Software Tools

C, Unix (Linux), and tools

Variable Length Argument List

- Standard library printf (and its sisters) has a variable length argument list
- C provides a formal mechanism for writing such functions
- They are not used very often, but are very helpful

How it’s done

- File stdarg.h contains a set of macros for this purpose, plus a data type called va_list
- They are portable, although they me be implemented in a very different way in every system

```
int fun(int arg1, ...);
va_list ap;
va_start(ap, arg1);
va_arg(ap, int); or va_arg(ap, double);
va_end(ap);
```

Example: minprintf

- A very simplified printf from the book

```
void minprintf(char *fmt, ...)
{
  va_list ap;
  /* .... */
  va_start(ap, fmt);
  for (...){
    ...
    Ival = va_arg(ap, int);
    Dval = va_arg(ap, double);
  }
  va_end(ap);
}
```

Files in the standard library

- Files are handled through the standard I/O library
- The library provides a portable uniform and convenient way to handle files

```
#include <stdio.h>

FILE *fopen(const char *pathname, const char *mode);
int fclose(FILE *stream);
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
size_t fwrite(const void *ptr, size_t size, size_t nmemb,
                     FILE *stream);
int ferror(FILE *stream);
```

Standard open files

- Every C program starts life with three files:
  - stdin
  - stdout
  - stderr

Suicides of programs

- If a process wants to terminate it calls exit.
  - A happy process calls exit(0)
  - An unhappy program calls exit(1) or exit(2)
  - In the main program it can also just issue return 0 or return 1, etc.
- A call to exit deallocates everything and releases any resources. Also kills its children!
  - Linux is not meant to provide good family values.

System calls

- The user can ask the OS for some services using system calls
- System calls are like functions but they have a primitive look and feel
- Very often we do not access them directly but through library functions that make them more programmer friendly
- System calls are implemented though software interrupts.

File descriptors

- Files are opened with the system call open
  - Similar to the fopen we saw before
- System call open returns a file descriptor
  - Quite different from the FILE pointer
- File descriptors are small integers

```
int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);
```

Input, Output

- Every process starts life with three open file descriptors
  - Standard input: 0
  - Standard output: 1
  - Stabdard error: 2

```
int getchar(void)
{
  char c;
  return (read(0, &c, 1) == 1) : (unsigned char)c : EOF);
}
```

Files in the standard library

- Files are handled through the standard I/O library
- The library provides a portable uniform and convenient way to handle files

```
#include <stdio.h>

FILE *fopen(const char *pathname, const char *mode);
int fclose(FILE *stream);
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
size_t fwrite(const void *ptr, size_t size, size_t nmemb,
                     FILE *stream);
int ferror(FILE *stream);
```

Directories

- Directories in Unix/Linux are special kinds of files.
- A directory is a list of file names and inode numbers
  - An inode contains all information about the file except its name
    - So two different entries in directories can have the same inode. So a file can have two different names! (Avoid it, of course)
- An inode number is an index to an inode table

Seeking

- We can position the reading or writing “head” on a file anywhere we want.
- System call lseek does this for us.

```
off_t lseek(int fd, off_t offset, int whence);
```

Errors

- If a system call encounters an error it returns (usually) -1 and sets the errno to the apropriate value

```
EBADF               Bad file descriptor
EACCES              Permission denied
EFBIG               File too large
EINTR               Interrupted  function  call
EINVAL              Invalid argument

See also strerror(3)
```