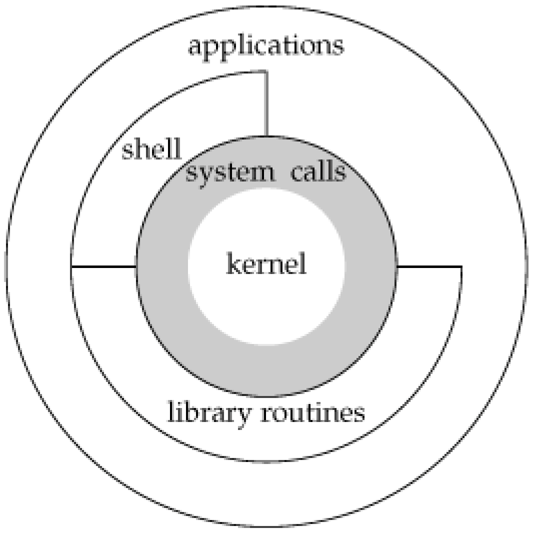
Q.2a. Draw the architecture of UNIX operating system. Explain the functionality of each component.

An operating system can be defined as the software that controls the hardware resources of the Computer and provides an environment under which programs can run. Generally, we call this software the kernel, since it is relatively small and resides at the core of the environment.

Architecture of the UNIX operating system



The interface to the kernel is a layer of software called the system calls (the shaded portion in. Libraries of common functions are built on top of the system call interface, but applications are free to use both. The shell is a special application that provides an interface for running other applications.

An operating system is the kernel and all the other software that makes a computer useful and gives the computer its personality. This other software includes system utilities, applications, shells, libraries of common functions, and so on.

b.Explain the concept of system calls and library functions.

Answer:

The functions which are a part of standard C library are known as Library functions. For example the standard string manipulation functions like strcmp(), strlen() etc are all library functions.

The functions which change the execution mode of the program from user mode to kernel mode are known as system calls. These calls are required in case some services are required by the program from kernel. For example, if we want to change the date and time of the system or if we want to create a network socket then these services can only be provided by kernel and hence these cases require system calls. For example, socket() is a system call.

Q.3a. What is the purpose of stat, fstat and lstat functions? Briefly explain each.

Answer:

The stat() function gets information about the named file and writes it to the area that buf points to. stat() updates any time-related fields before writing into the stat structure. The system must be able to search all directories leading to the file; however, read, write, or execute permission of the file is not required.

The fstat() function is identical to stat(), except that the file whose information is retrieved is specified by file descriptor rather than file name.

The lstat() function has the same effect as stat(), except when the specified path refers to a symbolic link. In that case, lstat() returns information about the link, while stat() returns information about the file the link references.

b.Explain the usage of sticky bit and symbolic links in UNIX.

Answer:

The most common use of the sticky bit today is on directories. When the sticky bit is set, only the item's owner, the directory's owner, or the super user can rename or delete files. Without the sticky bit set, any user with write and execute permissions for the directory can rename or delete contained files, regardless of owner. Typically this is set on the `/tmp` directory to prevent ordinary users from deleting or moving other users' files..

Solaris defines special behavior when the sticky bit is set on non-executable files: those files, when accessed, will not be cached by the kernel. This is usually set on swap files to prevent access on the file from flushing more important data from the system cache. It is also used occasionally for benchmarking tests.

The sticky bit is also set by the auto mounted to indicate that a file has not been mounted yet. This allows programs like ls to ignore uncounted remote files

Symbolic links operate transparently for most operations: programs that read or write to files named by a symbolic link will behave as if operating directly on the target file. However, programs that need to handle symbolic links specially (e.g., backup utilities) may identify and manipulate them directly.

A symbolic link contains a text string that is automatically interpreted and followed by the operating system as a path to another file or directory. This other file or directory is called the "target". The symbolic link is a second file that exists independently of its target. If a symbolic link is deleted, its target remains unaffected. If a symbolic link points to a target, and sometime later that target is moved, renamed or deleted, the symbolic link is not automatically updated or deleted, but continues to exist and still points to the old target, now a non-existing location or file. Symbolic links pointing to moved or non-existing targets are sometimes called broken, orphaned, dead, or dangling.

c.Explain file access permission groups and permission types.

Answer:

Each file and directory has three user based permission groups:

owner - The Owner permissions apply only the owner of the file or directory, they will not impact the actions of other users.

group - The Group permissions apply only to the group that has been assigned to the file or directory, they will not effect the actions of other users.

all users - The All Users permissions apply to all other users on the system, this is the permission group that you want to watch the most.

Permission Types

Each file or directory has three basic permission types:

read - The Read permission refers to a user's capability to read the contents of the file.

write - The Write permissions refer to a user's capability to write or modify a file or directory.

execute - The Execute permission affects a user's capability to execute a file or view the contents of a directory.

Q.4a. How would you use the fsync function with standard I/O stream?

Answer:

The fsync() function can be used by an application to indicate that all data for the open file description named by fildes is to be transferred to the storage device associated with the file described by fildes in an implementation-dependent manner. The fsync() function does not return until the system has completed that action or until an error is detected.

The fsync() function forces all currently queued I/O operations associated with the file indicated by file descriptor fildes to the synchronised I/O completion state. All I/O operations are completed as defined for synchronised I/O file integrity completion.

b. Describe three ways to position a standard I/O stream.

Answer:

A stream is associated with an external file (which may be a physical device) by ``opening" a file, which may involve ``creating" a new file. Creating an existing file causes its former contents to be discarded if necessary. If a file can support positioning requests (such as a disk file, as opposed to a terminal), then a ``file position indicator" associated with the stream is positioned at the start (byte number 0) of the file, unless the file is opened with append mode, in which case it is implementation-defined whether the file position indicator is initially positioned at the beginning or end of the file. The file position indicator is maintained by subsequent reads, writes, and positioning requests, to facilitate an orderly progression through the file. All input takes place as if bytes were read by successive calls to fgetc(); all output takes place as if bytes were written by successive calls to fputc().

When a stream is ``unbuffered", bytes are intended to appear from the source or at the destination as soon as possible; otherwise, bytes may be accumulated and transmitted as a block. When a stream is ``fully buffered", bytes are intended to be transmitted as a block when a buffer is filled. When a stream is ``line buffered", bytes are intended to be transmitted as a block when a newline byte is encountered. Furthermore, bytes are intended to be transmitted as a block when a buffer is filled, when input is requested on an unbuffered stream, or when input is requested on a line-buffered stream that requires the transmission of bytes. Support for these characteristics is implementation-defined, and may be affected via setbuf() and setvbuf().

c. Explain buffering techniques in standard I/O library.

Answer:
A buffer overflow occurs when data written to a buffer also corrupts data values in memory addresses adjacent to the destination buffer due to insufficient bounds checking. This can occur when copying data from one buffer to another without first checking that the data fits within the destination buffer.
The techniques to exploit a buffer overflow vulnerability vary per architecture, operating system and memory region. For example, exploitation on the heap (used for dynamically allocated memory), is very different from exploitation on the call stack

Q.5 a. Briefly explain the following functions: waitid, wait3, wait4.

Answer: Pages 226, 227/ reference book

b.Describe three reasons for use of interpreter files in UNIX.

Answer:

An **interpreter** is a computer program that executes, i.e. performs, instructions written in a programming language. An interpreter generally uses one of the following strategies for program execution:

1. parse the source code and perform its behavior directly

2. translate source code into some efficient intermediate representation and immediately execute this
3. explicitly execute stored precompiled code[1] made by a compiler which is part of the interpreter system

c. Explain various functions that set user IDs for the following:-
(<u>ANY TWO</u>) (5)
(i) Real user ID
(ii) Effective user ID
(iii) Write a program that prints real and effective user IDs.

Answer:
The **real ID** identify the real owner of the process and affect the permissions for sending signals. A process without super user privilege can signal another process only if the sender's real UID matches with the real UID of the receiver. Since child processes inherit the credentials from the parent, they can signal each other.

The **effective UID** (`euid`) and effective GID (`egid`) affect file creation and access. During file creation, the kernel sets the owner attributes of the file to the effective UID and effective GID of the creating process. During file access, the kernel uses the effective UID and effective GID of the process to determine if it can access the file.

The **saved user ID** (`suid`) is used when a program running with elevated privileges needs to temporarily do some unprivileged work: it changes its effective user ID from a privileged value (typically root) to some unprivileged one, and this triggers a copy of the privileged user ID to the saved user ID slot. Later, it can set its effective user ID back to the saved user ID (an unprivileged process can only set its effective user ID to three values: its real user ID, its saved user ID, and its effective user ID—i.e., unchanged) to resume its privileges. **(page 249 of ref book)**

Q.6a. Mention three ways for a process to terminate:-
(i) Normally
(ii) Abnormally (6)

Answer:

Normal termination occurs in five ways:

1. Return from main
2. Calling exit
3. Calling _exit or _Exit
4. Return of the last thread from its start routine
5. Calling pthread_exit from the last thread

Abnormal termination occurs in three ways:

6. Calling abort
7. Receipt of a signal
8. Response of the last thread to a cancellation request

b.Differentiate between :
(i) setjmp and longjmp functions
(ii)getrlimit and setrlimit functions

Answer:

(i) Both take as first argument a buffer, which is used to hold the machine state at the jump destination. When setjmp is called it populates that buffer with the current location state (which includes stack and frame pointers and the return address for the call to setjmp, and returns zero.

longjmp takes a buffer previously populated by setjmp. It also takes a (non-zero) second argument, which will ultimately be the result of the function call. longjmp restores the machine state from the buffer. It then jumps to the return address it has just restored, passing its second argument as the result. That return address is the return address from the original call to setjmp, so the effect will be as if setjmp has just returned with a non-zero argument.

(ii) Every process has a set of resource limits, some of which can be queried and changed by the geTRlimit and setrlimit functions.

These two functions are defined as XSI extensions in the Single UNIX Specification. The resource limits for a process are normally established by process 0 when the system is initialized and then inherited by each successive process. Each implementation has its own way of tuning the various limits.

Each call to these two functions specifies a single resource and a pointer to the following structure:

```
struct rlimit {
  rlim_t  rlim_cur;   /* soft limit: current limit */
  rlim_t  rlim_max;   /* hard limit: maximum value for rlim_cur */
};
```

c. Explain the features of orphaned process groups.

Answer:

When a controlling process terminates, its terminal becomes free and a new session can be established on it. (In fact, another user could log in on the terminal.) This could cause a problem if any processes from the old session are still trying to use that terminal.

To prevent problems, process groups that continue running even after the session leader has terminated are marked as *orphaned process groups*.

When a process group becomes an orphan, its processes are sent a SIGHUP signal. Ordinarily, this causes the processes to terminate. However, if a program ignores this signal or establishes a handler for it (see Signal Handling), it can continue running as in the orphan process group even after its controlling process terminates; but it still cannot access the terminal any more

Q.7 a. Explain Daemon Process? Give its characteristics.

Answer:

A daemon is a program that runs continuously and exists for the purpose of handling periodic service requests that a computer system expects to receive. The daemon program forwards the requests to other programs (or processes) as appropriate. Each server of pages on the Web has an *HTTPD* or Hypertext Transfer Protocol daemon that continually waits for requests to come in from Web clients and their users.

Characteristics

- The ps command prints the status of various processes in the system.
- $ ps –axj
- The -a option shows the status of processes owned by others
- -x shows processes that don't have a controlling terminal
- The -j option displays the job-related information: the session ID, process group ID, controlling terminal, and terminal process group ID
- Process with PID 0, 1, 2 are special processes and they remain or exist for the entire lifetime of the system. So no PPID, PGID, SID.
- The update daemon
- A program that flushes the kernel’s buffer cache at regular interval.
- The cron daemon
- Wakes up every minute and check whether any scheduled job is available for it to execute.
- If it is, it executes the job and goes back to sleep again.

b. What are the three options available to us when a signal occurs?

Answer:

- Continuous or discrete sample time
- Real- or complex-valued

- Fixed or variable size dimensions
- Floating- or fixed-point data type
- N-dimensional
- Simulink enumerations

Q.8 a. List any five Reentrant functions that may be called from a signal handler.

Ans Page 306 of ref book

b.Explain features of job control signal

Answer:

SIGCHLD	Child process has stopped or terminated.
SIGCONT	Continue process, if stopped.
SIGSTOP	Stop signal (can't be caught or ignored).
SIGTSTP	Interactive stop signal.
SIGTTIN	Read from controlling terminal by member of a background process group.
SIGTTOU	Write to controlling terminal by member of a background process group.

c.Briefly explain the canonical and non canonical modes of input processing

Answer:

In *canonical input processing* mode, terminal input is processed in lines terminated by newline (`'\n'`), EOF, or EOL characters. No input can be read until an entire line has been typed by the user, and the `read` function returns at most a single line of input, no matter how many bytes are requested.

In canonical input mode, the operating system provides input editing facilities: some characters are interpreted specially to perform editing operations within the current line of text, such as ERASE and KILL.

The constants `_POSIX_MAX_CANON` and `MAX_CANON` parameterize the maximum number of bytes which may appear in a single line of canonical input. You are guaranteed a maximum line length of at least `MAX_CANON` bytes, but the maximum might be larger, and might even dynamically change size.

In *noncanonical input processing* mode, characters are not grouped into lines, and ERASE and KILL processing is not performed. The granularity with which bytes are read in noncanonical input mode is controlled by the MIN and TIME settings.

Most programs use canonical input mode, because this gives the user a way to edit input line by line. The usual reason to use noncanonical mode is when the program accepts single-character commands or provides its own editing facilities.

The choice of canonical or noncanonical input is controlled by the `ICANON` flag in the `c_lflag` member of `struct termios`.

Q.9 a. What are the limitations of pipes? Mention the rules when one end of pipe is closed.

Answer:

In computer programming, especially in UNIX operating systems, a pipe is a technique for passing information from one program process to another. Unlike other forms of interprocess communication (IPC), a pipe is one-way communication only. Basically, a pipe passes a parameter such as the output of one process to another process which accepts it as input. The system temporarily holds the piped information until it is read by the receiving process.

Using a UNIX shell (the UNIX interactive command interface), a pipe is specified in a command line as a simple vertical bar (|) between two command sequences. The output or result of the first command sequence is used as the input to the second command sequence. The pipe system call is used in a similar way within a program.

For two-way communication between processes, two pipes can be set up, one for each direction. A limitation of pipes for interprocess communication is that the processes using pipes must have a common parent process (that is, share a common open or initiation process and exist as the result of a fork system call from a parent process).

b.Explain the following: (5+5)

(i) Message queues

(ii) Shared memory

Answer:

(i)Message queuing has been used in data processing for many years. It is most commonly used today in electronic mail. Without queuing, sending an electronic message over long distances requires every node on the route to be available for forwarding messages, and the addressees to be logged on and conscious of the fact that you are trying to send them a message. In a queuing system, messages are stored at intermediate nodes until the system is ready to forward them. At their final destination they

are stored in an electronic mailbox until the addressee is ready to read them.

Even so, many complex business transactions are processed today without queuing. In a large network, the system might be maintaining many thousands of connections in a ready-to-use state. If one part of the system suffers a problem, many parts of the system become unusable.

(ii) Shared memory is memory that may be simultaneously accessed by multiple programs with an intent to provide communication among them or avoid redundant copies. Shared memory is an efficient means of passing data between programs. Depending on context, programs may run on a single processor or on multiple separate processors.

Using memory for communication inside a single program, for example among its multiple threads, is also referred to as shared memory

A shared memory system is relatively easy to program since all processors share a single view of data and the communication between processors can be as fast as memory accesses to a same location. The issue with shared memory systems is that many CPUs need fast access to memory and will likely cache memory, which has two complications:

- CPU-to-memory connection becomes a bottleneck. Shared memory computers cannot scale very well. Most of them have ten or fewer processors.
- Cache coherence: Whenever one cache is updated with information that may be used by other processors, the change needs to be reflected to the other processors, otherwise the different processors will be working with incoherent data (see cache coherence and memory coherence). Such coherence protocols can, when they work well, provide extremely high-performance access to shared information between multiple processors. On the other hand they can sometimes become overloaded and become a bottleneck to performance.

TEXTBOOK

I. Advanced Programming in the UNIX Environment, W. Richards Stevens, Pearson Education, 2004