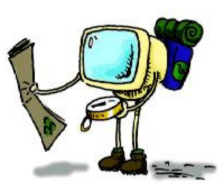
Programming Languages

Tevfik Koşar

Lecture - XXV
April 25th, 2006

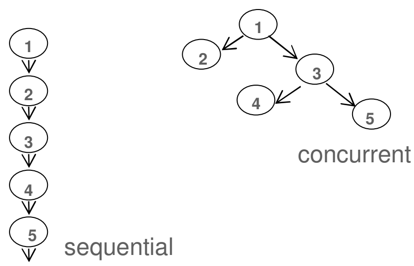
Roadmap

- Concurrent Programming
  - Shared Memory vs Message Passing
  - Divide and Compute
  - Threads vs Processes
  - Synchronization



Concurrent Programming

- So far, we have focused on sequential programming: all computational tasks are executed in sequence, one after the other.
- Next three lectures, we will focus on concurrent programming: multiple computational tasks are executed simultaneously, at the same time.



3

Concurrent Programming

- Implementation of concurrent tasks:
  - as separate programs
  - as a set of processes or threads created by a single program

- Execution of concurrent tasks:
  - on a single processor
  - ➔ Multithreaded programming
  - on several processors in close proximity
  - ➔ Parallel computing
  - on several processors distributed across a network
  - ➔ Distributed computing

4

Communication Between Tasks

Interaction or communication between concurrent tasks can done via:

- Shared memory:
  - all tasks has access to the same physical memory
  - they can communicate by altering the contents of shared memory
- Message passing:
  - no common/shared physical memory
  - tasks communicate by exchanging messages

5

Motivation

- Increase the performance by running more than one tasks at a time.
  - divide the program to n smaller pieces, and run it n times faster using n processors
- To cope with independent physical devices.
  - do not wait for a blocked device, perform other operations at the background

6

Divide and Compute

x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8

How many operations with sequential programming?
7

Step 1: x1 + x2
Step 2: x1 + x2 + x3
Step 3: x1 + x2 + x3 + x4
Step 4: x1 + x2 + x3 + x4 + x5
Step 5: x1 + x2 + x3 + x4 + x5 + x6
Step 6: x1 + x2 + x3 + x4 + x5 + x6 + x7
Step 7: x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8

7

Divide and Compute

1 + 2 + 3 + 4 + 5 + 6 + 7 + 8

Step 1: 1 + 2 = 3
Step 2: 3 + 3 = 6
Step 3: 6 + 4 = 10
Step 4: 10 + 5 = 15
Step 5: 15 + 6 = 21
Step 6: 21 + 7 = 28
Step 7: 28 + 8 = 36

8

Divide and Compute

$$\underbrace{x1 + x2}_{\boxed{}} + \underbrace{x3 + x4}_{\boxed{}} + \underbrace{x5 + x6}_{\boxed{}} + \underbrace{x7 + x8}_{\boxed{}}$$

Step 1: parallelism = 4

Step 2: parallelism = 2

Step 3: parallelism = 1

9

Divide and Compute

$$\underbrace{1 + 2}_{\boxed{3}} + \underbrace{3 + 4}_{\boxed{7}} + \underbrace{5 + 6}_{\boxed{11}} + \underbrace{7 + 8}_{\boxed{15}}$$

Step 1: parallelism = 4

$$\underbrace{\boxed{3}\quad\boxed{7}}_{\boxed{10}} \qquad \underbrace{\boxed{11}\quad\boxed{15}}_{\boxed{26}}$$

Step 2: parallelism = 2

$$\underbrace{\boxed{10}\qquad\boxed{26}}_{\boxed{36}}$$

Step 3: parallelism = 1

10

Gain from parallelism

In theory:

- dividing a program into n smaller parts and running on n processors results in n time speedup

In practice:

- This is not true, due to
  - Communication costs
  - Dependencies between different program parts
    - Eg. the addition example can run only in log(n) time not 1/n

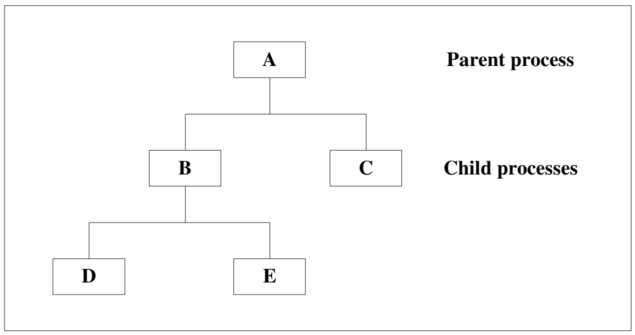
Prevent Blocking

- Do not wait for a blocked device, perform other operations at the background
  - During I/O perform computation
  - During continuous visualization, handle key strokes and I/O
    - Eg. video games
  - While listening to network, perform other operations
    - Listening to multiple sockets at the same time
  - Concurrent I/O, concurrent transfers
    - Eg. Web browsers

Threads vs Processes

Process Spawning:

Process creation involves the following four main actions:
- setting up the process control block,
- allocation of an address space and
- loading the program into the allocated address space and
- passing on the process control block to the scheduler

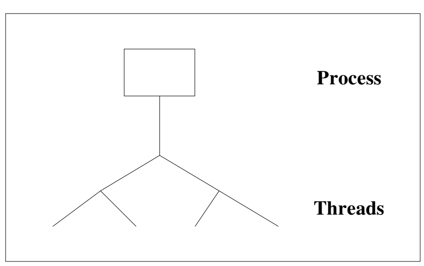


13

Threads vs Processes

Thread Spawning:

- Threads are created *within and belonging to* processes
- All the threads created within one process share the resources of the process including the address space
- Scheduling is performed on a per-thread basis.
- The thread model is a *finer grain scheduling model* than the process model
- Threads have a similar *lifecycle* as the processes and will be managed mainly in the same way as processes are



14

Threads vs Processes

- Heavyweight Process = Process
- Lightweight Process = Thread

Advantages (Thread vs. Process):
- Much quicker to create a thread than a process
- Much quicker to switch between threads than to switch between processes
- Threads share data easily

Disadvantages (Thread vs. Process):
- Processes are more flexible
  - They don't have to run on the same processor
- No security between threads: One thread can stomp on another thread's data
- For threads which are supported by user thread package instead of the kernel:
  - If one thread blocks, all threads in task block.

15

Synchronization

- Mechanism that allows the programmer to control the relative order in which operations occur in different threads or processes.

16

Synchronization - Threads

Int sum = 0;

Thread 1:

```
int t;
lock(sum);
sum = sum + x;
t = sum;
....
unlock(sum);
```

Thread 2:

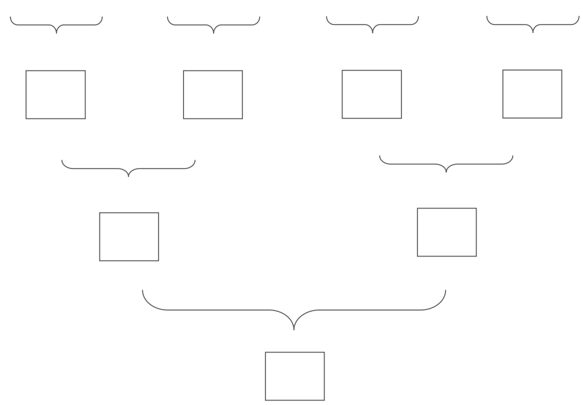
```
int t;
lock(sum);
sum = sum + y;
t = sum;
...
unlock(sum);
```

Use of semaphores for thread synchronization!

17

Synchronization - Processes

x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8



Step 1: parallelism = 4

Step 2: parallelism = 2

Step 3: parallelism = 1

Wait for a message from other processes before continuing processing!

18

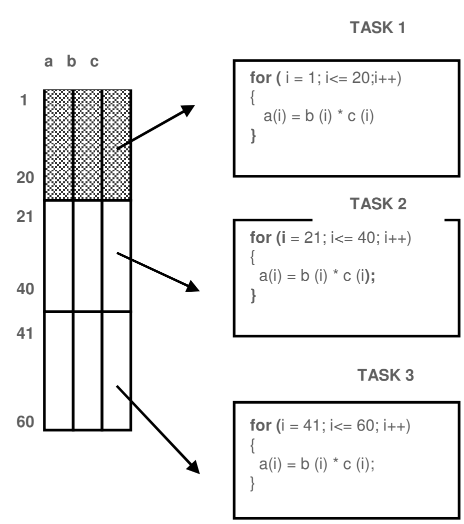
On a single processor machine

- You can have multiple threads
- You can also have multiple processes and have the effect of concurrency
  - timesharing

19

Data Parallelism

```
for (i = 1; i<=60; i++)
{
   a (i)  =  b  (i) * c (i);
}
```

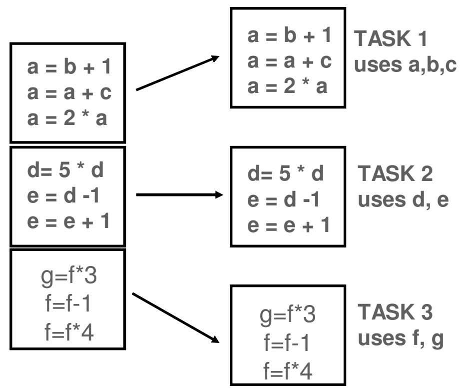


Decompose DATA into pieces.
All the tasks perform the same computation operating with their own piece of data.

20

Function Parallelism

- a=b+1;
- a=a+c;
- a=2*a;
- d=5*d;
- e=d-1;
- e=e+1;
- g=f*3;
- f=f-1;
- f=f*4;



Decompose COMPUTATIONS into pieces (functions). Data are taken to the tasks where they are needed.

21