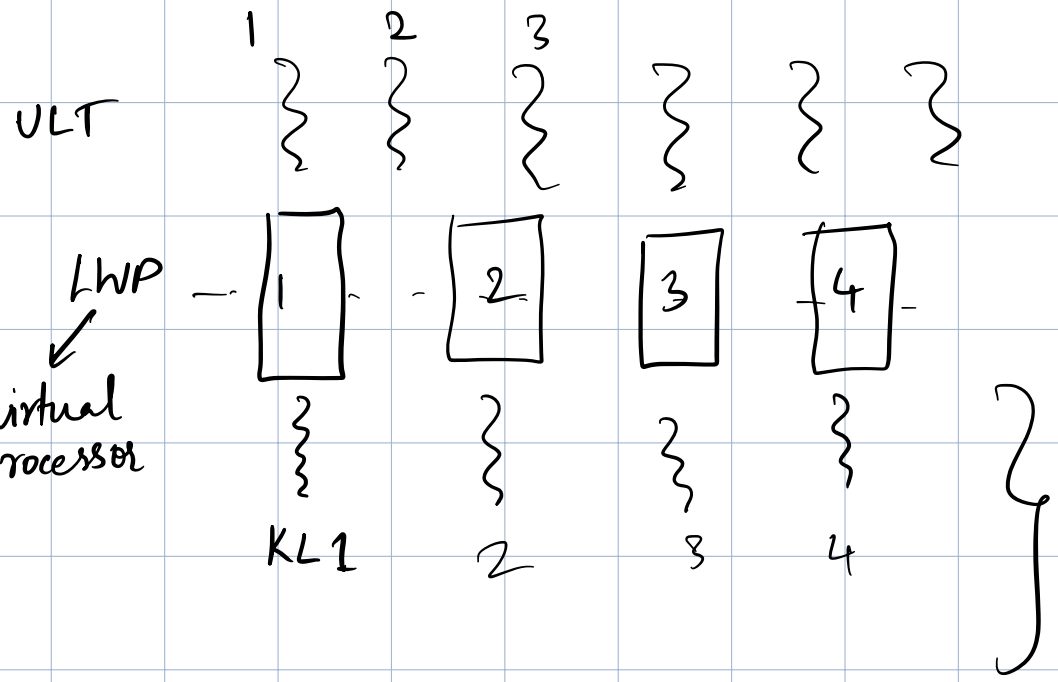
```
static __thread int threadID;   // gcc
```

Scheduler Activation

M:M and
Two-level models
require communication ⟵ uses thread
to maintain

LWP — data structure

kernel thread



TML → thread management library

OS takes care only on KLT

Linux threads

- Calls threads as task
  - can be a process or a task

- thread creation is done using clone()
  - fork
  - pthread_create

- flags for clone

clone with all flags false