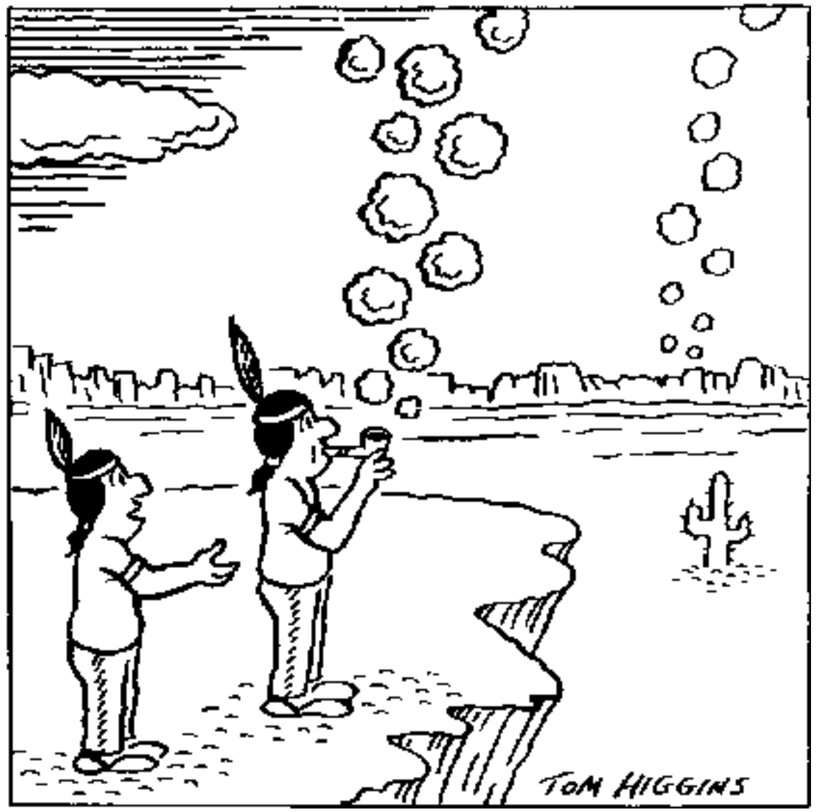
Pipes and Signals



"The best way to predict the future is to invent it."

Alan Kay

THE UNIVERSITY OF SYDNEY

Communicating Processes

- Processes can be started using the exec system calls
- Processes can operate in parallel using the fork system call followed by exec
- Processes sometimes wish to cooperate and exchange information during execution
- Pipes and signals are one way to manage cooperating processes

File Descriptors

- low level I/O is performed on file descriptors that are small integers indicating an open file
- when process is started file descriptor 0 is standard input, 1 is standard output, 2 is standard error output
- low level system call functions operate on file descriptors

THE UNIVERSITY OF SYDNEY

I/O sys

“If I was to change anything in Unix it would be to spell creat with an e”
Ken Thompson

- low level I/O functions include:
  - creat
  - open, close
  - read, write
  - ioctl
- eg read 100 characters from standard input into array “buffer”

```
read(0,buffer,100)
```

pipe

```
int pipe(int filedes[2]);
```

- filedes is a two element array of integers that is filled in with two file descriptors
- filedes[0] is for reading
- filedes[1] is for writing
- data written into filedes[1] can be subsequently read from filedes[0]
- the pipe function returns 0 on success, -1 on failure

Using pipes

- a parent process can communicate with a child by creating a pipe before the fork
- the parent can then write data to fildes[1] and the child can read filedes[0]
- the system has a small amount of buffering
- if the buffer is filled, the writer is suspended until the reader has read some data

Signals

- another one process can communicate with another is using a *signal*
- these are a form of *software interrupt*
- execution is interrupted and a function call is made at that point to a user specified function
- when the function returns, execution is resumed

Signals

- signals can be generated by one process to another using the *kill* system call
- signals are also generated by the operating system, eg when an access outside memory bounds is attempted (Segmentation Fault)

SIGHUP	1	Hangup
SIGINT	2	Interrupt
SIGQUIT	3	Quit
SIGILL	4	Illegal Instruction
SIGTRAP	5	Trace or Breakpoint Trap
SIGABRT	6	Abort
SIGEMT	7	Emulation Trap
SIGFPE	8	Arithmetic Exception
SIGKILL	9	Kill
SIGBUS	10	Bus Error
SIGSEGV	11	Segmentation Fault
SIGSYS	12	Bad System Call
SIGPIPE	13	Broken Pipe
SIGALRM	14	Alarm Clock
SIGTERM	15	Terminated
SIGUSR1	16	User Signal 1
SIGUSR2	17	User Signal 2

kill

- You can send a signal to a running process from the command line using the kill command

- Eg 	kill -9 12345

Will send the SIGKILL signal to process 12345.

- Some signals can be *caught* and handled by a user supplied function
- Some signals (such as SIGKILL) cannot be caught and caused the process to be terminated

kill

- You can send a signal to a running process using the kill system call function

```
#include <sys/types.h>
#include <signal.h>

Int kill (pid_t pid, int sig);
```

Where pid is the process ID of the process to be signaled and sig is the signal to be sent.

Catching Signals

- You can “catch” a signal by specifying a function that is called when the signal is received
- This is done using the signal function:

```
#include <signal.h>

void (*signal(int sig, void (*catch)(int)))(int);
```

This complicated looking declaration means that signal is called with 2 arguments: the first is the signal to catch, the second is a pointer to the function that will be called when the signal is received. The signal function returns a pointer to the function that previously caught the signal ….phew.

```
int toolong;

int wakeup()
{
          toolong = 1;
}

…
toolong = 0;
signal(SIGALRM, wakeup);
alarm(10);
scanf(…);
if (toolong == 1)
          /* signal was caught*/
          …
```

Summary

- For simple communications between processes a pipe can be used
- For communications where a process needs to be interrupted, signals can be used
- There are many other interprocess communication methods and techniques

- picture acknowledgement:

  http://www.pipes.org/Ephemeris

End of segment