Threading

Programmer develops the threads

Implicit threading

- Creation and management of threads done by compilers and run time libs than programmers

[either System Prog lang the hardware]

(1) Thread pools
(2) Fork-join
(3) Open MP
(4) Grand Central Dispatch
(5) Intel Building Blocks

→ Slightly F

I Thread pools

↳ number of threads in a pool while they await work

Thread pools

innovate achieve lead

- Create a number of threads in a pool where they await work
- Advantages:
  - Usually slightly faster to service a request with an existing thread than create a new thread
  - Allows the number of threads in the application(s) to be bound to the size of the pool
  - Separating task to be performed from mechanics of creating task allows different strategies for running task
    - i.e.,Tasks could be scheduled to run periodically
- Windows API supports thread pools:

```
DWORD WINAPI PoolFunction(AVOID Param) {
   /*
   * this function runs as a separate thread.
   */
}
```

JAVA (factory method) eg: new Task().fork();

static ExecutorService newSingleThreadExecutor()
newFixedThreadPool(int size)
newCachedThreadPool()

[you don't need a class for factory methods]

Fork - join Parallelism

multiple threads are forked and then joined

Fork – Join parallelism

innovate achieve lead

Multiple threads (tasks) are **forked**, and then **joined** OS level with help of syscalls

fork () → no arguments ⇒ implies create new IDS

- fork is an operation whereby a process creates a copy of itself
- The fork() call creates a new child process which runs concurrently with the parent. The child process is an exact copy of the parent except that it has a new process id. The fork() creates concurrency. (immediately after fork)
- On success, the PID of the child process is returned in the parent's thread of execution, and a 0 is returned in the child's thread of execution. On failure, a -1 will be returned in the parent's context, no child process will be created, and *errno* will be set appropriately.

(of fork)

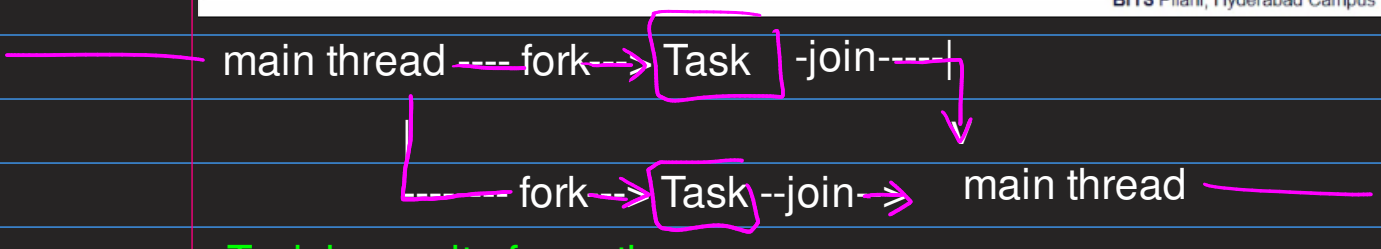
join()

- The join() is called by both the parent and the child. The child process calls join() after it has finished execution. This operation is done implicitly. The parent process waits until the child joins and continues later.
- The join() call breaks the concurrency because the child process exits. The join() inform the parent that child operation is finished.

There are two join scenarios:

- The child joins first and then the parent joins without waiting for any process.
- The parent process joins first and wait, the child process joins, and the parent continues thereafter.

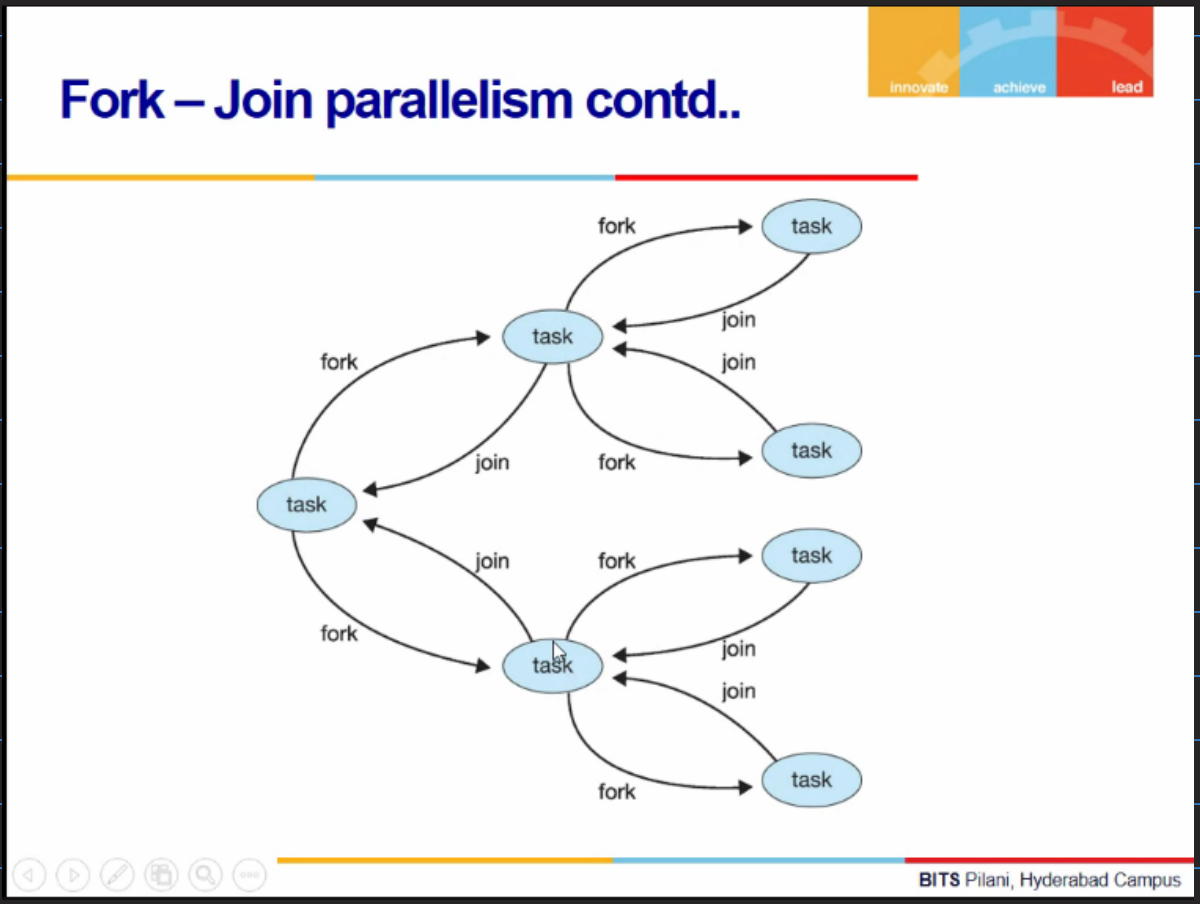
BITS Pilani, Hyderabad Campus



Task is a unit of exection

Algorithm:

First understand the size of the problem:
if it is small enough, don't parallelize it, since
there are overheads in comms and resource
requirements
Parallelize it as per amdahl's law (10% or sth)
else
subtask1 = fork(new Task(subset of problem))
subtask2 = fork(new Task(subset of problem))
...
// don't misunderstand the param inside, it just means
// to implement the task
result = join(subtask1)
result = join(subtask2)
...



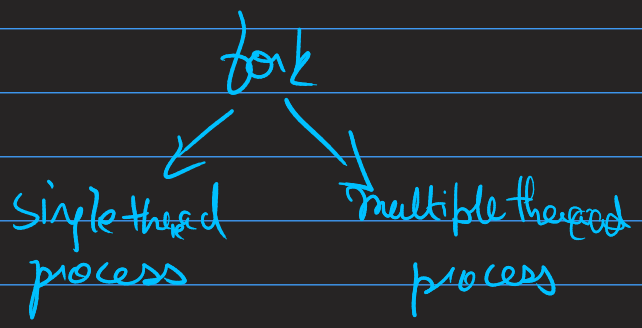
Cg : Binsearch xP

ForkJoinTask abstract base class

RecursiveTask RecursiveTask

OpenMP → an environment (supports for parallel programming in shared mem)

Does fork create a new process or thread



→ Executing things in parallel ⇒ parallelism

→ fork → creates a copy but some stuff is shared yeah

Parallelism → paradigm

parent and child can concurrently act

AUDIO CUT grand central Dispatch

class concluded

Signal handling / semantics next class

T1 portion → till wed class, sep 18th MCQ 15%