* **Lecture 11 - Named Pipes; Process Substitution; Advanced Permissions**

**Named Pipes**

* + pipes are one method of Inter-Process Communication in Unix
  + the familiar "|" is called an anonymous pipe, and connects the output of one process to the input of another
  + a named pipe also connects two processes, and consists of two pipes, one for input, one for output
  + a named pipe is also known as a FIFO: "First In First Out"
  + named pipes can be accessed by any local process which has required access permissions
  + mkfifo command is used to create a named pipe
  + often put in the /tmp directory
  + delete with rm
  + to use a named pipe, one process opens it for reading, the other process opens for writing
  + for the following examples, each Matrix server has it's own /tmp directory, so make sure both userids are on the same server
  + you can check this with the hostname command
  + the first process creates the named pipe and gives read/write access to everyone, then it reads from the pipe and waits till another process writes to it:

==> mkfifo /tmp/unx510.fifo

==> chmod 666 /tmp/unx510.fifo

==> cat /tmp/unx510.fifo\_

* + the second process displays information about the pipe, and writes to it:

==> ls -l /tmp/unx510.fifo

﻿prw-rw-rw- 1 lczegel users 0 Aug 6 21:45 /tmp/unx510.fifo

﻿==> cat cars > /tmp/unx510.fifo

﻿==> \_

* + the first process successfully reads from the pipe, then writes to it and waits for another process to read it:

==> mkfifo /tmp/unx510.fifo

﻿==> chmod 666 /tmp/unx510.fifo

﻿==> cat /tmp/unx510.fifo

﻿plym fury 77 73 2500

chevy nova 79 60 3000

ford mustang 65 45 17000

volvo gl 78 102 9850

ford ltd 83 15 10500

Chevy nova 80 50 3500

fiat 600 65 115 450

honda accord 81 30 6000

ford thundbd 84 10 17000

toyota tercel 82 180 750

chevy impala 65 85 1550

ford bronco 83 25 9525

﻿==> echo 'Wow, it works!' > /tmp/unx510.fifo\_

* + the second process successfully reads from the pipe:

==> ls -l /tmp/unx510.fifoprw-rw-rw- 1 lczegel users 0 Aug 6 21:45 /tmp/unx510.fifo

﻿==> cat cars > /tmp/unx510.fifo

﻿==> cat /tmp/unx510.fifoWow, it works!

==> \_

* + the first process deletes the pipe:

==> mkfifo /tmp/unx510.fifo

﻿==> chmod 666 /tmp/unx510.fifo

﻿==> cat /tmp/unx510.fifo

﻿plym fury 77 73 2500

chevy nova 79 60 3000

ford mustang 65 45 17000

volvo gl 78 102 9850

ford ltd 83 15 10500

Chevy nova 80 50 3500

fiat 600 65 115 450

honda accord 81 30 6000

ford thundbd 84 10 17000

toyota tercel 82 180 750

chevy impala 65 85 1550

ford bronco 83 25 9525

﻿==> echo 'Wow, it works!' > /tmp/unx510.fifo

﻿==> rm /tmp/unx510.fifo==> \_

* + could also have used: mkfifo -m 666 /tmp/unx510.fifo

**Client-Server Applications**

* + an example of a simple server using a named pipe:

fifo=/tmp/log\_server\_pipeif [[ ! -p $fifo ]] && ! mkfifo -m 666 $fifothen echo "Error: could not create pipe" >&2exit 1fiwhile read log\_recdoecho $log\_recdone < $fifo

* + an example of a matching client:

fifo=/tmp/log\_server\_pipeif [[ ! -p $fifo ]]then echo "Sorry, log server not running" >&2exit 1fiprintf "Please enter a comment: "read log\_recordecho "$log\_record" > $fifo

* + note that the server often terminates when the read from the pipe mistakenly sees a client termination as an EOF
  + the server can keep the pipe output open, so that the pipe doesn't close with a read error:

fifo=/tmp/log\_server\_pipeif [[ ! -p $fifo ]] && ! mkfifo -m 666 $fifothen echo "Error: could not create pipe" >&2exit 1fiwhile read log\_recdoecho $log\_recdone < $fifo 3>$fifo

* + alternatively, the server loop can be made independent of read failure:

fifo=/tmp/log\_server\_pipeif [[ ! -p $fifo ]] && ! mkfifo -m 666 $fifothen echo "Error: could not create pipe" >&2exit 1fiwhile :doif read log\_rec < $fifothenecho "$log\_rec"fidone

* + note that with any of these examples, the pipe input can be redirected to either the "read" or the "while"
  + unlike a normal file, once something is read from a pipe, it is gone and will not be read again if the pipe is closed and reopened
  + this client can accept the comment as a single argument, or as multiple arguments:

fifo=/tmp/log\_server\_pipeif [[ ! -p $fifo ]]then echo "Sorry, log server not running" >&2exit 1fiif [[ -z $1 ]]thenread -p "Please enter a comment: " log\_recordelselog\_record="$\*"fiecho "$log\_record" > $fifo

* + note that the output of the log server could be redirected to a file, with write permission needed by the server, but not by the client

**Process Substitution**

* + a named pipe is internally generated, to connect to another process as either input or output
  + <( command-list ) - output of the command list can be used as an input file
  + can be similar to anonymous piping:

==> head -1 <(grep ford cars | tr "a-z" "A-Z")

FORD MUSTANG 65 45 17000

﻿==> grep ford cars | tr "a-z" "A-Z" | head -1

﻿FORD MUSTANG 65 45 17000

﻿==> \_

* + if a file needs to be redirected, so does a process substitution:

==> tr "a-z" "A-Z" < <(grep ford cars | head -1)

FORD MUSTANG 65 45 17000

﻿==> \_

* + multiple command lists can be used as input to a command, this cannot be done with anonymous piping:

==> cp cars cars2==> echo "ford - new car added" >> cars2

==> diff <(grep ford cars) <(grep ford cars2)4a5> ford - new car added

==> \_

* + >( command-list ) - output of a command can be used as input to the command list
  + similar to piping, but output of a command can be sent to multiple command lists (using "tee"):

==> cat cars | tee >(awk '/ford/ {total+=$5} END {print "Total fords: " total}') \>(awk '/chevy/ {total+=$5} END {print "Total chevys: " total}')

plym fury 77 73 2500

chevy nova 79 60 3000

ford mustang 65 45 17000

volvo gl 78 102 9850

ford ltd 83 15 10500

Chevy nova 80 50 3500

fiat 600 65 115 450

honda accord 81 30 6000

ford thundbd 84 10 17000

toyota tercel 82 180 750

chevy impala 65 85 1550

ford bronco 83 25 9525

﻿==> Total chevys: 4550 Total fords: 54025\_

* + standard output of the "tee" can be redirected to /dev/null:

==> cat cars | tee >(awk '/ford/ {total+=$5} END {print "Total fords: " total}') \>(awk '/chevy/ {total+=$5} END {print "Total chevys: " total}') \>/dev/null

﻿==> Total chevys: 4550 Total fords: 54025\_

* + the processes created are independent, execution order is indeterminate:

==> echo 1 | tee >(echo 2) >(echo 3) >(echo 4) > >(cat)342

﻿==> 1echo 1 | tee >(echo 2) >(echo 3) >(echo 4) > >(cat)243

﻿==> 1\_

* + the processes can be synchronized, in this example using sleep:

==> echo 1 | tee >(sleep 2; echo 2) >(sleep 3; echo 3) >(sleep 4; echo 4) > >(sleep 5; cat)

==> 2341echo 1 | tee >(sleep 2; echo 2) >(sleep 3; echo 3) >(sleep 4; echo 4) > >(cat)

==> 1234\_

**Advanced Permissions**

There are actually 12 permission bits, not just the 9 commonly used

* + 1 - Set UID bit - when executing this file, use the userid of the file's owner (rather than the userid of the user executing the file)
  + 2 - Set GID bit - when executing the file, use the groupid of the file (rather than the groupid of the user executing it)
  + 3 - Sticky bit - permit only the owner of this directory, or the owner of a contained file, to delete or overwrite the file
  + 4, 5, 6 - Owner read, write, execute permission
  + 7, 8, 9 - Group read, write, execute permissions
  + 10, 11, 12 - Other read, write, execute permissions

**Setting Permissions**

Absolute method: chmod wxyz filename

* + "wxyz" are octal digits
  + first digit: Set UID has value of 4, Set GID has value of 2, Sticky bit has value of 1
  + second digit: Owner read has value of 4, write has value of 2, execute has value of 1
  + third digit: Group read has value of 4, write has value of 2, execute has value of 1
  + fourth digit: Other read has value of 4, write has value of 2, execute has value of 1

Relative method: chmod who operation permission filename

* + "who" can be u (user), g (group), o (other), or a (all)
  + "operation" can be + (add), - (remove), or = (set)
  + "permission" can be r (read), w (write), x (execute), s (set UID or set GID, depending on "who"), t (sticky bit)

**Displaying Permissions**

ls -l displays only 9 characters for permissions, usually: rwxrwxrwx

* + "x" for user will show "S" if file has set UID permission, or "s" if file has both set UID and execute permissions
  + "x" for group will show "S" if file has set GID permission, or "s" if file has both set GID and execute permissions
  + "x" for other will show "T" if file has sticky bit set, or "t" if file has both sticky bit set and execute permission

×