# Shell Programs

COMP755 Advanced Operating Systems

## Shell Interface Program

- The shell is the user interface.
- It is part of the OS
- Runs at the user level *(not kernel level)*
- The shell reads the user's commands and then causes the applications to be loaded and executed.

## Shell Features

- Reads user input and executes programs.
- Provides the prompt.
- Implements redirection
- Implements parallel execution with &
- Shells may also provide a graphical user interface.
- May implement simple commands

## Human Interface

- The shell is the part of the OS that is most important to most users.
- Considerable effort has gone into trying to make the interface intuitive and easy to use

## Popular Shells

- command.com in DOS
- C shell in Unix
- Korn shell in Unix
- Borne shell in Unix
- explorer.exe – Windows user interface
- Cygwin – Unix shell for Windows

## Shell Outline

```
do forever {
     print prompt;
     read command;
     if (fork()==0) { // child process
          use exec() to load program;
     }
     wait for child process to terminate;
}
```

## **execv** function

```
int _execv( const char *cmdname,
        const char *argv[] );
```

where:

- *cmdname* is the name of the program file.
- *argv* is an array of pointer to strings containing the parameters. By convention argv[0] = cmdname; The last argv entry must be NULL.

The execv function causes the specified program to overlay the calling program. This function does not return if successful.

## Simple C Program

```
int main ( int argc, char *argv[] )
{
    ...
    return 5;
}
```

The **argv** array passed to the main function when you start a C programs is the **argv** array passed to **execv** by the shell.

## Microsoft Processes

- You can start a process under Microsoft Windows using:

```
Process::Start(S"myprog.exe");
```

This is a Microsoft Foundation Class object.

## Standard I/O Streams

When a program is started, it has three I/O streams open:

0 stdin – Standard input, usually keyboard

1 stdout – Standard output, usually screen

2 stderr – Standard error, usually screen

## Redirection

Standard I/O streams can be redirected from the command line

```
ls > myfile.txt
myprog < usualstuff.txt
cc nogood.c >& allmsg.txt
```

## Redirection Implementation

- The freopen("filename", mode, *stream) function will direct output to stream to the specified filename.
- The shell can redirect stdin or stdout to the filename specified on the command line.
- The shell forks a new process, redirects stdin and/or stdout then does the exec()

## Multiple Processes

- If you put a “&” at the end of a command, the shell will not wait for the process to terminate before printing the next prompt.
- You can run a process “in the background” by putting a “&” at the end of the line.
- You can put a “&” between commands to execute them in parallel.

## Example Multiple Processes

```
makefile bigthing  &

xterm &

ls & cc myprog.c

ls & cc myprog.c & ps
```

## Shell Outline

```
do forever {
     print prompt;
     read command;
     if (fork()==0) { // child process
          use exec() to load program;
     }
     if (no “&”)
          wait for child to terminate;
}
```

## Piping

The output stream of one program can become the input stream of another program

```
arp –a | grep ncat
head myfile | lpr
```

## Commands Implemented in the Shell

- Some shells implement simple commands such as:
  - **cd**
  - **pwd**
  - **history**

## Scripts

- The shell can execute script files to produce commands for the shell.
- Scripts are a program whose output is a series of commands to the shell.
- DOS batch files are a simple example of scripting.
- Unix scripts can be much more elaborate.