Unix System Programming
Using C++

UNIX And ANSI Standards
The ANSI/ISO C++ standard

- Objective: “To make writing good programs easier and more pleasant for individual programmer”

- 1980’s by Bjarne Stroustrup at AT & T Bell Labs

- ANSI C++ standard developed by WG21 committee of ISO & ANSI X3J16 in 1994
Difference between ANSI C and C++

- All functions must be declared or defined before they can be referenced.

- int foo();
  
  ANSI C: int foo(…);
  
  C++: int foo(void);

- C++ encrypts external function names for type-safe linkage.
The POSIX standards

IEEE society in 1980’s formed a special Task Force called POSIX for creating standards for OS interfacing.

- **POSIX.1**: Basic OS APIs for manipulation of files and processes.

- **POSIX.1b**: Standard APIs for real-time OS interface. It includes Interprocess Communication.

- **POSIX.1c**: Standards for multi-threaded programming interface.
The POSIX standards

To ensure a user program confirms to **POSIX.1 standard**,

```
#define _POSIX_SOURCE

OR

% cc -D_POSIX_SOURCE *.c
```

**POSIX.1b standard,**

```_
_POSIX_C_SOURCE

198808L

199009L

199309L
```

```_
_POSIX_VERSION

in the <unistd.h>
```
Show Posix Version

#define _POSIX_SOURCE
#define _POSIX_C_SOURCE 199309L
#include <iostream.h>
#include <unistd.h>

int main()
{
    #ifdef _POSIX_VERSION
        cout<<“System confirms to POSIX”<<_POSIX_VERSION<<endl;
    #else
        cout<<“_POSIX_VERSION is undefined”;
    #endif
    return 0;
}

Department of MCA PESI, Unix
System Programming.
The POSIX Environment

- `#include <header_file_name>`
  
  Each `#included` file need not be a physical file
  The `/user/include` directory does not have to exist.

- POSIX standards do not mandate that all systems support the concept of `supervisor`.

- Userid of Zero does not have any special privilege.
The POSIX Feature Test Macros

- _POSIX_JOB_CONTROL
- _POSIX_SAVED_IDS
- _POSIX_CHOWN_RESTRICTED
- _POSIX_NO_TRUNC
- _POSIX_VDISABLE
Show test macros

#define _POSIX_SOURCE
#define _POSIX_C_SOURCE 199309L
#include <iostream.h>
#include <unistd.h>

int main()
{
    #ifdef _POSIX_JOB_CONTROL
        cout<<"System supports job control\n";
    #else
        cout<<"System does not support job control\n";
    #endif
}

Department of MCA PESI, Unix
System Programming.
Limits Checking

<limits.h> - has the set of configuration limits

_POSIX_CHILD_MAX
_POSIX_NAME_MAX
_POSIX_PATH_MAX

.......
APIs for configuration limits

#include <unistd.h>

long sysconf(const int limit_name);
long pathconf(const char * pathname, int flimit_name);
long fpathconf(const int fdesc, int flimit_name);

limit_name in <unistd.h> has values
   _SC_CHILD_MAX
   _SC_OPEN_MAX…..etc

flimit_name in <unistd.h> has values
   _PC_PATH_MAX
   _PC_NAME_MAX

Department of MCA PESI, Unix System Programming.
#define _POSIX_SOURCE
#define _POSIX_C_SOURCE 199309L
#include <stdio.h>
#include <iostream.h>
#include <unistd.h>

int main()
{
    int res;
    if((res=sysconf(_SC_OPEN_MAX)) == -1)
        perror(“sysconf”);
    else
        cout<<“OPEN_MAX:”<<res<<endl;
}

Department of MCA PESI,Unix
System Programming.
The POSIX.1 FIPS Standard

FIPS: Federal Information Processing Standard

POSIX.1 FIPS - Developed by NIST (National Institute of Standard and Technology)

- _POSIX_JOB_CONTROL
- _POSIX_SAVED_IDS
- _POSIX_NO_TRUNC
- _POSIX_CHOWN_RESTRICTED
- _POSIX_VDISABLE
- NGROUP_MAX = 8

- Read and write APIs should return the no of bytes after signals.
- Group ID – Inherit that of the parent
The X/Open Standards

*X/Open* organization was formed by a group of *European* companies to propose a common OS interface for their computing systems.

**XPG3 – 1989**

**XPG4 – 1994**

**COSE (Common Open software Environment)** formed in 1993 by HP, IBM, Novell, OSF and Sun Microsystems

**SPEC 1179**

Incorporated in XPG4 as part of X/Open Common Application Environment
UNIX and POSIX APIs

System Calls

- Determine system Config and user info
- Files manipulation
- Process creation and control
- Interprocess Communication
- Network Communication
System Calls

**User mode:** is the normal execution context of any user process. It allows the process to access its process-specific data only.

**Kernel mode:** Is a protective execution environment that allows a user to access kernels data in a restricted manner.

Context switching
The UNIX and POSIX Development Environment

- `<unistd.h>` APIs
- `/usr/include/sys` headers
- `<stdio.h>` - perror function - Called by a user process when API execution fails. It prints the system-defined diagnostic message for any failure.
- lib.a & lib.so
- libsocket.a & libsocket.so – Socket APIs

Department of MCA PESI, Unix System Programming.
API Common Characteristics

**APIs**

- return an integer value indicating the termination status of execution.
- return –1 if API execution has failed.
- set a global variable errno in `<errno.h>`
- user process can call perror/strerror

**Error Status Code**

EACCESS

EPERM

EIO……etc
1. Give the difference between ANSI C and C++.

2. What are the major differences between ANSI ‘C’ and K & R ‘C’? Explain each with examples.

3. What is POSIX? List few POSIX standards. Name a few POSIX APIs.

4. Give the differences between POSIX and UNIX.

5. Give the different POSIX feature test macros. Explain with a program and also give the output of the program.

6. Give the prototype and explain with an example the following APIs.
   a. `sysconf`
   b. `pathconf`
   c. `fpathconf`

7. Expand FIPS. Describe the characteristics of POSIX. 1 FIPS standard and X/open standard.

8. What are the different features to be implemented in all FIPS conforming systems?

9. Explain the common characteristics of API and describe the error status codes.