Advanced Operating Systems

{ Part 1: Process

Introduction

- **System Calls** - how a program requests a service from an operating system's kernel
  - **may include**:
    - hardware-related services (e.g. accessing a hard disk drive)
    - creation and execution of new processes
    - communication with integral kernel services such as process scheduling

- System calls provide an essential interface between a process and the operating system

Introduction

- Operating System
  - Manages computer hardware and provides environment for application programs to run.
- Mode
  - User mode
  - Kernel mode
- System Calls
  - Provide an interface to the services made available by the OS

Introduction

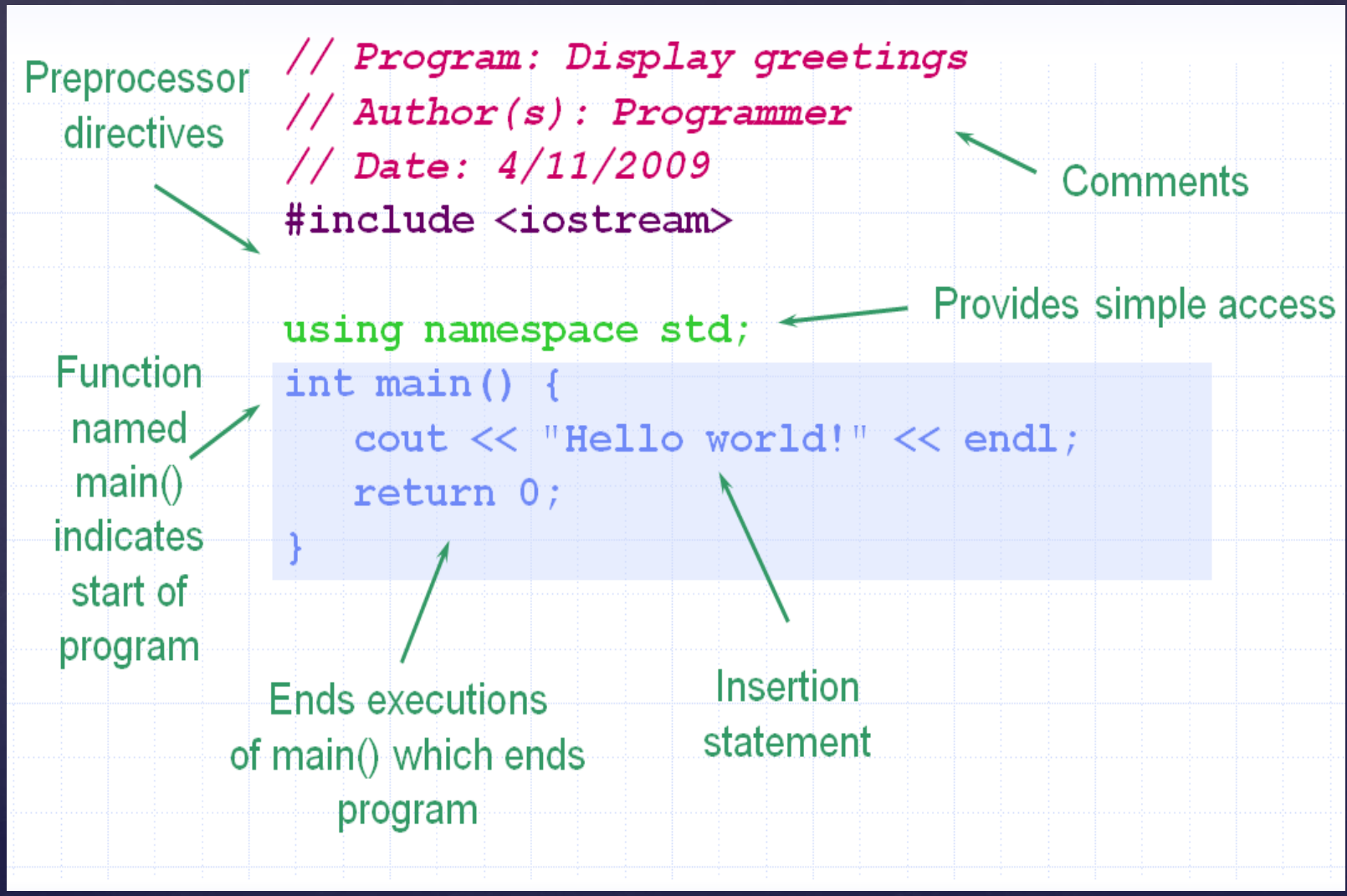
- System calls provide an essential interface between a process and the operating system
- For example, UNIX has about 60 system calls that are at the *heart* of the operating system

Creating, Compiling and Running a C/C++ program

- Creating a C/C++ Program
  - Editor on a PC machine such as Notepad, Text Pad
  - Editor on a UNIX machine such as vi editor, Pico
- Compiling a C/C++ Program
  - GNU Compiler (g++), g++ myFile.C
  - also can use ***-o*** and filename in the compilation:

    g++ -o myFile myFile.C
- Running the program
  - a.out or myFile

C/C++ program


Preprocessor directives
// Program: Display greetings
// Author(s): Programmer
// Date: 4/11/2009
#include <iostream>
Comments
using namespace std;
Provides simple access
Function named main() indicates start of program
int main() {
cout << "Hello world!" << endl;
return 0;
}
Ends executions of main() which ends program
Insertion statement

Where to start

- Available UNIX/Linux: cs1.calstatela.edu in CSULA campus
- Use telnet to login. There are several different kind of telnet.

Vi Editor

- modal editor (two mode)
- command mode
  - The initial mode where most of the keys perform vi commands
- insert mode
  - The mode where what you type is inserted into your document
- command line mode transitions
  - command mode to insert mode
    Several commands/keys put you into insert mode
    i a I A o O
  - insert mode to command mode
    Hit the escape key

Vi editor commands

- To create a file
  - Vi filename.c
- To save a file
  - :wq write any changes and quit
  - :q Quit(will only do so if no changes)
  - :q! quit without saving changes
- To delete
  - x -- delete a character
  - dd – delete the entire line
  - d$ -- delete from the current position to the end of
  - the line

Time Functions

- **We can access the clock time with Operating System calls.**
- **Uses of time functions include:**
  - Telling the time
  - Timing programs and functions
  - Setting number seeds

Time Functions

- **DateTime is a predefined class in C++ from the System library**
  - accesses system date & time with **DateTime.Today**

Time in C++ on UNIX

- time_t is typedefed to a long (int) in <sys/types.h> and <sys/time.h>
- time_t time(time_t *tloc) -- returns the time since 00:00:00 GMT, Jan. 1, 1970, measured in seconds.
- If tloc is not NULL, the return value is also stored in the location to which tloc points. On failure, it returns (time_t) -1.
- char *ctime(time_t *clock) -- ctime() converts a long integer, pointed to by clock, to a 26-character string

Time in C++ on UNIX

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <time.h>

using namespace std;

main()
{
  int i;
  time_t time_local;
  for (int i = 0; i < 10; i++)
  {
    time(&time_local);
    cout << ctime(&time_local) << endl;
    sleep(1);
  }
}
```

Time in Java

```
import java.util.Date;


public class My_Time
{
        public static void main(String[] args)
        {
                long currentTime;
                Date date;
                for (int i = 0; i < 10; i++)
                {
                        currentTime = System.currentTimeMillis();
                        date = new Date(currentTime);
                        System.out.println(date.toString());
                        System.out.println(currentTime);
                   try
                   {
                        Thread.sleep(1000);
                   } catch (Exception e) {}
                 }
        }
}
```

Time in C#

- **DateTime is a predefined class in C# from the System library**
  - accesses system date & time with **DateTime.Today**

- **Visual Studio 2010,**
  - **Project -> Console Applicayion**

```
using System;

namespace MyTime
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Today is : {0}", DateTime.Today);
            DateTime yesterday = DateTime.Today.AddDays(-1);
            Console.WriteLine("Today is : {0}", yesterday);
        }
    }
}
```

Process Control

- A process is a single running program.
  - Process ID: pid
  - Process Status: UNIX command ps
  - System call: getpid()

- A program usually runs as a single process.
  - Programs run as several separate communicating processes.

Process(C/C++)

- Every process has a parent process(except init)
- Think of processes on a Linux system as if they form a tree, with init at it's root
  - The parent process ID, "ppid", is simply the process ID of the process's parent
  - When referring to process ID's in C/C++ code, always use pid_t typedef

Process(C/C++)

- The command “ps”, gives us a list of processes and some info on them such as their process ids

  Take a try on Linux machine

Process(C/C++)

- We can easily and separately access that information by getpid() and getppid()
- We definitely need <stdlib.h> and <unistd.h>

```
#include <iostream>
#include <string>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
using namespace std;
```

Process(C/C++)

There are two ways to create a new process:

(1) Using the system function can run a program easy. System call system(char *string) – where string can be the name of a UNIX utility, an executable shell script or a user program. System returns the exit status of the shell. system is prototyped in <stdlib.h>.

```
#include <stdlib.h>
using namespace std;
int main ()
{
    int return_value;
    return_value = system ("ls -l /");
    return return_value;
}
```

Process(C/C++)

(2) Using fork() system call to create a subprocess, hands the command to the Bourne shell for execution

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

```
#include <sys/types.h>
#include <sys/wait.h>
#include <iostream>
#include <unistd.h>
#include <cstdlib>

using namespace std;
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Process(C/C++)

```
int main()
{
    int pid;
    pid = fork();
    if (pid == 0) // This is child process
    {
      cout << "This is Child Process. pid = " << getpid() << endl;
    }
    else // This is parent process
    {
      cout << "This is Parent Process. pid = " << getpid() << endl;
      wait(NULL);
      cout << "This is Parent Process. Finished " << endl;
    }
}
```

Process(C/C++)

- Although using system is relatively simple but it's inefficient and has security risk. Using fork and exec methods are preferred for creating new processes
- Linux's fork function makes a child process that is an exact copy of it's parent process
- If you want the process to be the instance of another program, use exec
- To create a new process, use fork and then exec to make a copy of the current process and then change the process to be an instance of a program

Process(C/C++)

- When fork is called, a duplicate of the process is created
  - Referred to as the “child” of the “parent” process
  - Executes the same program from the same place
  - Has it’s own unique process ID
- Fork function provides different return values to the parent and the child
  - one process(parent) goes in to fork and two(parent and child) come out
  - Child ID is returned zero while parent ID is returned with the child ID

Process(C/C++)

- exec functions replace the program running with another program
  - When exec is called by a program, the process stops executing that program and starts a new one
  - When using an exec function in programs, the name of the function should be passed as the first element of the argument list
  - Since exec replaces the program with another one, it never returns unless an error occurs

Process(C/C++)

- execl stands for execute and leave
- execl(char *path, char *arg0,...,char *argn, 0);
  - The last parameter must always be 0.
  - It is a NULL terminator.
  - Where path points to the name of a file holding a command that is to be executed,
  - arg0 points to a string that is the same as path (or at least its last component).
  - arg1 ... argn are pointers to arguments for the command and
  - 0 simply marks the end of the (variable) list of arguments.

Process(C/C++)

- execl example

```
main()
{
        cout << ``Files in Directory are: n'';
        execl(`/bin/ls'',``ls'', ``-l'',0);
}
```

Process(C/C++)

- Using execvp and execlp, a program name is used to search for a program in the current path
- Using execv and execve, an argument list for the new program as a NULL-terminated array of pointers to strings is used

Process(C/C++)

- When a program is invoked from the shell, the shell sets the first element of the argument list to the name of the program, the second element of the argument list to the first command-line argument and so on.

Process(C/C++)

```
int global=2;
pid_t process_ID = 0;
main(){
    string process_type;
    int iStack=50;
    pid_t pID = fork();
    if (pID==0) {
        process_type = "Child Process: ";
        global++;
        iStack++;
        process_ID = (int) getpid();
        execlp("ls", "ls", "-l", "/", NULL);
        cout<<"Main program is over"<<endl;
    }
```

Process(C/C++)

```
else if (pID > 0) {
    process_type = "Parent Process: ";
    process_ID = (int) getppid();
}
    cout<<process_type;
    cout<<" Global variable: "<<global;
    cout<<" Stack Variable: "<<iStack<<endl;
    cout<<" Process ID is "<<process_ID<<endl;

}
```

Process(C/C++) - wait(int *)

- int wait (int *status_location)
  - Force a parent process to wait for a child process to stop or terminate.
  - wait() return the pid of the child or -1 for an error.
  - The exit status of the child is returned to status_location.

Process(C/C++) - exit(int)

- void exit(int status)
- terminates the process which calls this function and returns the exit status value.
- Both UNIX and C++ (forked) programs can read the status value (from wait(), echo $? For bash).
- By convention, a status of 0 means normal termination any other value indicates an error or unusual occurrence.

```
#include <iostream>
#include <cstdlib>
#include <unistd.h>
#include <sys/wait.h>

using namespace std;

int main()
{
   int pid;
   int child_exit_value;
   pid = fork();
   if (pid == 0) // child process
   {
     cout << "Child process pid = " << getpid() << endl;
     sleep(1);
     exit(8);
   }
   else // parent process
   {
     child_exit_value = 0;
     wait(&child_exit_value);
     cout << "Parent process pid = " << getpid() << endl;
     cout << "The exit system call in Child process return value : " << endl;
     cout << WEXITSTATUS(child_exit_value) << endl;
   }
}
```

Process(C/C++) – Multi Children

- int wait (int *status_location)
  - If the calling process has multiple child processes, the function returns when one returns.
  - To wait for one child: wait(NULL).
  - To wait for many children: waitpid(pid[i], NULL, 0);

```
#include <iostream>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <cstdlib>

using namespace std;

int main()
{
   int pid[3];
   pid[0] = fork();
   if (pid[0] > 0)
   {
      cout << "[parent] 1 pid = " << getpid() << endl;
      pid[1] = fork();
      if (pid[1] > 0)
      {
        cout << "[parent] 2 pid = " << getpid() << endl;
        pid[2] = fork();
        if (pid[2] > 0)
        {
          cout << "[parent] 3 pid = " << getpid() << endl;
        }
        else if (pid[2] == 0)
        {
          cout << "[child 3] pid = " << getpid() <<  endl;
          exit(3);
        }
      }
      else if (pid[1] == 0)
      {
        cout << "[child 2] pid = " << getpid() <<  endl;
        exit(2);
      }
      for(int i = 0; i < 3; i++)
      {
        waitpid(pid[i],NULL,0);
      }
      cout << "[parent] 1 finished " << endl;
   }
   else if (pid[0] == 0)
   {
      cout << "[child 1] pid = " << getpid() << endl;
      exit(1);
   }
}
```

Process(Java)

- Before JDK 5.0, the only way to fork off and execute a process was to use the exec() method of the java.lang.Runtime class.
  - The java.lang.Runtime.exec(String command) method executes the "command" in a separate process.

```
public class CreateProcess
{
    public static void main(String[] args)
    {
      try
      {
    System.out.println("Create a process to execute notepad.exe");
    Process process = Runtime.getRuntime().exec("notepad.exe");
    System.out.println("Notepad is now open in other window.");
      } catch (Exception ex) {}
    }
}
```

- New ways of executing a command in a separate process involves a class called ProcessBuilder.

Process(Java)

- Before calling the exec()
  - Specify the command and it's arguments
  - Environment variable settings
  - Working directory

- All versions of the method return a java.lang.Process object for managing the created process
  - After this, we can get the input/output stream of the subprocess and other information

Java Process Exec()

```
import java.util.*;
import java.io.*;

public class J_Process
{
  public static void main(String[] args) {
    try {
        Process myProcess = Runtime.getRuntime().exec("/bin/ls");
        InputStreamReader myProReader = new
InputStreamReader(myProcess.getInputStream());
        BufferedReader myProInput = new BufferedReader(myProReader);
        String line;
        while ((line = (myProInput.readLine())) != null)
        {
          System.out.println(line);
        }
    } catch (Exception e)
    { }
  }
}
```

Process(Java)

- The ProcessBuilder class has two constructors
  - public ProcessBuilder(List command)
    - Accepts a List for the command and it's argument
  - public ProcessBuilder(String… command)
    - Accepts a variable number of String arguments

- Call start() to execute the command

Process(Java) - Windwos

```
import java.io.*;
import java.util.*;

public class Build_Process_Windows {
    public static void main(String[] args) throws Exception
    {
    String[] proCommands = {"cmd", "/c", "dir"};
    ProcessBuilder processBuilder = new ProcessBuilder(proCommands);
    processBuilder.directory(new File("c:\\temp"));
    Process myProcess = processBuilder.start();

     BufferedReader readOutputCMD = new BufferedReader(new InputStreamReader(
                                myProcess.getInputStream()));
     String outputLine;
     System.out.println("Output of " + Arrays.toString(proCommands));
     while ((outputLine = readOutputCMD.readLine()) != null)
     {
        System.out.println(outputLine);
     }

     int processExitValue = myProcess.waitFor();
     System.out.println("Process Exit Value is: " + processExitValue);
    }
}
```

Process (Java) - LINUX

```
import java.io.*;
import java.util.*;

public class Build_Process
{
    public static String getProcessID() throws Exception
    {
        List<String> proCommands = new ArrayList<String>();
        proCommands.add("/bin/bash");
        proCommands.add("-c");
        proCommands.add("echo $PPID");
        ProcessBuilder processBuilder = new ProcessBuilder(proCommands);

        Process myProcess = processBuilder.start();
        myProcess.waitFor();
        if(myProcess.exitValue() == 0)
        {
            BufferedReader outReader = new BufferedReader(new InputStreamReader(myProcess.getInputStream()));
            return outReader.readLine().trim();
        }
        else
        {
            System.out.println("Error while getting PID");
            return "";
        }
    }
  public static void main(String[] args) throws Exception
  {
     System.out.println("Process ID is" + getProcessID());
  }
}
```

Process(Java) – a little complex one

- combine with threads

Process (C#)

In order to access system calls from C#, we must include the following **namespaces** (collections of classes):

```
using System;
using System.Diagnostics;
```

"using" is a keyword that imports a namespace

Process (C#)

- Creating a new process in C# (use Process class)

  Process myProcess = new Process();
- Retrieving Process ID

  Console.WriteLine(myProcess.Id);

  //prints process Id
- Start process

  myProcess.Start()

  //starts specified process (arguements optional)

Process (C#)

Executing a process

```
using System.Diagnostics;

namespace MyProcess_1
{
    class Program
    {
        static void Main(string[] args)
        {
            Process.Start("notepad.exe");
        }
    }
}
```

Process (C#)

- Accessing process name

  myProcess.ProcessName()
  //returns String of process name

- Accessing process status

  myProcess.Responding
  //returns boolean true if running, false if
  //idle

Process (C#)

Accessing Process Information

```
using System;
using System.Diagnostics;

namespace MyProcess_2
{
   class Program
   {
      static void Main(string[] args)
      {
         Process[] myProcesses = Process.GetProcesses();
         Console.WriteLine("Number of Processes: {0}", myProcesses.Length);
         foreach (Process tmpProcess in myProcesses)
         {
            Console.WriteLine(tmpProcess.Id);
         }
      }
   }
}
```

Process (C#)

Accessing Process Information

```
using System;
using System.Diagnostics;

namespace MyProcess_3
{
    class Program
    {
        static void Main(string[] args)
        {
            Process[] myProcesses = Process.GetProcesses();
            Console.WriteLine("Number of Processes: {0}", myProcesses.Length);
            Console.WriteLine("{0, -8} {1, -30} {2, -10}", "PID", "Process Name", "Status");
            foreach (Process tmpProcess in myProcesses)
            {
                Console.WriteLine("{0, -8} {1, -30} {2, -10}",
                    tmpProcess.Id, tmpProcess.ProcessName, tmpProcess.Responding ? "Running" : "Idle");
            }
        }
    }
}
```

Process (C#)

- Waiting for a process to end

```
Process waitProcess = myProcess.Start();
waitProcess.WaitForExit();
//exits after myProcess is complete
```

- Process Exit Code

```
myProcess.ExitCode
//returns 0 if normal termination occurs
// any other value indicates an error or unusual
//occurrence
```

Process

- Forcing a process to end
  myProcess.Kill();
  //forcess the process to terminate immediately

```
using System;
using System.Diagnostics;
using System.Threading;

namespace MyProcess_4
{
    class Program
    {
        static void Main(string[] args)
        {
            Process myProcess = Process.Start("notepad.exe");
            Thread.Sleep(3000);
            myProcess.Kill();
        }
    }
}
```