Systems Programming

updated : 28 sept 2019

Stdio, I/O Buffering, Shells

Overview

Last Time

- Make – implementation
- Process creation – Fork, execve
- System limits

Readings for today

- Stdio
- Chapter 13 – I/O buffering
- Shells:
  - Basics: read command into doubly linked list
  - Shell variables, set command
  - background,
  - Substitutions: variable substitutions, pseudo filename completion, history substitution,
  - Simple I/O redirection
- Shell version 2: signals, pipes, command substitution

Fork revisited

- rv=fork()
- Properties inherited (shorter list last time)
  - Open files, uids, cwd, root directory, umask,
  - signal mask and table,
  - environment, shared ,memory
- Properties not inherited
  - Return value of fork, pid ppid, times,
  - file locks,
  - pending signals

PS EXAMPLES

To see every process on the system using standard syntax:

```
ps -e
ps -ef
ps -eF
ps -ely
```

To see every process using BSD syntax:

```
ps ax
ps axu
```

To print a process tree:

```
ps -ejH
ps axjf
```

To get info about threads:

```
ps -eLf
ps axms
```

To get security info:

```
ps –eo euser, ruser, suser, fuser, f, comm, label
ps axZ
ps –eM
```

To see every process running as root (real & effective ID) in user format:

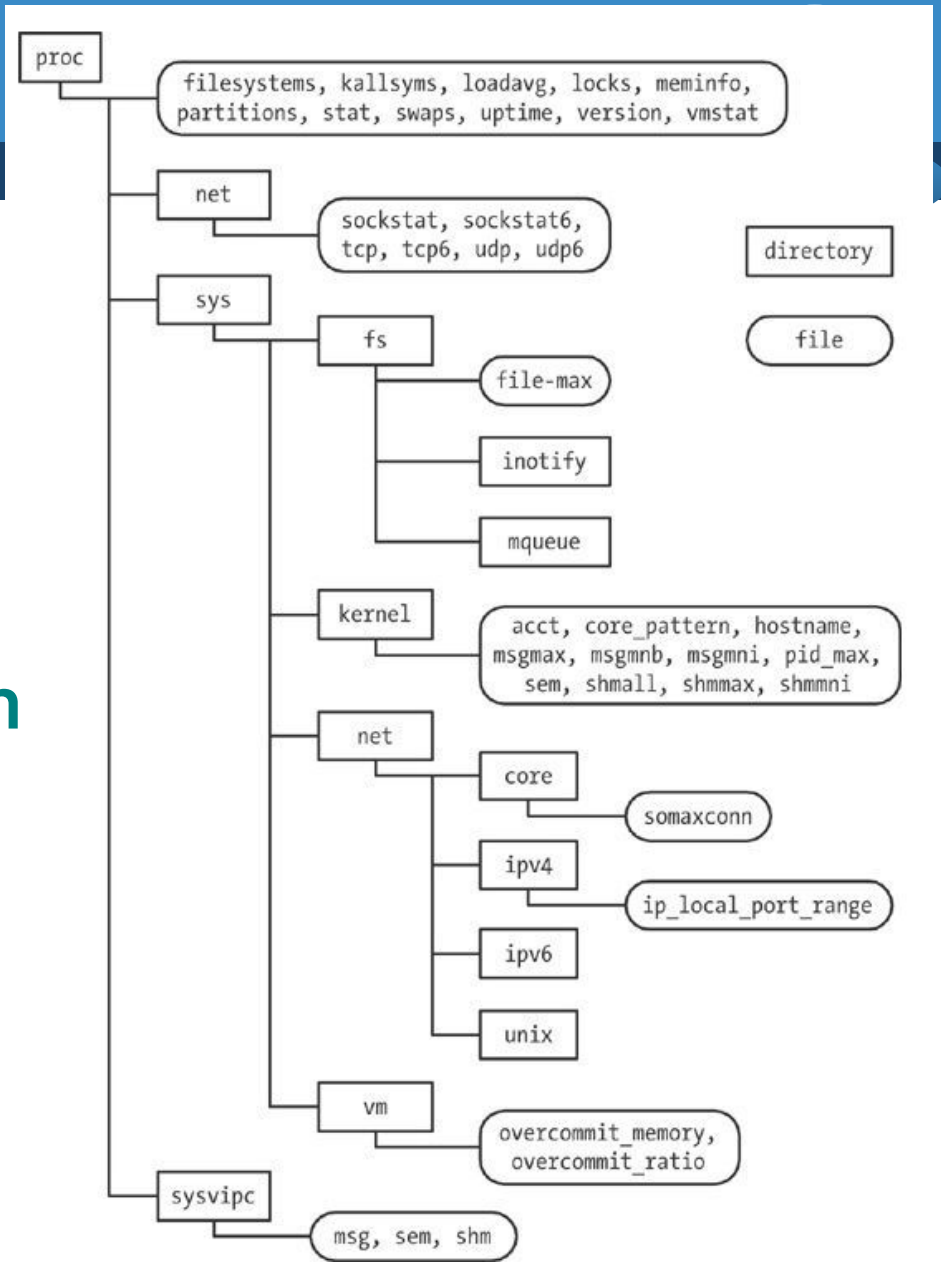
```
ps -U root -u root u
```

Print only the process IDs of syslogd:

```
ps -C syslogd -o pid=
```

Figure 12-1 /proc hierarchy

proc : virtual file system



FILE I/O Buffering

- 13.1 Kernel Buffering of File I/O: The Buffer Cache
- 13.2 Buffering in the *stdio* Library
- 13.3 Controlling Kernel Buffering of File I/O
- 13.4 Summary of I/O Buffering
- 13.5 Giving the Kernel Hints About I/O Patterns: *posix_fadvise()*
- 13.6 Bypassing the Buffer Cache: Direct I/O
- 13.7 Mixing Library Functions and System Calls for File I/O
- 13.8 Summary
- 13.9 Exercises

Kernel Buffering of File IO

- When working with disk files, the read() and write() system calls don't directly initiate disk access. Instead, they simply copy data between a user-space buffer and a buffer in the kernel buffer cache

- The following call transfers 3 bytes of data from a buffer in user-space memory to buffer in kernel space:
- write(fd, "abc", 3);

Why buffer?

Buffering in the Stdio - library

- **Open File Descriptor**
  `fd = open();`
- **Setting the buffering mode of a stdio stream**
  `setvbuf(fd, buffer, mode, buffer-size);`
- **Flushing a stdio buffer**
  `fflush(fd);`
- **Close File Descriptor**
  `close();`

FILE structure

Used to be in `/usr/include/stdio.h`

```
int _fileno;
int _blksize;
int _flags2;
int _cnt;
_IO_off_t _old_offset; /* This used to be _offset but...*/
```

Now, cleverly `/usr/include/stdio.h` contains

```
typedef struct _IO_FILE FILE;
```

FILE structure

```
struct _IO_FILE {
  int _flags;           /* High-order
word is _IO_MAGIC; rest is flags. */
#define _IO_file_flags _flags


  /* The following pointers correspond
to the C++ streambuf protocol. */
  /* Note:  Tk uses the _IO_read_ptr and
_IO_read_end fields directly. */
  char* _IO_read_ptr;   /* Current read
pointer */
  char* _IO_read_end;   /* End of get
area. */
  char* _IO_read_base;  /*Start of
putback+get area. */
  char* _IO_write_base; /* Start of put
area. */
  char* _IO_write_ptr;  /* Current put
pointer. */
  char* _IO_write_end;  /* End of put
area. */
  char* _IO_buf_base;   /* Start of
reserve area. */
  char* _IO_buf_end;    /* End of
reserve area. */
  /* The following fields are used to
support backing up and undo. */
  char *_IO_save_base; /* Pointer to
start of non-current get area. */
  char *_IO_backup_base;  /* Pointer
to first valid character of backup
area */
  char *_IO_save_end; /* Pointer to
end of non-current get area. */
...
...
...
...
```

Stdin, stdout, stderr

- extern FILE *stdin;
- extern FILE *stdout;
- extern FILE *stderr;

Which is faster read() or getc()?

- readoneAtTime.c
- getc_version.c
- getchar_version.i
  - getchar and getc are supposed to be macros!!!

Example different performances on different I/O Buffer Size

time dd if=/dev/zero of=/some-file bs=[BUF_SIZE] count=[n * BUF_SIZE]

BUF_SIZE	Time (seconds)			
	Elapsed	Total CPU	User CPU	System CPU
1	107.43	107.32	8.20	99.12
2	54.16	53.89	4.13	49.76
4	31.72	30.96	2.30	28.66
8	15.59	14.34	1.08	13.26
16	7.50	7.14	0.51	6.63
32	3.76	3.68	0.26	3.41
64	2.19	2.04	0.13	1.91
128	2.16	1.59	0.11	1.48
256	2.06	1.75	0.10	1.65
512	2.06	1.03	0.05	0.98
1024	2.05	0.65	0.02	0.63
4096	2.05	0.38	0.01	0.38
16384	2.05	0.34	0.00	0.33
65536	2.06	0.32	0.00	0.32

Table 13-1. Time required to copy

Table 13-1: Time required to duplicate a file of 100 million bytes

BUF_SIZE	Time (seconds)			
	Elapsed	Total CPU	User CPU	System CPU
1	107.43	107.32	8.20	99.12
2	54.16	53.89	4.13	49.76
4	31.72	30.96	2.30	28.66
8	15.59	14.34	1.08	13.26
16	7.50	7.14	0.51	6.63
32	3.76	3.68	0.26	3.41
64	2.19	2.04	0.13	1.91
128	2.16	1.59	0.11	1.48
256	2.06	1.75	0.10	1.65
512	2.06	1.03	0.05	0.98
1024	2.05	0.65	0.02	0.63
4096	2.05	0.38	0.01	0.38
16384	2.05	0.34	0.00	0.33
65536	2.06	0.32	0.00	0.32

TLPI/filebuff/write_bytes.c

Table 13-2: Time required to write a file of 100 million bytes

BUF_SIZE	Time (seconds)			
	Elapsed	Total CPU	User CPU	System CPU
1	72.13	72.11	5.00	67.11
2	36.19	36.17	2.47	33.70
4	20.01	19.99	1.26	18.73
8	9.35	9.32	0.62	8.70
16	4.70	4.68	0.31	4.37
32	2.39	2.39	0.16	2.23
64	1.24	1.24	0.07	1.16
128	0.67	0.67	0.04	0.63
256	0.38	0.38	0.02	0.36
512	0.24	0.24	0.01	0.23
1024	0.17	0.17	0.01	0.16
4096	0.11	0.11	0.00	0.11
16384	0.10	0.10	0.00	0.10
65536	0.09	0.09	0.00	0.09

SetBuffer

SETBUF(3) Linux Programmer's Manual SETBUF(3)

NAME

setbuf, setbuffer, setlinebuf, setvbuf - stream buffering operations

SYNOPSIS

```
#include <stdio.h>
 void setbuf(FILE *stream, char *buf);
 void setbuffer(FILE *stream, char *buf, size_t size);
 void setlinebuf(FILE *stream);
 int setvbuf(FILE *stream, char *buf, int mode, size_t
size);
```

DESCRIPTION The three types of buffering available are unbuffered, block buffered, and line buffered.

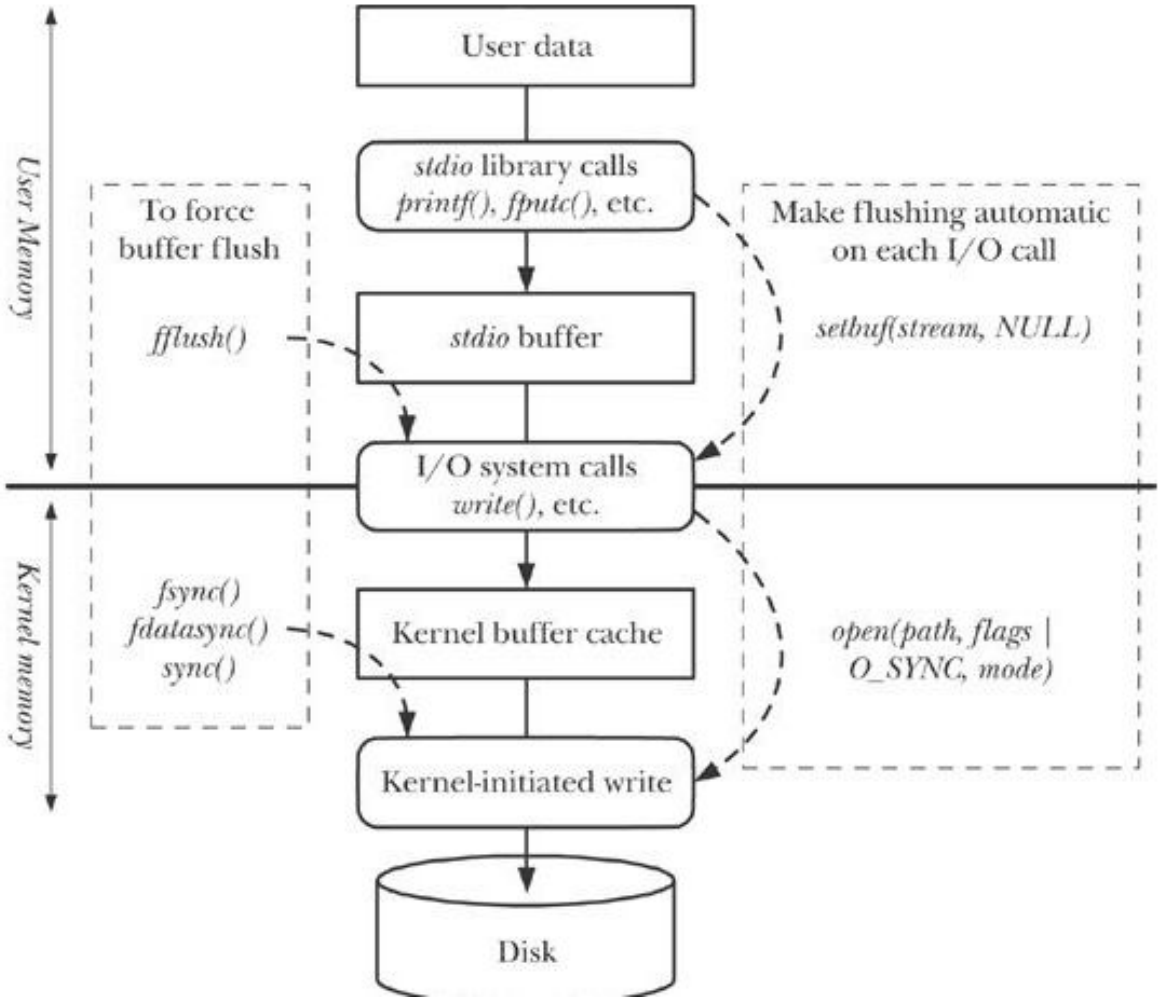
Setvbuf Mode

- _IONBF unbuffered
- _IOLBF line buffered
- _IOFBF fully buffered

Flushing a stdio buffer, etc

- fflush(*stream) -> flush to OS from apps
  - see the manual
- fsync(fd) -> tells OS to flush everything to disk

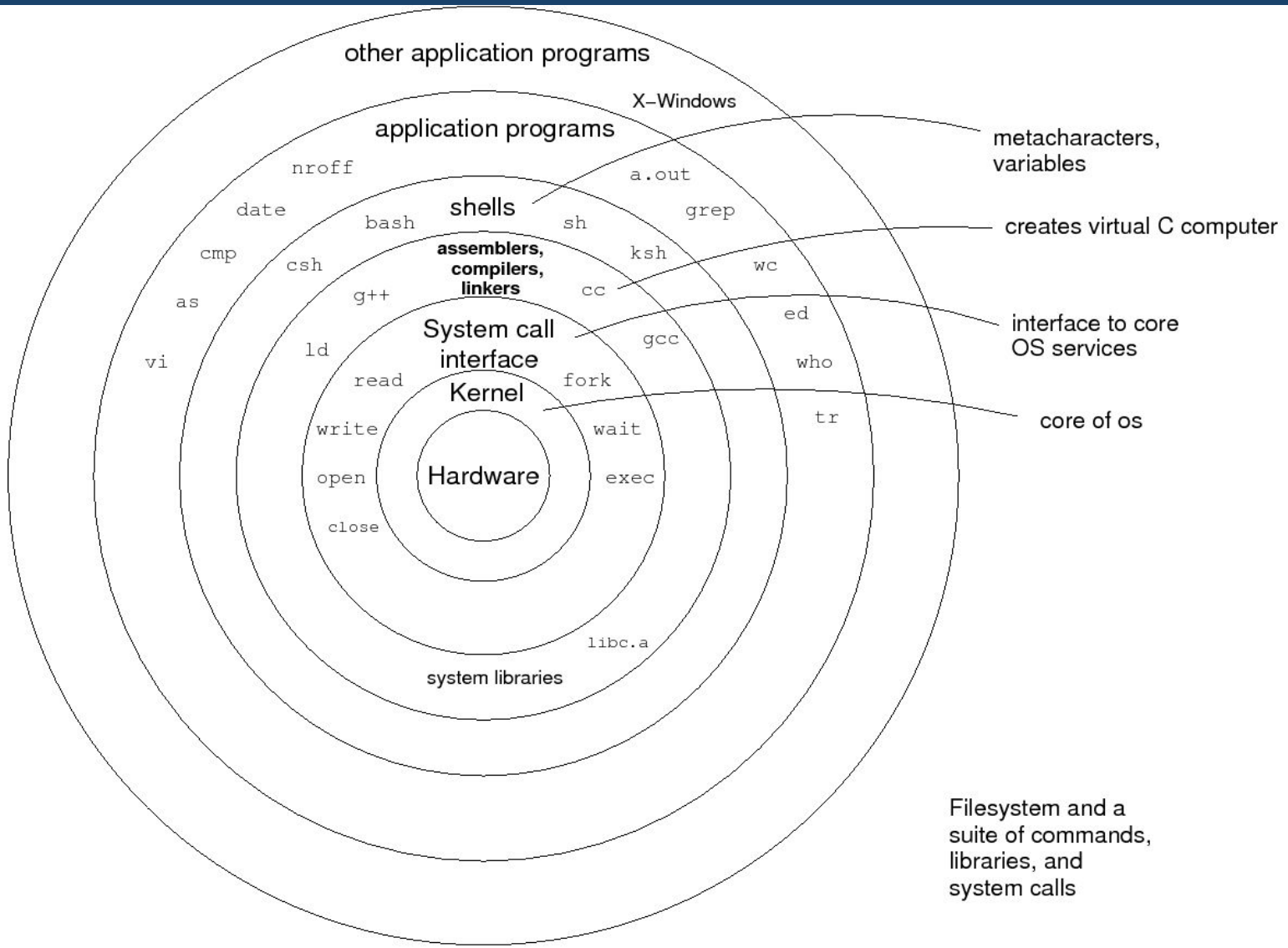
Fig 13-1 Summary of I/O buffering



Shell Implementation

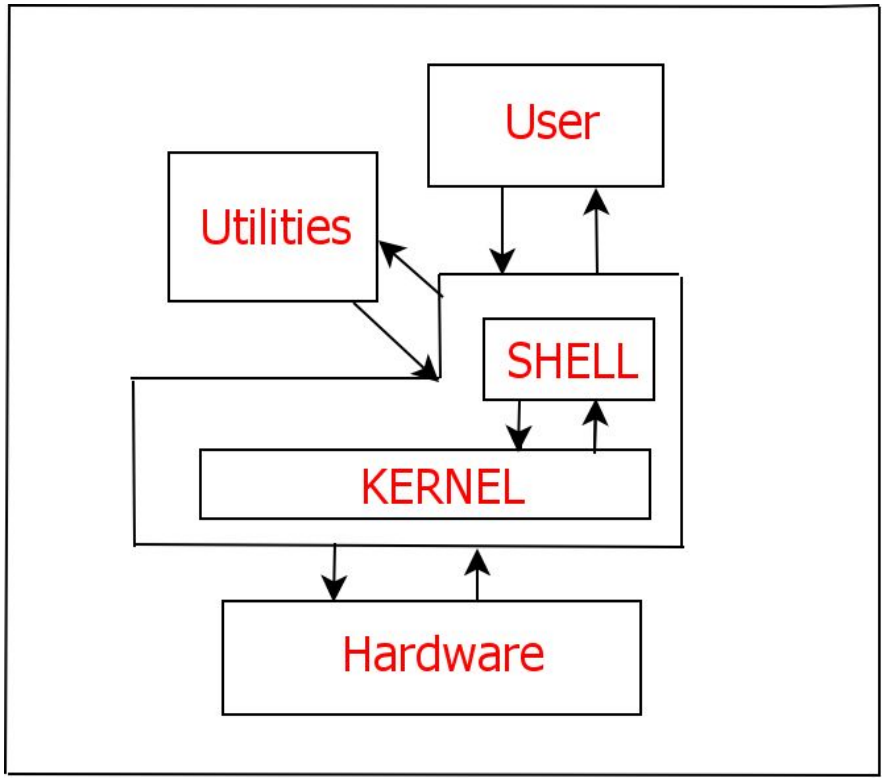
- What does the shell do?

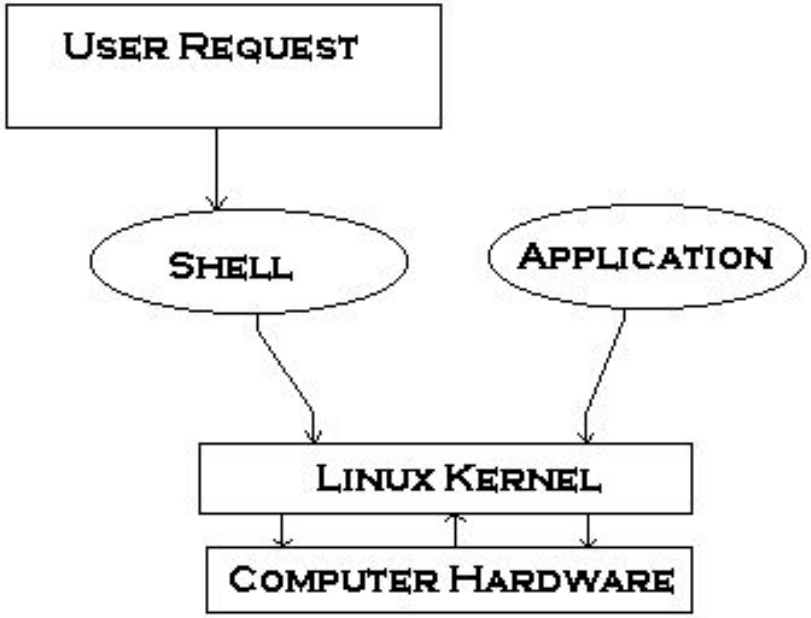
Another Illustration (Shell)



Conceptual Architecture of UNIX SYSTEMS

Another Illustration (Shell)





Bash Reference Manual

- Reference Documentation for Bash
- Edition 4.2, for Bash Version 4.2.
- December 2010

- 166 page pdf
  - http://www.gnu.org/software/bash/manual/bash.pdf

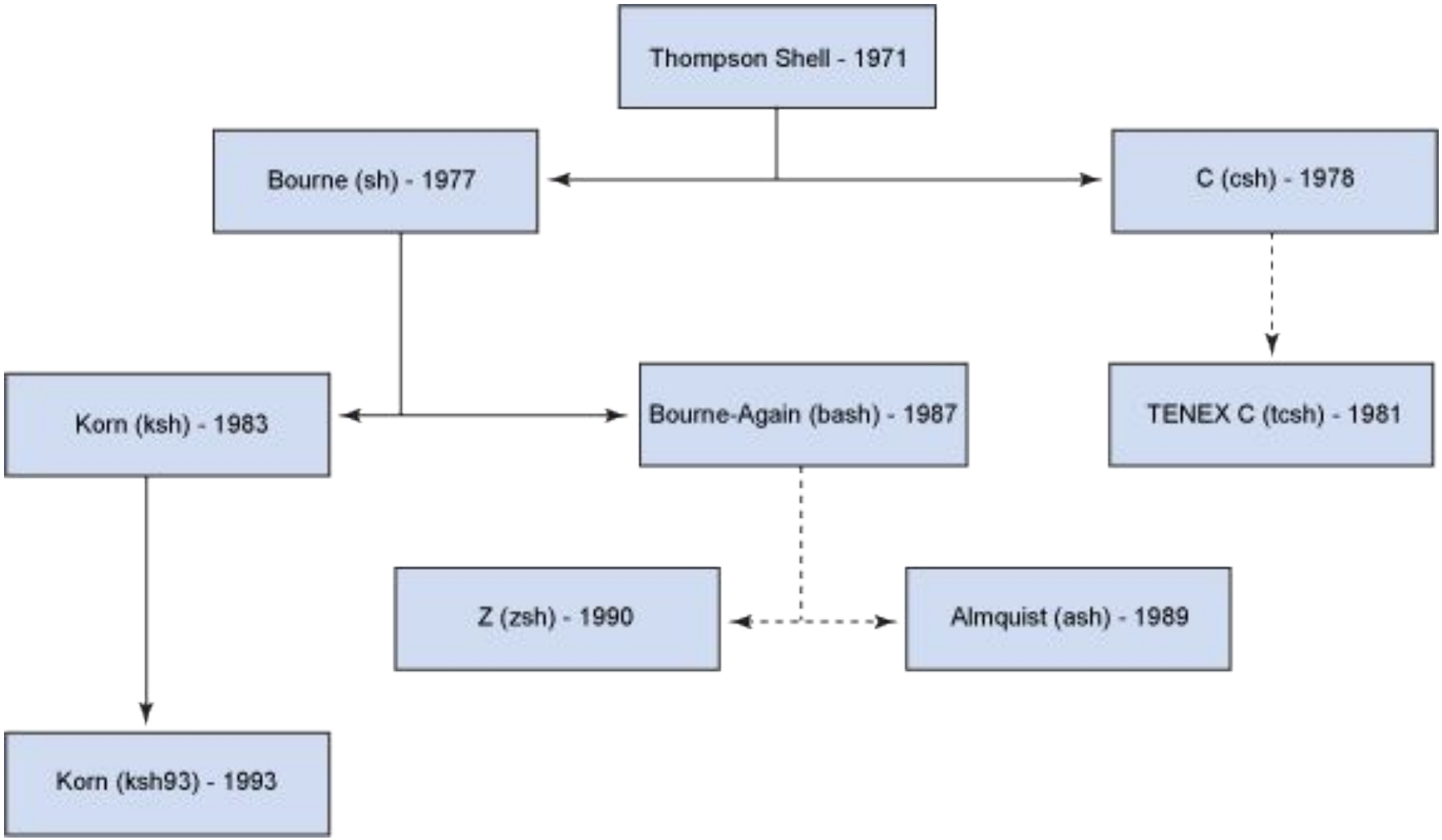
Shell variables / Environment

- We can use Shell as a basic scripting language
- Every scripting Language have Variables

Please open references at:

https://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_03_02.html

Shell History



QA

FILE structure

```
  struct _IO_FILE *_chain;

  int _fileno;
#if 0
  int _blksize;
#else
  int _flags2;
#endif
 struct _IO_marker *_markers;
_IO_off_t _old_offset; /* This
used ...
#define __HAVE_COLUMN /*
temporary */
  /* 1+column number of
pbase(); 0 is unknown. */
  unsigned short _cur_column;
  signed char _vtable_offset;
  char _shortbuf[1];

  /*  char* _save_gptr;  char*
_save_egptr; */

  _IO_lock_t *_lock;
#ifdef _IO_USE_OLD_IO_FILE
};
```