Operating Systems CS F372

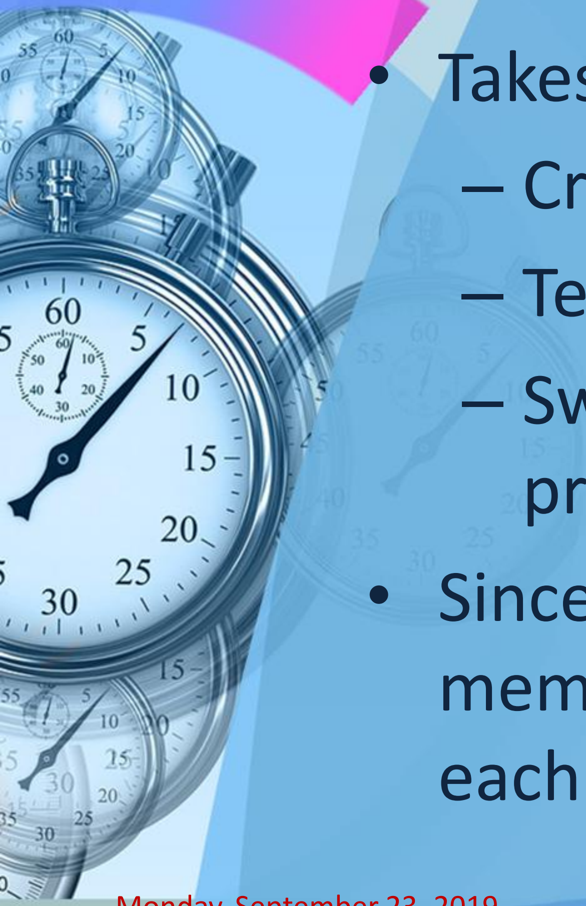
Threads

BIJU K RAVEENDRAN

Benefits

- Responsiveness
  - Even if one thread is blocked other threads can continue execution
- Resource sharing
  - Sharing memory & other resources of the process it belongs to
- Economy
  - It is more economical to create & context switch threads
- Scalability: Utilization of multiprocessor architectures
  - Increases multi threading (threads can run parallel)

Benefits



- Takes less time to
  - Create a new thread than a process
  - Terminate a thread than a process
  - Switch between two threads within the same process
- Since threads within the same process share memory and files, they can communicate with each other without invoking the kernel

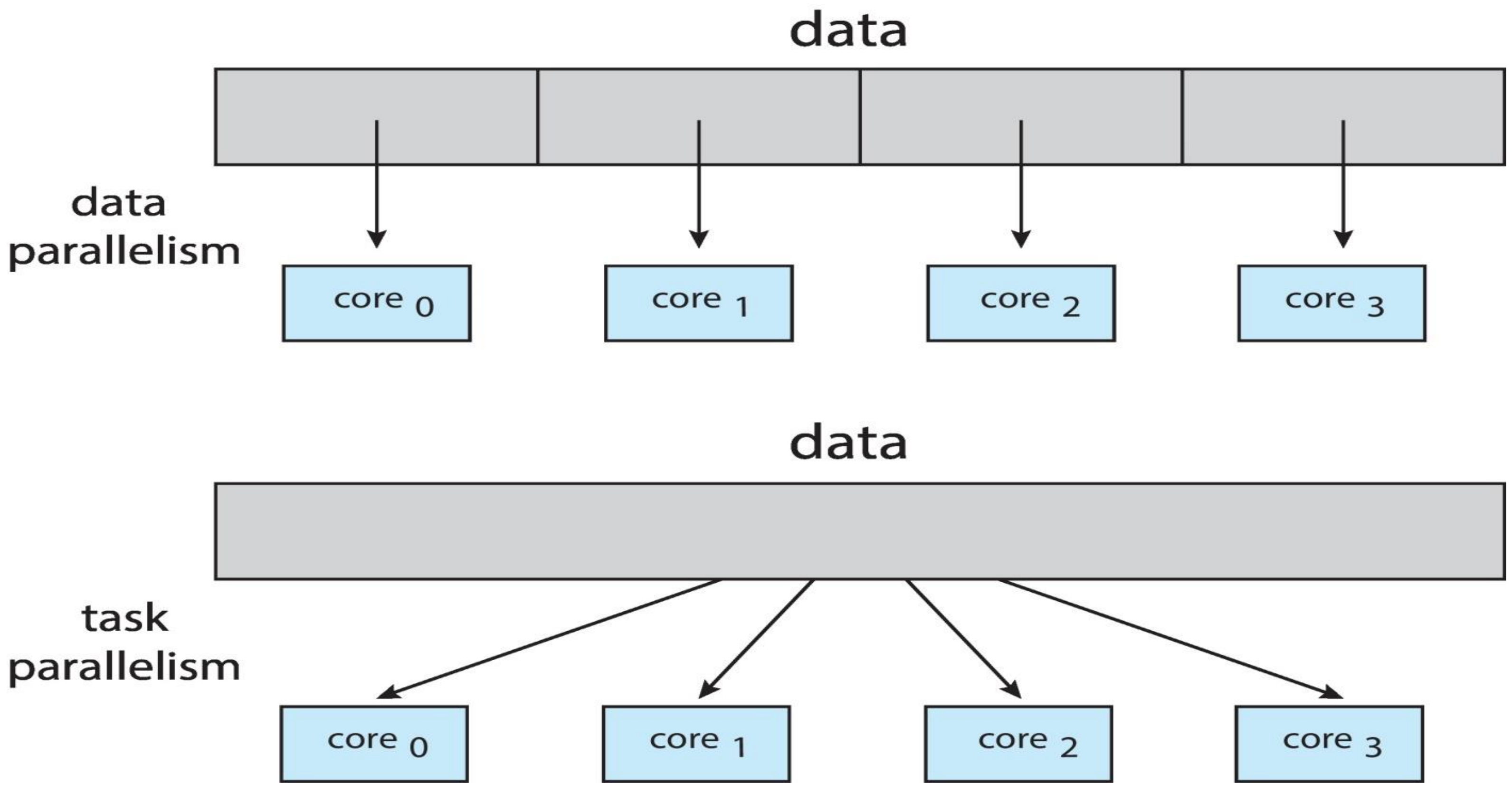
Multi-core programming

- Multicore or multiprocessor systems putting pressure on programmers, challenges include:
  - Dividing activities, Balance, Data splitting, Data dependency, Testing and debugging
- *Parallelism* implies a system can perform more than one task simultaneously
- *Concurrency* supports more than one task making progress
  - Single processor / core, scheduler providing concurrency

Multi-core programming

- Types of parallelism
  - **Data parallelism** – distributes subsets of the same data across multiple cores, same operation on each
  - **Task parallelism** – distributing threads across cores, each thread performing unique operation

Data and Task Parallelism



Thread Libraries

- Thread library provides programmer with API for creating and managing threads
- Two primary ways of implementing
  – Library entirely in user space
  – Kernel-level library supported by the OS

pthread Libraries

- May be provided either as user-level or kernel-level
- A POSIX standard (IEEE 1003.1c) API for thread creation and synchronization
- API specifies behavior of the thread library, implementation is up to development of the library
  - Only Specification NOT the Implementation
- Common in UNIX operating systems (Solaris, Linux, Mac OS X)

Thread Example Program

```
#include<pthread.h>
#include<stdio.h>
int sum;
void *runner(void *param);
int main(int argc,char *argv[])
{   printf("Main Thread:The pid is
            %d\n",getpid());
    printf("Main thread:The tid  is
            %u\n",pthread_self());
    pthread_t tid;
    pthread_attr_t attr;
    pthread_attr_init(&attr);
    pthread_create(&tid,&attr,   runner,
argv[1]);
    pthread_join(tid,NULL);
    printf("Main: sum= %d\n",sum);
}
```

```
void *runner ( void *param )
{   int upper=atoi(param);
    int i;  sum=0;
    printf("New Thread: The pid is
            %d\n",getpid());
    printf("New Thread: The tid is
            %u\n",pthread_self());
    if (upper>0)
    {   for ( i=1; i <= upper; i++ )
        {      sum = sum + i;  }   }
    printf("New Thread: sum = %d
            \n",sum);
    pthread_exit(0);
}
```

Thread Example Program

```
#include<pthread.h>
#include<stdio.h>
#include<asm/unistd.h>
int sum;
void *runner(void *param);
int main(int argc,char *argv[])
{ printf("Main Thread: TID_self =%u,
TID=%d,PID=%d\n",pthread_self(),
syscall(__NR_gettid),getpid());
    pthread_t tid;
    pthread_attr_t attr;
    pthread_attr_init(&attr);
    pthread_create(&tid,&attr,   runner,
argv[1]);
    pthread_join(tid,NULL);
    printf("Main: sum= %d\n",sum);
}
```

```
void *runner ( void *param )
{   int upper=atoi(param);
    int i;  sum=0;
     printf("New Thread: TID_self=
%u,TID=%d, PID=%d\n",
pthread_self(), syscall(__NR_gettid),
getpid());
    if (upper>0)
    {    for ( i=1; i <= upper; i++ )
         {      sum = sum + i;  }   }
    printf("New Thread: sum = %d
             \n",sum);
    pthread_exit(0);
}
```

```
#include<pthread.h>
#include<stdio.h>
int sum;
void *runner ( void *param );
int main ( int argc, char *argv[ ] )
{
    pthread_t tid1, tid2;
    pthread_attr_t attr;
    pthread_attr_init(&attr);
    pthread_create(&tid1, &attr,
            runner, argv[1]);
    pthread_create(&tid2, &attr,
            runner, argv[2]);
    pthread_join(tid1, NULL);
    pthread_join(tid2, NULL);
    printf( "sum = %d \n", sum);
}
```

```
void *runner ( void *param )
{   int upper = atoi (param);
    int i; sum=0;
    if ( upper > 0 )
    {       for ( i=1; i <= upper; i++ )
            {   sum = sum + i;   }
    }
    printf(" The i value is %d and
            the sum value is %d\n",i,
            sum);
    pthread_exit(0);
}
```