Processes

Getting Number of processors on the Machine

Console.WriteLine(Environment.ProcessorCount);

Get Current Process and Info on the process:

This should give you some information on the current process and its public parameters.

static void GetProcessInfo()

{

Process myProcess = Process.GetCurrentProcess();

//StartProcessWithArgs();

Console.WriteLine($" Physical memory usage : {myProcess.WorkingSet64}");

Console.WriteLine($" Base priority : {myProcess.BasePriority}");

Console.WriteLine($" Priority class : {myProcess.PriorityClass}");

Console.WriteLine($" User processor time : {myProcess.UserProcessorTime}");

Console.WriteLine($" Privileged processor time : {myProcess.PrivilegedProcessorTime}");

Console.WriteLine($" Total processor time : {myProcess.TotalProcessorTime}");

Console.WriteLine($" Paged system memory size : {myProcess.PagedSystemMemorySize64}");

Console.WriteLine($" Paged memory size : {myProcess.PagedMemorySize64}");

}

Create a Child Process:

static void OpenChrome()

{

Process.Start("Chrome");

}

Open Chrome with Arguments:

static void OpenChrome()

{

Process.Start("Chrome", "www.cnn.com");

}

Create a Child Process with Arguments :

static void StartProcessWithArgs()

{

ProcessStartInfo startInfo = new ProcessStartInfo();

startInfo.FileName = "WINWORD.EXE"; // Microsoft word

startInfo.Arguments = "WordDoc.docx"; // Path to the file you want to open

Process.Start(startInfo);

}

Creating child and Parent Process:

## Child Process:

static void Main(string[] args)

{

try

{

string childIndex = args[0];

for (int i = 0; i < 10; i++)

{

Console.WriteLine("I am Child Process " + childIndex);

Thread.Sleep(1000);

}

}

catch{}

}

## Parent Process:

## Start One Child Process:

static void StartOneChildProcesses(int index)

{

Process p = new Process();

p.StartInfo.FileName = @"C:\NEU\OperatingSystems\ProcessInformation\ChildProcess\bin\Debug\ChildProcess.exe";

p.StartInfo.Arguments = “1”;

p.Start();

}

## Starting Multiple Child Processes:

static void StartChildProcesses(int index)

{

Process p = new Process();

p.StartInfo.FileName = @"C:\NEU\OperatingSystems\ProcessInformation\ChildProcess\bin\Debug\ChildProcess.exe";

p.StartInfo.Arguments = index.ToString();

p.Start();

}

static void Main(string[] args)

{

Parallel.For(0, 5, index =>

{

Console.WriteLine("Starting process: " + index);

StartChildProcesses(index);

});

}

Kill a child process:

static void KillChildProcess()

{

Process current = Process.GetCurrentProcess();

Process p = new Process();

p.StartInfo.FileName = @"C:\NEU\OperatingSystems\ProcessInformation\ChildProcess\bin\Debug\ChildProcess.exe";

p.StartInfo.Arguments = "1";

p.Start();

Thread.Sleep(2000);

p.Kill();

}

## Kill All Child Processes:

Since Child Processes can spawn their own child processes we need to recursively call all the child processes and then kill then in order.

private static void KillProcessAndChildren(int pid)

{

// Cannot close 'system idle process'.

if (pid == 0)

{

return;

}

ManagementObjectSearcher searcher = new ManagementObjectSearcher

("Select \* From Win32\_Process Where ParentProcessID=" + pid);

ManagementObjectCollection moc = searcher.Get();

foreach (ManagementObject mo in moc)

{

KillProcessAndChildren(Convert.ToInt32(mo["ProcessID"]));

}

try

{

Process proc = Process.GetProcessById(pid);

proc.Kill();

}

catch (ArgumentException)

{

// Process already exited.

}

}

## Printing Getting Memory usage of Chrome:

Since chrome process spawn multiple other processes

static void PrintAllProcessesInfo()

{

Process[] processList = Process.GetProcesses();

long totalBytes = 0;

// Loop through the array to show information of every process in your console

foreach (Process process in processList)

{

if (process.ProcessName.Contains("chrome"))

{

Console.WriteLine(@" {0} | ID: {1} | Status {2} | Memory (private working set in Bytes) {3}",

process.ProcessName, process.Id, process.Responding, process.PrivateMemorySize64);

totalBytes += process.PrivateMemorySize64;

}

}

Console.WriteLine("Total Mem used in mb = " + totalBytes / (1024 \* 1024));

}

Kill a child process:

## Shared Memory (Memory Mapped File)

Two processes can share a file which is memory mapped. Both can read/write a file and communicate with each other. Memory mapped file is a file which has been given a part of virtual memory and it is directly mapped to a file in Hard Disk.

Benefits of using Memory Mapped file is that there is I/O speedup. Specially on large files. You can use small RAM for even large files.

## Process A:

static void Main(string[] args)

{

using (MemoryMappedFile memoryMappedFile =

MemoryMappedFile.CreateFromFile(@"c:\temp\log.txt", FileMode.OpenOrCreate, "log-map", 10000))

{

using (MemoryMappedViewAccessor viewAccessor = memoryMappedFile.CreateViewAccessor())

{

byte[] textBytes = Encoding.UTF8.GetBytes("Writing text from Process A");

viewAccessor.WriteArray(0, textBytes, 0, textBytes.Length);

}

}

}

## Process B:

static void Main(string[] args)

{

using (MemoryMappedFile memoryMappedFile =

MemoryMappedFile.CreateFromFile(@"c:\temp\log.txt", FileMode.OpenOrCreate, "log-map"))

{

using (MemoryMappedViewAccessor viewAccessor = memoryMappedFile.CreateViewAccessor())

{

byte[] bytes = new byte[10];

viewAccessor.ReadArray(0, bytes, 0, bytes.Length);

string text = Encoding.UTF8.GetString(bytes).Trim('\0');

Console.WriteLine("Read following from Memory Mapped file: " + text);

}

}

}

## Messaging Queue:

Processes can communicate with each other using Messaging Queue, in windows you can enable messaging (MSMQ) Service. You would have to start the MSMQ server from “Turn Windows Features On or Off”. Once you have enabled the Messaging queue, you can go “Computer Management” on your windows machine and MSMQ are available in Services and Applications. You can create Private queues and send data to queues.

## Sending Message to Message Queue:

private static void SendMessage(string queueName, string msg)

{

MessageQueue messageQueue = null;

if (!MessageQueue.Exists(queueName))

messageQueue = MessageQueue.Create(queueName);

else

messageQueue = new MessageQueue(queueName);

try

{

Message m = new Message(msg, new XmlMessageFormatter(new String[] { "System.String,mscorlib" }));

messageQueue.Formatter = new XmlMessageFormatter(new String[] { "System.String,mscorlib" });

messageQueue.Send(msg);

}

catch { }

finally

{

messageQueue.Close();

}

}

## Receiving Message from Message Queue:

private static void ReceiveMessage(string queueName)

{

if (!MessageQueue.Exists(queueName))

return;

MessageQueue messageQueue = new MessageQueue(queueName);

messageQueue.Formatter = new XmlMessageFormatter(new String[] { "System.String,mscorlib" });

try

{

Message[] messages = messageQueue.GetAllMessages();

foreach (Message m in messages)

{

Console.WriteLine(m.Body);

}

}

catch { }

finally

{

messageQueue.Close();

}

}

You would send and receive messages like this:

SendMessage(@".\Private$\MsgQueueAshish", "This is a message");

ReceiveMessage(@".\Private$\MsgQueueAshish");