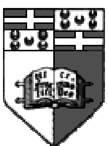
Operating Systems II
Systems Programming in C

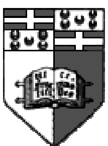
Joseph Cordina
e-mail: joseph.cordina(at)um.edu.mt
Rm 203, New Computing Building
Tel :2340-2254



Course Proposed Syllabus

How many we do will depend on time and your enthusiasm

- Proper C Programming
- File Input / Output
- Process Control
- Signal Handling
- IPC
  - Pipes
  - Semaphores
  - Shared Memory
  - Message Queues
- Socket Programming
- Source to executable process
- Inline assembly
- gcc, gdb, rcs
- Make
- Thread Scheduling, SMP
- Others



Course Details

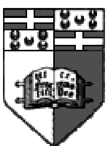
Textbook:

B.W. Kernighan & D. M. Ritchie. The C Programming Language. 2nd ed. Prentice-Hall. ISBN 0-13-110362-8

Haviland K., Gray D. and Salama B. Unix System Programming 2nd ed. Addison Wesley 1998. ISBN 0-201-87758-9

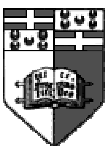
W.R. Stevens. Advanced Programming in the UNIX Environment. Addison Wesley 1993. ISBN 0-201-56317-7

- Lectures
  - 2 hours a week
- Tutorials
  - At your discretion
- SunOS or UNIX compliant system
- Slides will be given yet not complete
- Practice makes Perfect
  - Lectures will merely highlight main points, since its really more practical based
  - There are exercises for each set of slides



Preliminaries

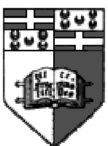
- Usable knowledge of C is necessary
  - Arrays, pointers, strings and relationship between these
  - Scoping and typecasting
  - Structures, pointers to functions, address resolution, #define, #include, typedef, etc.
  - Questions ?
- Compile your programs using gcc (-o output name)
  - Normal output 'a.out'
- Pipelining (|), redirection (<,>,&>) and general UNIX commands
- Debug using gdb
- man pages
  - man <subject>
  - man [-S] <sectionno> <subject>



Preliminaries (cont)

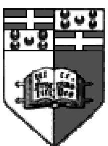
- man Sections
    - 1 – User Commands
    - 2 – System Calls
    - 3 – Library Functions
    - 4 – Special Files
    - 5 – File Formats
    - 6 – Games
    - 7 – Conventions and Miscellaneous
- *man intro* gets you started

- Program editors
    - vi, vim, emacs, xemacs, textedit
    - On Linux: gvim with ‘color syntax’ option



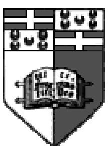
Systems Programming

- An OS kernel provides services which can be utilized by a program.
- An application program is directly related to the problem domain while a system program extends and personalizes services provided by the OS
- System programs are of a more general nature and can be encapsulated in libraries to provide an interface between the application and the kernel.



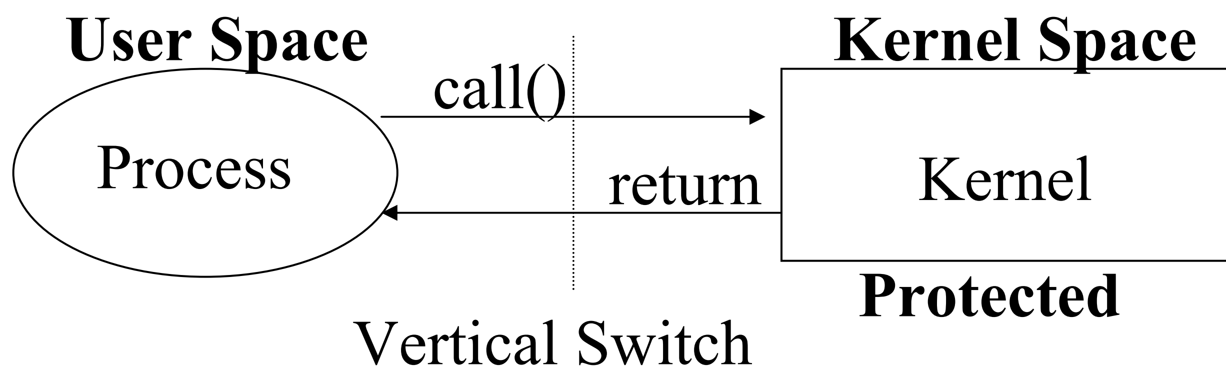
UNIX History

- The UNIX idea developed from a variety of projects at MULTICS and other Bell Lab Projects
- 1973: the kernel written in C
- UNIX Version 7 was developed
- ANSI C (1989)
- There are 2 standards that ensure portability by defining an interface and appropriate limits
  – IEEE POSIX.1 (1990)
  – X/OPEN XPG (1989)
- Implementations
  – AT&T System V Release 4 (SOLARIS)
  – 4.3+BSD 1990s including for 0x86 machines (SunOS)
- Other implementations mix and match standards: Linux, AIX, etc.

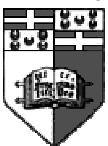


Kernel ⇓◊ Process

- Kernel is the underlying heart of the OS
- The process is the user program



- Most system calls go first through a library call.
- Example:
  - malloc() is provided by a library
  - sbrk() is provided by the kernel
  - Application calls malloc() which calls sbrk() and kernel returns the necessary memory
- Not all library calls enter the kernel for example sin()

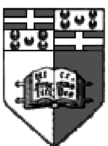


Memory Allocation

- *sbrk()* is a system call that modifies the allocated storage space size given by the kernel to the user process

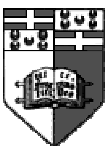
```
#include <stdlib.h>
#include <unistd.h>
void *sbrk(ptrdiff_t increment);
        returns pointer to data or –1 on error
```

- Memory allocated is not guaranteed to be consecutive.
- free is not supported by the kernel.
- Some systems place sbrk at the library level yet this does not affect our programming but affects process efficiency.



Error Handling

- Every function and program in C returns a value. So does every program in UNIX
- By UNIX convention, a return value of 0 is OK, -1 for error, any other for passing back results. A NULL is returned on error from functions returning pointers.
- Every system call on an error sets the integer *errno* in the background.
- <errno.h> defines all the constants for each value of *errno*.
- *errno* is never 0 and retains value until next error result.



Error Handling (cont)

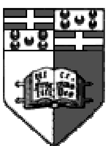
```
#include <string.h>

char *strerror(int errnum)
```

- *strerror* returns a more meaningful string
- *perror* prints message to stderr

```
#include <stdio.h>

void perror(const char *msg)
```



Error Constants

E2BIG Arg list too long

EACCES Permission denied

EAGAIN Resource temporarily unavailable

EBADF Bad file descriptor

EBADMSG Bad message

EBUSY Resource busy

ECANCELED Operation canceled

ECHILD No child processes

EDEADLK Resource deadlock avoided

EDOM Domain error

EEXIST File exists

EFAULT Bad address

EFBIG File too large

EINPROGRESS Operation in progress

EINTR Interrupted function call

EINVAL Invalid argument

EIO Input/output error

EISDIR Is a directory

EMFILE Too many open files

EMLINK Too many links

EMSGSIZE Inappropriate message buffer length

ENAMETOOLONG Filename too long

ENFILE Too many open files in system

ENODEV No such device

ENOENT No such file or directory

ENOEXEC Exec format error

ENOLCK No locks available

ENOMEM Not enough space

ENOSPC No space left on device

ENOSYS Function not implemented

ENOTDIR Not a directory

ENOTEMPTY Directory not empty

ENOTSUP Not supported

ENOTTY Inappropriate I/O control operation

ENXIO No such device or address

EPERM Operation not permitted

EPIPE Broken pipe

ERANGE Result too large

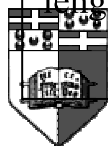
EROFS Read-only file system

ESPIPE Invalid seek

ESRCH No such process

ETIMEDOUT Operation timed out

EXDEV Improper link



Exercises

- Using the man pages, check what is *sysconf, pathconf* and *fpathconf* and how to use them,
- Use *sysconf* to obtain the _SC_CLK_TCK and _SC_OPEN_MAX. What are these constants ?
- Write a program to allocate some memory using malloc().
- Make a loop to allocate memory ad infinitum and see if an error is returned. Print this error using the errno variable. Try allocating a negative amount.