Threading

Outside Materials

The process model

- Process - program in execution
- CPU switches back & forth among them
- Example of processes:

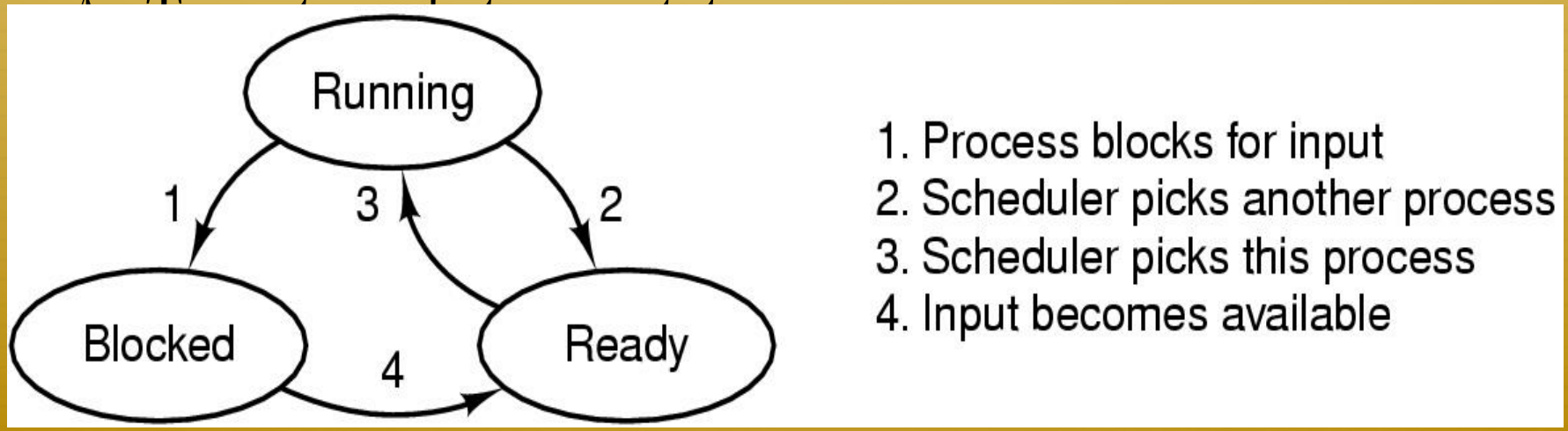
  bring up the *TaskManager*

  and take a look
- If I ran 2 IExplorer instances

  are these two separate processes or one?

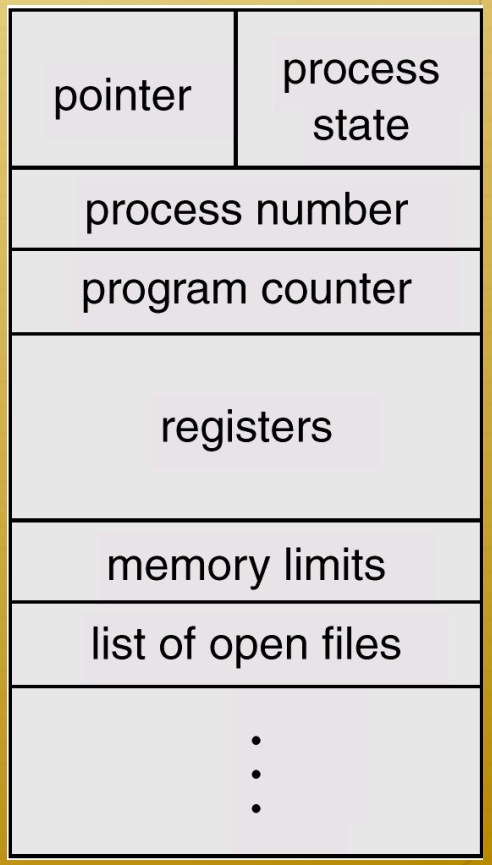
Process states

- Possible process states
  - running
  - blocked
  - ready
- Transitions between states



Implementing processes

- OS maintains a process table
- PCB: information associated with each process
  - Process state
  - Program counter
  - CPU registers
  - CPU scheduling information
  - Memory-management information
  - Accounting information
  - I/O status information
  - …

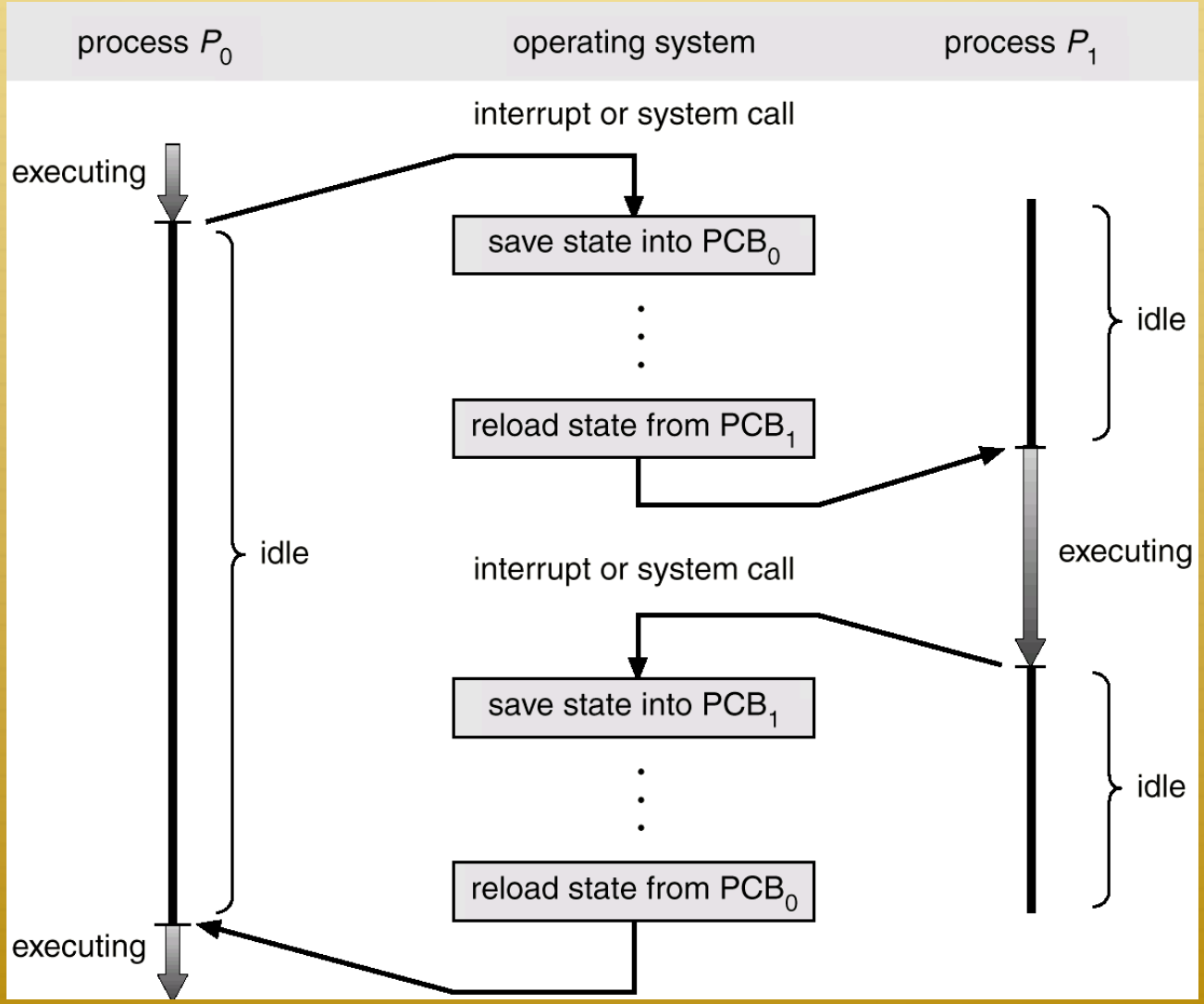


Handling interrupts

What gets done when an interrupt occurs

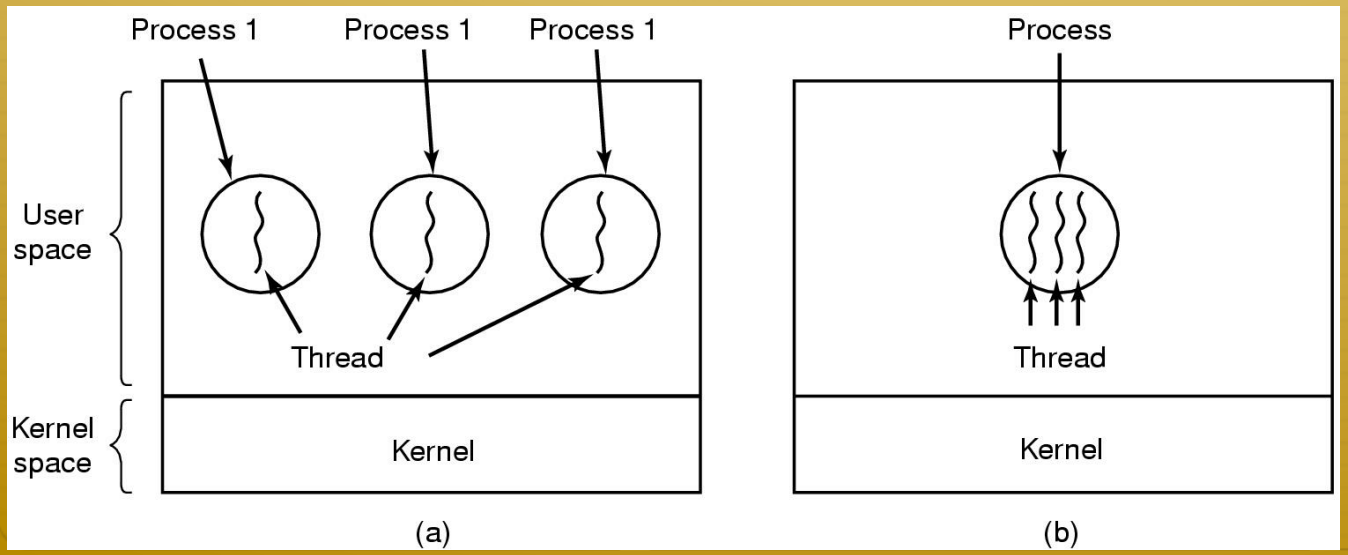
1. Save program counter
2. Save current registers
3. Save process information
4. Put process in waiting queue
5. Scheduler decides which process to run next
6. Load selected process information
7. Put selected process on running queue
8. Run selected process

CPU switch between processes



The thread model

- Traditionally: 1 process – 1 thread of execution
- Thread model:
  - Process to group resources
  - Multiple threads for execution



(a) 3 processes w/ 1 thread each
(b) 1 process w/ 3 threads

The thread model

- Private items

Threads share:
data section
and code

Per process	Per thread
Address space	Program counter
Global variables	Registers
Open files	Stack
Child processes	State
Pending alarms	
Signals and signal handlers	
Accounting information	

- No protection between threads

Creating Threads in C#

- **ThreadStart**
  - Input Parameter: delegate, method to invoke upon starting
- **Start:** starts the thread
- **Join:** Blocks the calling thread until a thread terminates.
- **Sleep:** makes thread sleep for a certain amount to time

Threading in C#

```
public static int Main(string[] args)
{ ....
    Thread backgroundThread = new
    Thread(new ThreadStart(w.DoSomeWork));

    backgroundThread.Start();

    w.completeWork();
}
```

When will w.completeWork get called?

Threading in C#

```
public static int Main(string[] args)
{
    Thread backgroundThread =new Thread(new
    ThreadStart(w.DoSomeWork));

    backgroundThread.Start();

    backgroundThread.Join();

    w.completeWork();
}
```

When will w.completeWork get called?

Class Assignment

Write a .Net program that

- shows two threads counting from 1 to 10 simultaneously with
- different sleep time
- let the user enter the sleep time for each thread

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace tryingThreads
{
    class Program
    {

        public void Dosomework ()
        {
            for (int i =0; i< 10; i++)
            {
                Thread.Sleep(10);
                System.Console.WriteLine("Thread named "+ Thread.CurrentThread.Name+ "changed i to i = " + i
            }
        }
        static void Main(string[] args)
        {
            Thread mythread = Thread.CurrentThread;
            mythread.Name = "backgroundThread";
            Program myprogram = new Program();
            Thread anotherthread = new Thread(myprogram.Dosomework);
            anotherthread.Start();
            myprogram.Dosomework();
            Console.WriteLine("press any key to continue");
            Console.Read();

        }
    }
}
```

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace tryingThreads
{
    class Program
    {
        int Counter { set; get; }
        public void Dosomework ()
        {
            while (Counter < 10)
            {
                Thread.Sleep(10);
                Counter += 1;
                System.Console.WriteLine("Thread named " + Thread.CurrentThread.Name + " counter = " + Counter)
            }
        }
        static void Main(string[] args)
        {
            Thread mythread = Thread.CurrentThread;
            mythread.Name = "backgroundThread";
            Program myprogram = new Program();
            myprogram.Counter = 0;
            Thread anotherthread = new Thread(myprogram.Dosomework);
            anotherthread.Start();
            myprogram.Counter = 0;
            myprogram.Dosomework();
            Console.WriteLine("press any key to continue");
            Console.Read();

        }
```

Try this example

Run it multiple times

What is the problem?

```
using System.Threading ;

namespace ThreadOrdering
{
    class Class1
    {
        int someVariable;

        public void InitializeVariable()
        {
            Thread.Sleep(1);
            someVariable=3;
        }

        public void GetVaraibleMul10()
        {
            Console.WriteLine ("multiplier by 10:"+ 10*someVariable);
            Thread.Sleep(100);
        }
        public void GetVaraibleMul20()
        {
            Console.WriteLine ("multiplier by 20:"+ 20*someVariable);
            Thread.Sleep(100);
        }


        static void Main(string[] args)
        {
            Class1 mycl = new Class1 ();
            Thread t0 = new Thread(new ThreadStart(mycl.InitializeVariable));
            Thread t1 = new Thread(new ThreadStart(mycl.GetVaraibleMul10 ));
            Thread t2 = new Thread(new ThreadStart(mycl.GetVaraibleMul20 ));
            t0.Start();
            t1.Start();
            t2.Start();
        }
    }
```

Try this example

Run it multiple times

What is the problem?

Synchronization Problem

- You have no control on when your thread code gets executed (why?)
- Threads share memory (so why would that matter?)

Thread Safe

- Atomic operations
- Mutual Exclusion
- Thread Local Storage

Solving the Order problem

Using events and signals

```
{
    int someVariable;
    public ManualResetEvent myEvent;
    public Class1 ()
    {
        myEvent = new ManualResetEvent (false); // initial state unset
    }

    public void InitializeVariable()
    {
        myEvent.Reset(); // resets the event
        Thread.Sleep(1);
        someVariable=3;
        myEvent.Set (); // will signal the event
    }

    public void GetVaraibleMul10()
    {
        myEvent.WaitOne (); // blocks until a signal is received
        someVariable *= 10;
        Console.WriteLine ("multiplier by 10:"+ someVariable);
        Thread.Sleep(100);
        Console.WriteLine (" Done Mul 10");
    }
    public void GetVaraibleMul20()
    {
        myEvent.WaitOne (); // blocks until a signal is received
        someVariable *= 20;
        Console.WriteLine ("multiplier by 20:"+ someVariable);
        Thread.Sleep(100);
        Console.WriteLine (" Done Mul 20");
    }

    static void Main(string[] args)
    {
        Class1 mycl = new Class1 ();
        Thread t0 = new Thread(new ThreadStart(mycl.InitializeVariable));
```

Synchronization Problem

- Problem: Race condition

  Atomic vs. nonAtomic Instructions

  i=2; //atomic

  i += 2; // nonAtomic,

  value of i retrieved

  add 2 to that value

  story new value to I

- You may have a bulk of code that you don't want two threads to be doing at the same time

How to avoid this problem?

- Use

  lock (this)

  { ….

  <critical section>

  ..}

Locks

```
//unsafe
class MyUnsafeThread
{
  int x;
  void Increment() { x++; }
  void Assign()    { x = 123; }
}

//safe

class MySafeThread
{
  readonly object mylock = new object();
  int x;

  void Increment() { lock (mylock) x++; }
  void Assign()    { lock (mylock) x = 123; }
}
```

Monitors

```
using System;
using System.Collections;
using System.Threading;

class MainClass
{
  public static ArrayList MyList = new ArrayList();

  static void Main(string[] args)
  {
     Thread ThreadOne = new Thread(new ThreadStart(MonitorExample));
    ThreadOne.Start();
  }
  static void MonitorExample()
  {
    Monitor.Enter(MyList);
    MyList.Add("a value");
    Monitor.Exit(MyList);
  }
}
```