DEPARTMENT : CSE/ISE

Scheme and Solution for IAT1-March 2017

Unix System Programming (10CS62)

1. POSIX definition -2 Marks

   Each subgroup- 1 Mark each

   Program - 6Marks

   POSIX or “Portable Operating System Interface” is the name of a family of related standards specified by the IEEE to define the application-programming interface (API), along with shell and utilities interface for thesoftware compatible with variants of the UNIX operating system. Because many versions of UNIX exist today and each of them provides its own set of API functions, it is difficult for system developers to create applications that can be easily ported to different versions of UNIX.

   Some of the subgroups of POSIX are POSIX.1, POSIX.1b & POSIX.1c .

   POSIX.1

   This committee proposes a standard for a base operating system API; this standard specifies APIs for the manipulating of files and processes.

   It is formally known as IEEE standard 1003.1-1990[6] and it was also adopted by the ISO as the international standard ISO/IEC 9945:1:1990.

   POSIX.1b

   This committee proposes a set of standard APIs for a real time OS interface; these include IPC (inter- process communication).

   This standard is formally known as IEEE standard 1003.4-1993[7].

   POSIX.1c

   This standard specifies multi-threaded programming interface. This standard is formally known as IEEE standard 1003.4-1993[8]

```
#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_CHILD_MAX))==-1)
perror("sysconf");
else
cout<<"Maximum number of child processes:"<<res<<endl;
if((res=pathconf("/",_PC_PATH_MAX))==-1)
perror("pathconf");
else
cout<<"max path length:"<<(res+1)<<endl;
if((res=pathconf("/",_PC_LINK_MAX))==-1)
perror("pathconf");
else
cout<<"Maximum number of links of a file:"<<res<<endl;
return 0;
}
```

2a) 3 Differences-3 Marks

ANSI C	C++
Uses K&R C default function declaration for any functions that are referred before their declaration in the program.	Requires that all functions must be declared / defined before they can be referenced.
int foo(); ANSI C treats this as old C function declaration & interprets it as declared in following manner. int foo(........); meaning that foo may be called with any number of arguments.	int foo(); C++ treats this as int foo(void); Meaning that foo may not accept any arguments.
Does not employ type_safe linkage technique type_safe and does not catch user errors.	Encrypts external function names for linkage. Thus reports any user errors.

b)3 Differences -3Marks

The major difference between the stream pointer and the file descriptors are as follows:

Stream pointer	FILE descriptor
Stream pointers are allocated via the fopen function call. Eg: FILE *fp; fp=fopen(&&);	File descriptor are allocated via the open system call Eg: int fd; fd=open(&..);
Stream pointer is efficient to use for application doing extensive read from or write to files.	File descriptors are more efficient for applications that do frequent random access of file
Stream pointers is supported on all operating system such as VMS,CMS,DOS and UNIX that provide C compilers	File pointers are used only in UNIX and POSIX 1 compliant systems

c)4 differences -4 Marks
Differences between hard link and symbolic link are listed below:

Hard link	Symbolic link
Does not create a new inode.	It creates a new inode
It increases the hard link count of the file	Does not change the had link count of the file
It can t link directory files, unless it is done by superuser	It can link directory files.
It cant link files across different file system	It can link files across different file system
Eg: ln /urs/cse/abc /usr/cse/xyz	Eg: ln -s /urs/cse/abc /usr/cse/xyz

3.a) Explanation -1mark
Diagram -2 Marks
steps for to open a file -2 Marks
steps for to close a file -2 marks

In UNIX system V, the kernel maintains a file table that has an entry of all opened files and also there is an inode table that contains a copy of file inodes that are most recently accessed. A process, which gets created when a command is executed will be having its own data space (data structure) wherein it will be having file descriptor table. The file descriptor table will be having an maximum of OPEN_MAX file entries. Whenever the process calls the open function to open a file to read or write, the kernel will resolve the pathname to the file inode number.

The steps involved for open() are :

1. The kernel will search the process descriptor table and look for the first unused entry. If an entry is found, that entry will be designated to reference the file .The index of the entry will be returned to the process as the file descriptor of the opened file.

2. The kernel will scan the file table in its kernel space to find an unused entry that can be assigned to reference the file.

If an unused entry is found the following events will occur:

The process file descriptor table entry will be set to point to this file table entry.

The file table entry will be set to point to the inode table entry, where the inode record of the file is stored.

The file table entry will contain the current file pointer of the open file. This is an offset from the beginning of the file where the next read or write will occur.

The file table entry will contain an open mode that specifies that the file opened is for read only, write only or read and write etc. This should be specified in open function call.

The reference count (rc) in the file table entry is set to 1. Reference count is used to keep track of how many file descriptors from any process are referring the entry.

The reference count of the in-memory inode of the file is increased by 1. This count specifies how many file table entries are pointing to that inode.

If either (1) or (2) fails, the open system call returns -1 (failure/error)

The following events will occur whenever a process calls the close function to close the files that are opened.

1. The kernel sets the corresponding file descriptor table entry to be unused.

2. It decrements the rc in the corresponding file table entry by 1, if rc not equal to 0 go to step 6.

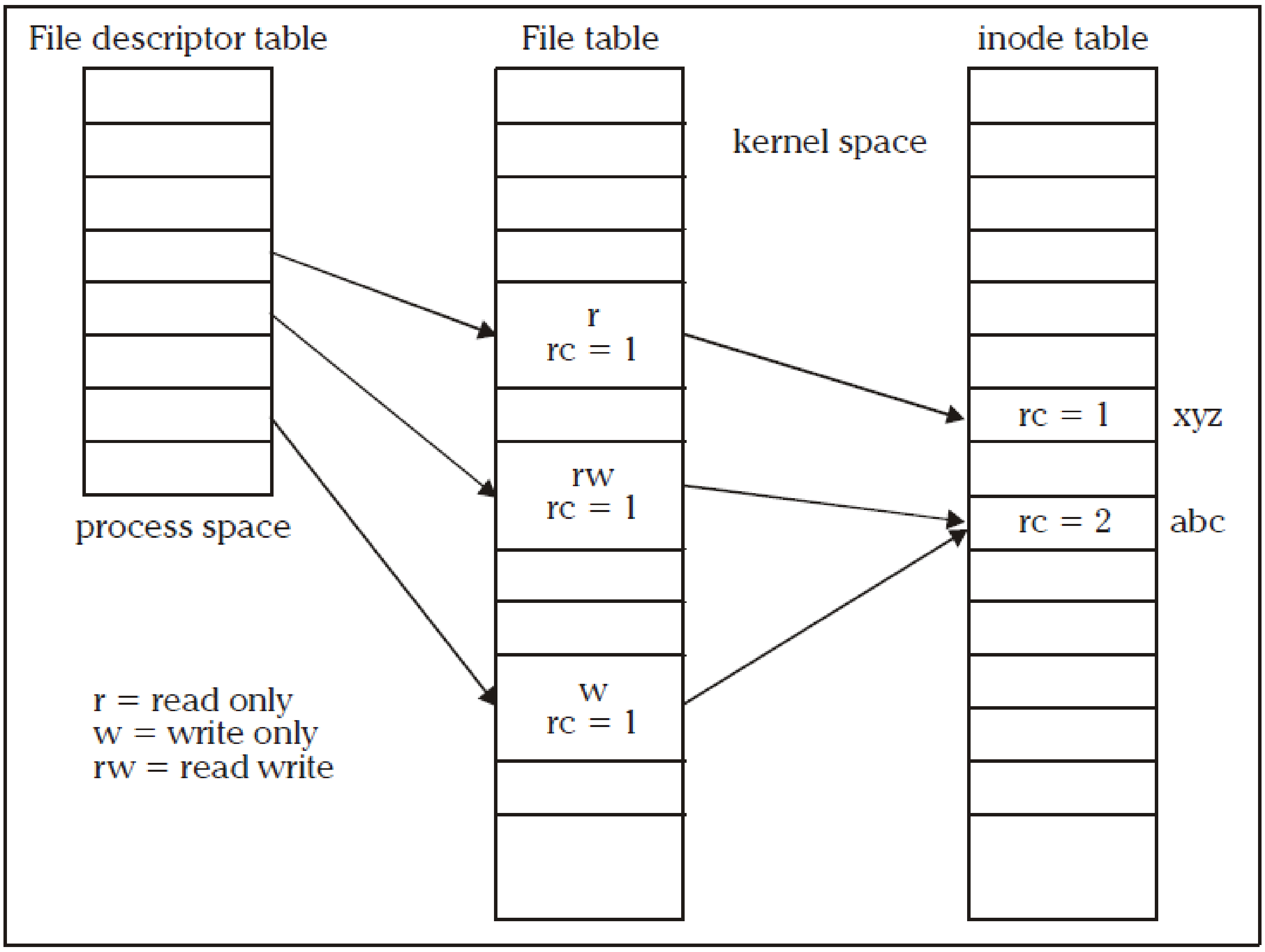
3. The file table entry is marked as unused.

4. The rc in the corresponding file inode table entry is decremented by 1, if rc value not equal to 0 go to step 6.

5. If the hard link count of the inode is not zero, it returns to the caller with a success status otherwise it marks the inode table entry as unused and de-allocates all the physical dusk storage of the file.

6. It returns to the process with a 0 (success) status.

Data Structure for File Manipulation



b) 9 commonly defined files- 3 marks(1 mark for ,listing 3 files with usage)
The following files are commonly defined in most UNIX systems

FILE	Use
/etc	Stores system administrative files and programs
/etc/passwd	Stores all user information’s
/etc/shadow	Stores user passwords
/etc/group	Stores all group information
/bin	Stores all the system programs like cat, rm, cp,etc.
/dev	Stores all character device and block device files
/usr/include	Stores all standard header files.
/usr/lib	Stores standard libraries
/tmp	Stores temporary files created by program

Q.4a) Differentiate between ANSI C and K & R C. Explain each with example 7M

ANSI C supports :
Function Prototyping -1M
Support of const and volatile datatype qualifiers-2M
Permit function pointers to be used without dereferencing-2M
Support of wide character and internationalization-2M

Write explanation with example

b) Why are the APIs more time consuming than the C library functions

User mode-1M

Kernel Mode 1

Explanation 1M

Context Switching

A *user mode* is the normal execution context of any user process, and it allows the process to access its specific data only.

A *kernel mode* is the protective execution environment that allows a user process to access kernels data in a restricted manner.

When the APIs execution completes, the user process is switched back to the user mode. This context switching for each API call ensures that process access kernels data in a controlled manner and minimizes any chance of a runway user application may damage an entire system. So in general calling an APIs is more time consuming than calling a user function due to the context switching. Thus for those time critical applications, user should call their system APIs only if it is necessary.

Q. 5 a) Explain the following APIs with their prototypes

i) open( ) ii) lseek( ) ii) access() v) utime( )

Each 2.5 M -1M Prototype + 1.5M explanation

i)Open( )

Open:

It is used to open or create a file by establishing a connection between the calling process and a file.

Prototype:

```
#include < sys/types.h>
#include <unistd.h>
#include <fcntl.h>
int open(const char *path_name, int access_mode, mode_t permission);
```

path_name : The pathname of a file to be opened or created.

Access mode flags:

1) **O_RDONLY:**

2) **O_WRONLY:**

3) **O_RDWR:**

Permission:

The permission argument is required only if the O_CREAT flag is set in the access_mode argument. It specifies the access permission of the file for its owner, group and all the other people.

ii) lseek()

The lseek system call can be used to change the file offset to a different value. It allows a process to perform random access of data on any opened file. Lseek is incompatible with FIFO files, characted device files and symbolic link files.

Its prototype is:

```
#include <sys/types.h>
#include <unistd.h>
off_t lseek (int fdesc , off_t pos, int whence);
```

- **fdesc:** is an integer file descriptor that refers to an opened file.
- **pos:** specifies a byte offset to be added to a reference location in deriving the new file offset value.
- **whence:** specifies the reference location.

Whence value	Reference location
SEEK_CUR	current file pointer address
SEEK_SET	The beginning of a file
SEEK_END	The end of a file

iii) access( )

The access function checks the existence and/or access permission of user to a named file. The prototype is given below:

```
#include <unistd.h>
int access (const char* path_name, int flag);
```

path_name: The pathname of a file.

flag: contains one or more of the following bit-flags.

Bit Flag	Use
F_OK	Checks whether a named file exists.
R_OK	Checks whether a calling process has read permission
W_OK	Checks whether a calling process has write permission
X_OK	Checks whether a calling process has execute permission

iv) utime ( )

```
#include <sys/types.h>

#include <utime.h>

int utime(const char *filename, const struct utimbuf *buf);
```

utime() changes the access and modification times of the inode specified by filename to the actime and modtime fields of buf respectively.

If buf is NULL, then the access and modification times of the file are set to the current time.

The utimbuf structure is:

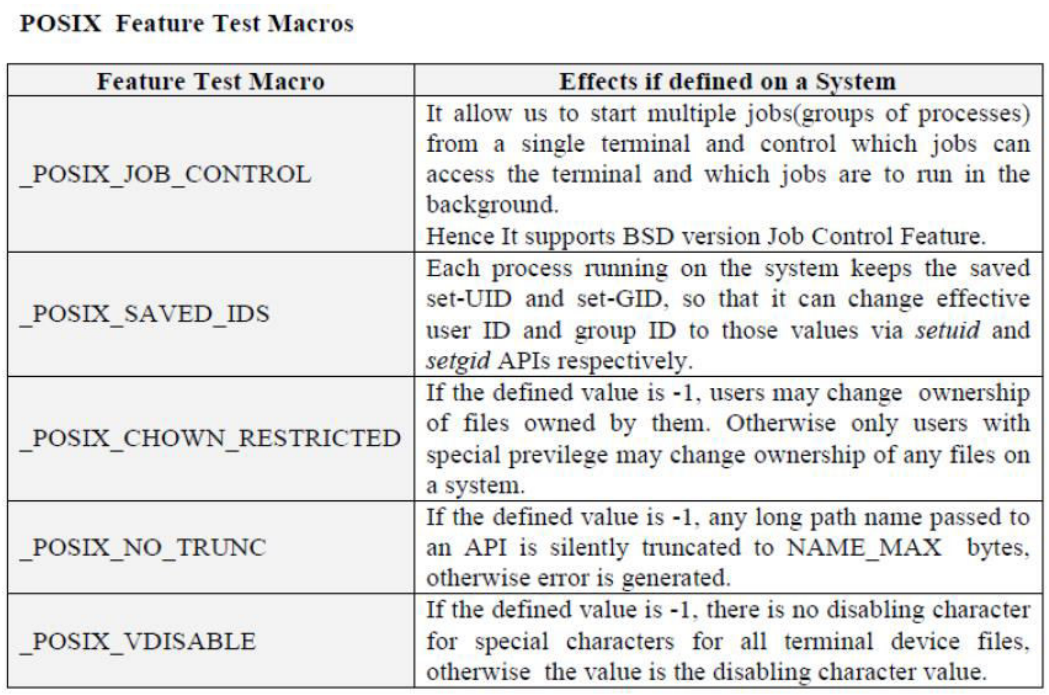
```
struct utimbuf {

    time_t actime;       /* access time */

    time_t modtime;      /* modification time */

};
```

Q.6 a) What do you mean by the term feature test macros? List all the test macros along with their meaning.

Test macro explanation-1M

Feature test macros allow the programmer to control the definitions that are exposed by system header files when a program is compiled.

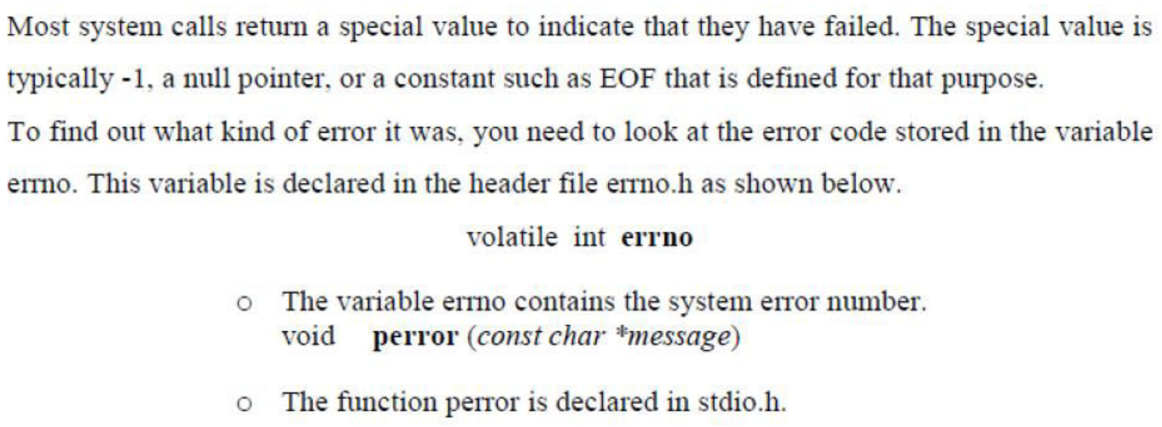
Each test macro 1M*5=5M


POSIX Feature Test Macros

Feature Test Macro	Effects if defined on a System
_POSIX_JOB_CONTROL	It allow us to start multiple jobs(groups of processes) from a single terminal and control which jobs can access the terminal and which jobs are to run in the background. Hence It supports BSD version Job Control Feature.
_POSIX_SAVED_IDS	Each process running on the system keeps the saved set-UID and set-GID, so that it can change effective user ID and group ID to those values via *setuid* and *setgid* APIs respectively.
_POSIX_CHOWN_RESTRICTED	If the defined value is -1, users may change ownership of files owned by them. Otherwise only users with special previlege may change ownership of any files on a system.
_POSIX_NO_TRUNC	If the defined value is -1, any long path name passed to an API is silently truncated to NAME_MAX bytes, otherwise error is generated.
_POSIX_VDISABLE	If the defined value is -1, there is no disabling character for special characters for all terminal device files, otherwise the value is the disabling character value.

Q. 6 b) Discuss the common characteristics of API along with their error status code (any four)

Common characteristics of API-1M


Most system calls return a special value to indicate that they have failed. The special value is typically -1, a null pointer, or a constant such as EOF that is defined for that purpose.

To find out what kind of error it was, you need to look at the error code stored in the variable errno. This variable is declared in the header file errno.h as shown below.

volatile int **errno**

- The variable errno contains the system error number.
  void **perror** (*const char *message*)

- The function perror is declared in stdio.h.

Any four error codes= 2*0.5M=2M

Following table shows Some Error Codes and their meaning:

Errors	Meaning
EPERM	API was aborted because the calling process does not have the super user privilege.
EINTR	An APIs execution was aborted due to signal interruption.
EIO	An Input/Output error occurred in an APIs execution.
ENOEXEC	A process could not execute program via one of the Exec API.
EBADF	An API was called with an invalid file descriptor.
ECHILD	A process does not have any child process which it can wait on.
EAGAIN	An API was aborted because some system resource it is requested was temporarily unavailable. The API should call again later.
ENOMEM	An API was aborted because it could not allocate dynamic memory.
EACCESS	The process does not have enough privilege to perform the operation.
EFAULT	A pointer points to an invalid address.
EPIPE	An API attempted to write data to a pipe which has no reader.
ENOENT	An invalid file name was specified to an API.

Q.7 a) Mention the different file types in Unix /POSIX System. Also explain how to create these files-6M

Listing all files-1M

Each file description 5*1M=5M

The different type's files available in UNIX / POSIX are:

- Regular files Example: All .exe files, C, C++, PDF Document files.
- Directory files Example: Folders in Windows.
- Device files
  - Block Device files: A physical device that transmits block of data at a time.
    For example: floppy devices CDROMs, hard disks.
  - Character Device files: A physical device that transmits data in a character based manner.
    For example: Line printers, modems etc.
- FIFO files Example: PIPEs.
- Link Files

b) List out any four POSIX.1b defined system configuration limits along with their meaning and minimum value - 4M

Any four limits-4*1M=4M

Meaning	Min. value	Meaning
_POSIX_AIO_MAX	1	No. of simultaneous asynchronous I/O
_POSIX_TIMER_MAX	32	Max no of timers that can be used by a process
_POSIX_MQ_OPEN_MAX	2	Max no of message queues per process
_POSIX_SEM_VALUE_MAX	32767	Max value that may be assigned to a semaphore