

Concurrent Architectures: UNIX Part #1

Alistair Rendell

©2009 School Computer Science, Australian National University

UNIX

- Origins – The Multics Project
 - MIT and GE
 - Ken Thomson and Dennis Ritchie
 - The C language
 - PDP 11
- Features
 - Hierarchical file system
 - Compatible I/O (file, device & inter-process)
 - Command language choice
 - Portability
- Decendents
 - Version 7, BSD 4.2, SystemV, Solaris, XINU, Minix, Linux

©2009 School Computer Science, Australian National University

The Single UNIX® Specification History & Timeline

1969	The Beginning	The history of UNIX starts back in 1969, when Ken Thompson, Dennis Ritchie and others started working on the "Bell-lab PDP-7 in a corner" at Bell Labs and what was to become UNIX.
1971	First Edition	It had an assembler for a PDP-11/20, the system, <code>fork()</code> , <code>read</code> and <code>ed</code> . It was used for text processing of patent documents.
1972	First UNIX Installation	The first installations had 3 users, no memory protection, and a 500 KB disk.
1973	Fourth Edition	UNIX leaves home. Also widely known as Version 6, this is the first to be widely available outside of Bell Labs. The first BSD version (1.0) was derived from V6.
1975	Sixth Edition	It was an "improvement" over all preceding and following Unixes (Bourne). It had C, UUCP and the Bourne shell. It was ported to the VAX and the kernel was more than 40 Kbytes (95).
1979	Seventh Edition	It was an "improvement" over all preceding and following Unixes (Bourne). It had C, UUCP and the Bourne shell. It was ported to the VAX and the kernel was more than 40 Kbytes (95).
1980	Xenix	Microsoft introduces Xenix. 32V and 4BSD introduced.
1982	System III	AT&T UNIX System Group (USG) release System III, the first public release outside Bell Laboratories. SunOS 1.0 ships. HP-UX introduced. Unix-11 introduced.
1983	System V	Compaq Research Group, UNIX System Group (USG) and a third group merge to become UNIX System Development Lab. AT&T announces UNIX System V, the first supported release. Installed base 45,000.
1984	4.3BSD	University of California at Berkeley releases 4.3BSD, includes TCP/IP, new signals and much more. X/Open formed.
1984	SVR2	System V Release 2 introduced. At this time there are 100,000 UNIX installations around the world.
1990	4.3BSD	4.3BSD released, including internet name server. SVID introduced. NFS shipped. AIX announced. Installed base 250,000.
1987	SVR3	System V Release 3 including STREAMS, TLI, RFE. At this time there are 750,000 UNIX installations around the world. IRIX introduced.
1988	PODIX	PODIX published. Open Software Foundation (OSF) and UNIX International (UI) formed. Unix 4.2 ships.
1989	AT&T UNIX Software Operations	AT&T UNIX Software Operations formed in preparation for spinoff of UNIX development group. Motif 1.0 ships.
1989	UNIX System V Release 4	UNIX System V Release 4 ships, unifying System V, BSD and Xenix. Installed base 1.2 million.
1990	XPG3	X/Open introduces XPG3 Brand. OSF/1 debut. Plan 9 from Bell Labs ships.

"The Single UNIX Specification brings all the benefits of a single standard operating system, namely application and information portability, scalability, flexibility and freedom of choice for customers"

Allen Brown, President and CEO, The Open Group

The Story of the License Plate...

In 1983 Digital Equipment Corporation (DEC) was ramping up their engineering group to create and ship their first UNIX system product. One of the stewards of the group was Armando J. Armando.



one made of it, but instead of saying "Live Free or Die" across the bottom of the plate (as in the real case), it had it across the top. Across the bottom was the trademark

acknowledgement

UNIX. Each UNIX license plate up in small numbers and hand-drawn sheets out at events. They usually ran out. The demand for the license plates never got small. People saw them on an office wall, or heard about them somewhere, and wanted one of their own.

Armando left the state for the many classes of California, and had taken his car and license plate with him. Or so many people thought. In 1989 Jan "maddog" Hall was purchasing a new car, a new wrangler. And of course the license plate had to be relevant. So Jan, a long time DEC employee and UNIX system guru, submitted his application with many variations and the clerk said "I think we can give you your first choice..." and gave him the temporary paper plates (to be used on the car until the metal plates were manufactured) with "UNIX" on them. And so it has been ever since. Jan's wrap has been the holder of the UNIX license plate.

The Open Group thanks Jan "maddog" Hall for sharing the story of the UNIX license plate.

The UNIX Brand

The UNIX Brand is used to identify products that have been certified as conforming to the Single UNIX Specification, initially UNIX 93, followed subsequently by UNIX 95, UNIX 98 and now UNIX 03.

UNIX is a registered trademark of The Open Group in the United States and other countries.

1991		UNIX System Laboratories (USL) becomes a company, majority-owned by AT&T, Linux. Novell commences Linux development. Solaris 1.0 debut.
1992	SVR4.1	USL releases UNIX System V Release 4.1 (DevKit). October - X/Open Brand launched by X/Open. December 23rd - Novell announces intent to acquire USL. Solaris 2.0 and HP-UX 9.0 ships.
1993	4.4BSD	4.4BSD the final release from Berkeley. June 16 - Novell acquires USL.
Late 1993	SVR4.1MP	Novell decides to get out of the UNIX business. Rather than sell the business as a single entity, Novell transfers the rights to the UNIX trademark and the specification to X/Open Company. COSE Business delivers "Spec 1197" to X/Open for trademark. In December Novell ships SVR4.1MP, the final USL OEM release of System V.
1994	Single UNIX Specification	BSD + +-Line eliminated all code claimed to infringe on USL/Novell. As the owner of the UNIX trademark, X/Open introduces the Single UNIX Specification (formally Spec 1170) which separates the UNIX trademark from any actual code stream itself, thus allowing multiple implementations.
1995	UNIX 95	X/Open introduces the UNIX 95 branding program for implementations of the Single UNIX Specification. Novell sells UnixWare business to SCO. Digital UNIX introduced. UnixWare 2.0 ships. OpenServer 3.0 debut.
1996		The Open Group forms as a merger of the Open Software Foundation (OSF) and X/Open. UnixWare 2.1, HP-UX 10.20 and IRIX 6.2 ship.
1997	Single UNIX Specification, Version 2	The Open Group introduces Version 2 of the Single UNIX Specification, including support for real-time, shared and 64-bit and larger processors. The specification is made freely available on the web. IRIX 6.4, AIX 4.3 and HP-UX 11 ship.
1998	UNIX 98	The Open Group introduces the UNIX 98 family of brands, including Base, Workstation and Server. First UNIX 98 registered products shipped by Sun, IBM and NEC. The Open Group announcement starts to take off with announcements from NetScope and IBM. UnixWare 7 and IRIX 6.5 ship.
1999	UNIX at 30	The UNIX system reaches thirty. Solaris 7 ships. Linux 2.3 kernel released. The Open Group and the IEEE commence joint development of a revision to PODIX and the Single UNIX Specification. First LinuxWorld conference. Dot com fever on the stock market. True64 UNIX ships.
2001	Single UNIX Specification, Version 3	Version 3 of the Single UNIX Specification unites IEEE PODIX, The Open Group and the industry efforts. Linux 2.4 kernel released. The value of procurement of open system referencing the UNIX brand exceeds 155 billion. AIX 5L ships.
2003	ISO/IEC 9945	The core volume of Version 3 of the Single UNIX Specification are approved as an international standard "Whatnow?" test suites shipped for UNIX 03 brand. Solaris 9.0 ships. Linux 2.6 kernel released.

THE Open GROUP
Making statements work.™

Unix Process Reproduction

- Unix process creation is by cloning
- Standard Unix C library contains function `fork`
- Typical usage:


```
pid = fork();
```
- `fork` spawns a duplicate
 - same memory, regs, PC
- `fork` returns
 - 0 to child
 - and to the parent
 - PID (of child) if it works
 - -1 if it fails
 - never 0!

©2009 School Computer Science, Australian National University

Unix Processes: Overwriting

- Maybe another program is more suitable?
 - Passing responsibility
 - Many computations have quite different phases
 - A child process may need to do a very different job
- Use exec family

```
exec(filename, args);
```
- Does following
 - Memory is replaced by the image given in the executable file
 - Registers initialized in the usual way for a program
 - Parameters are passed as if the new program was called from the shell
 - Only returns in case of a failure

©2009 School Computer Science, Australian National University

Exec Example

```
if (Fork()==0) {
    execl("/usr/bin/grep",
          arg2("grep", keyword));
    abort();
} else {
    <rest of PARENT program>
}
```

- Questions
 - what is the child?
 - what is concurrent?
 - why does grep appear twice?
 - why might it abort?

©2009 School Computer Science, Australian National University

Process Requiem!

- Co-ordination can be a problem
 - A child was spawned to do something in parallel (concurrently)
 - At some point the parent must be sure it is complete
- The `wait` primitive solves it
 - waits for termination of a (any) child
 - returns the PID of the child that has died
 - argument gives exit code for the child

©2009 School Computer Science, Australian National University

Example of Wait

```
if (Fork() == 0){
    <Child's task>
    exit(0);
} else {
    <Parent's task>
    wait (Codes);
}
```

- Concurrency?

©2009 School Computer Science, Australian National University

Unix Processes Dozing

- Why wait for time to pass?
 - It may want to do something periodically
 - It can make sure other processes get a chance
- The `sleep` primitive solves it

```
int sleep(int secs);
```

 - The supplied argument is the duration (in seconds)
 - May return early if woken up (by a signal)
 - Results indicate sleep-time remaining
 - Use of `sleep(0)`

©2009 School Computer Science, Australian National University

UNIX Shells

- Job control language
 - Pre-Unix JCLs were proprietary and idiosyncratic
- Not part of Operating System
 - But must reflect its facilities
 - Can have a shell per user
- There are many common ones
 - `sh` – Bourne shell
 - `csh` – C shell
 - `ksh` – Korn shell
 - `tcsh` – Extended C-shell
 - `bash` – Bourne-again shell

©2009 School Computer Science, Australian National University

UNIX Shells (2)

- Features of the Shell
 - A virtual machine
 - Has a language
 - can invoke programs
 - types, variables, control
 - Can write programs
- Access to Unix's hierarchical file system supported by `cd`, `pwd`, `ls`, etc
- Uniform I/O is supported by re-direction of `stdin` and `stdout`

```
ls -als > DIR
ls -als > /dev/null
ls -als | grep def
```

©2009 School Computer Science, Australian National University

UNIX Shells (3)

- Processes supported by shells

```
ps
emacs assignment1.adb &
ps -aux | grep u9876543
```
- Can write executable programs in shell code
 - Usual termed a script
- Interpreted, not compiled
- First line of file indicates the appropriate interpreter

©2009 School Computer Science, Australian National University

C-Shell Scripts (Example)

- Example of shell script

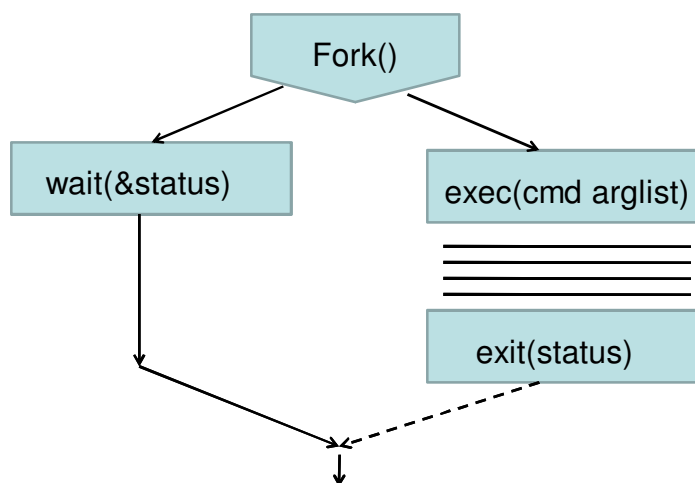
```
#!/bin/csh
set filist = 'cat $1'
foreach nam ( $filist )
  if (-e $nam) then
    cp $nam $2/$nam.bak
  else
    echo $nam "does not exist"
  endif
end
```

- What does it do?
- What does \$1 and \$2 refer to?

©2009 School Computer Science, Australian National University

Command-Line Interpretation

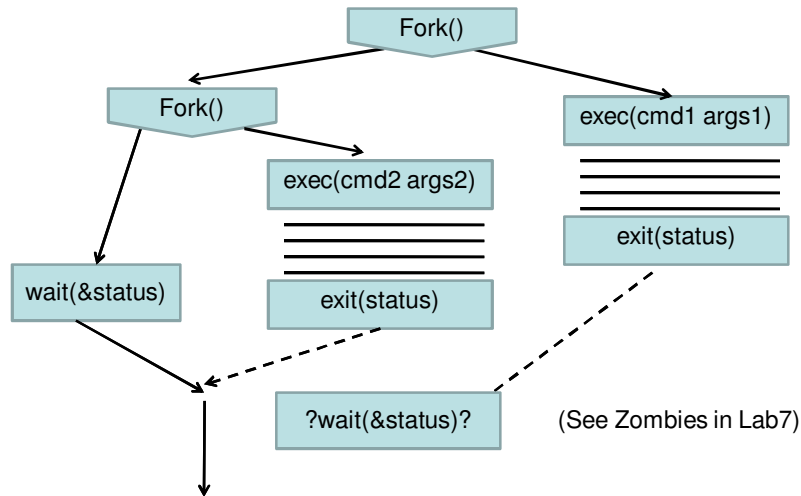
- cmd arglist



©2009 School Computer Science, Australian National University

Backgrounding a Command

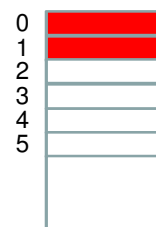
- `cmd1 args1 & cmd2 args2`



©2009 School Computer Science, Australian National University

UNIX File Descriptors

- What are they?
 - Non-negative integers denoting an open I/O channel
 - They index the File Descriptor Table (FDT) associated with each current process
 - The indexes (0,1,2,...etc.) are file descriptors
 - The corresponding table entries mean something to the operating system!



©2009 School Computer Science, Australian National University

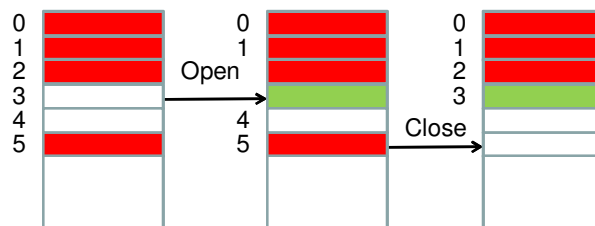
File Descriptors

- Three channels are standard
 - `stdin` file descriptor 0
 - `stdout` file descriptor 1
 - `stderr` file descriptor 2
- Initially
 - `stdin` keyboard
 - `stdout` current xterm window
 - `stderr` current xterm window
- FDT is preserved across `exec`
- FDT copied during `fork`

©2009 School Computer Science, Australian National University

Opening and Closing Files

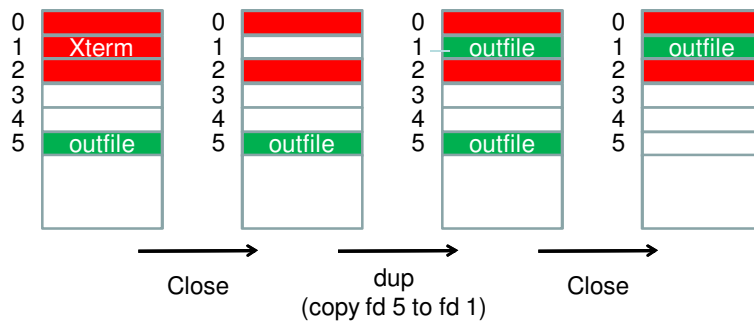
- Opening a file
 - first free entry in the FDT is chosen
- Closing a file
 - Entry in FDT is marked free
 - Opposite to open!



©2009 School Computer Science, Australian National University

Redirecting I/O

- Redirecting for example stdout
 - User cannot change FDT directly
 - There is a primitive `dup` for this purpose



©2009 School Computer Science, Australian National University

UNIX Pipes

- Consider
 - `ps -aux | grep 666`
 - `ps` writes to `stdout`
 - `grep` reads from `stdin`
 - I/O has been re-directed by the shell
 - Shell doesn't handle the characters
- Mental model of a pipe



- It's a buffer for characters
- Handled by I/O sub-system of Unix
- May only be one-way (Solaris has two-way)

©2009 School Computer Science, Australian National University

UNIX Pipes Dependencies



- P₂ will wait for production
- P₁ will not wait for consumption
- There may also be buffering in P₁
- The importance of timely flushing!
 - what's this mean?
- So what sort of communication primitive is a pipe?

©2009 School Computer Science, Australian National University

Running a Shell as a Subprocess

- Can have a shell to do a job
 - Unix allows easy creation of subprocesses
 - A subprocess can run a shell script
- An easy-to-code method

```
    csh (cmd)
```

 - cmd is shell code
 - A process is created to run cmd
 - Result is an input stream on which output of cmd is delivered

©2009 School Computer Science, Australian National University

Creating UNIX Pipes

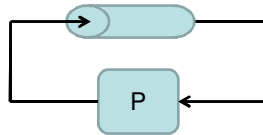
- Opening a pipe

```
void pipe (int pipefd[2])
```

- `pipefd[0]` set to file descriptor of input end of pipe

- `pipefd[1]` set to file descriptor of output end of pipe

- The result

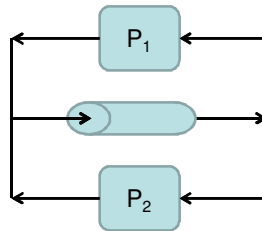


does not connect two processes!

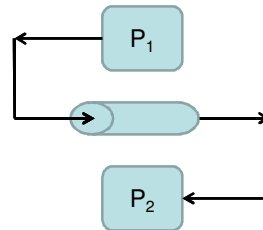
©2009 School Computer Science, Australian National University

Connecting Two Processes by a Pipe

- Fork the process, but do it after opening a pipe



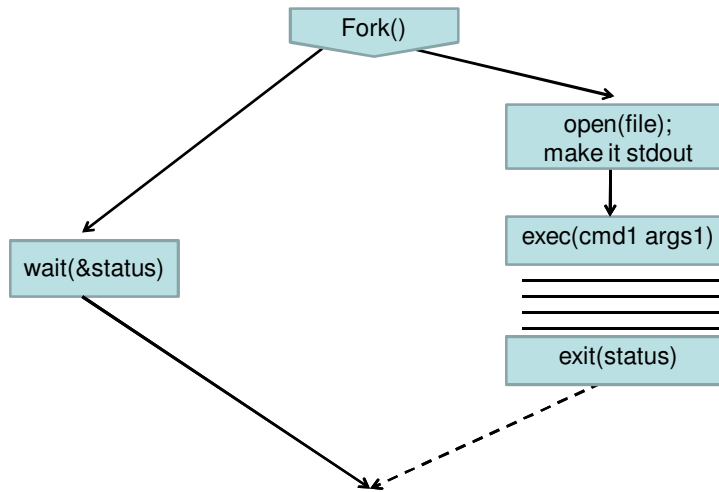
- Close unwanted descriptors



©2009 School Computer Science, Australian National University

Command line I/O Redirection

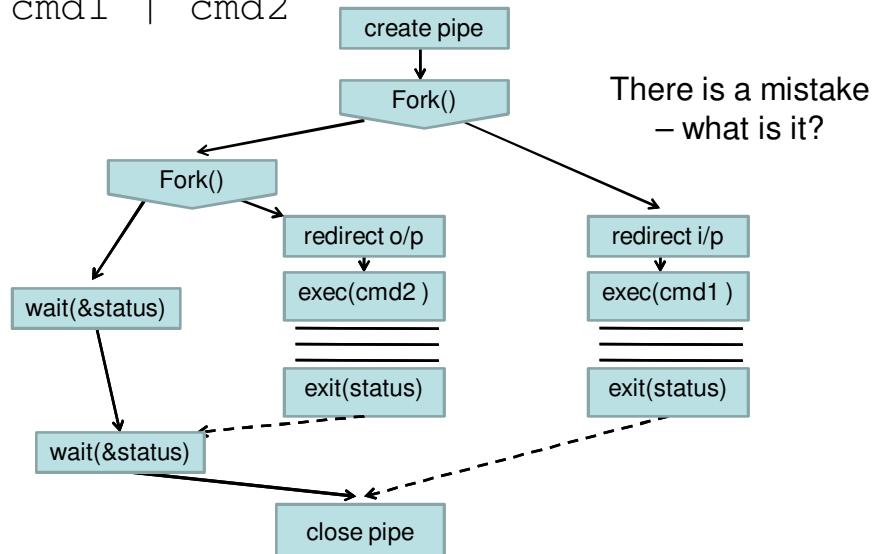
- `cmd1 args1 > file`



©2009 School Computer Science, Australian National University

Command Line Pipes

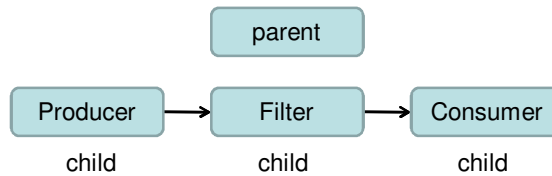
- `cmd1 | cmd2`



©2009 School Computer Science, Australian National University

Subprocess Pipelines

- Desired state

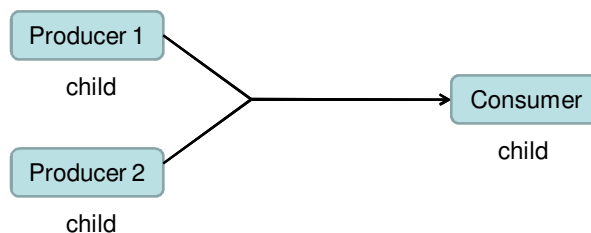


- make pipes (before forking)
- fork two times
- close pipes in parent
- each child closes descriptors that it won't use
- each child does an exec if appropriate

©2009 School Computer Science, Australian National University

Sharing Pipes

- Pipes can be shared



- Integrity of messages
 - Pipe is just a stream of chars
 - No interleaving BUT care required with buffering
 - Messages less than 2k

©2009 School Computer Science, Australian National University

Recall Slide from CDS-L3

```
int data_pipe [2], c, rc;

if (pipe (data_pipe)==-1){
    perror ("no pipe");exit(1);
}

if (fork () == 0) {
    close (data_pipe [1]);
    while ((rc = read
        (data_pipe [0],&c,1))>0){
        putchar (c);
    }
    if (rc == -1) {
        perror ("pipe broken");
        close (data_pipe [0]);
        exit (1);
    }
    close (data_pipe [0]);
    exit (0);
} else {
    close (data_pipe [0]);
    while ((c = getchar ()) > 0) {
        if (write
            (data_pipe[1],&c,1)==-1){
            perror ("pipe broken");
            close (data_pipe [1]);
            exit (1);
        }
    }
    close (data_pipe [1]);
    pid = wait ();
}
```

©2009 School of Computer Science, Australian National University