

RTOS* Assignment-3

*Dr Layak Ali, Assistant Professor, CUK

Y V Sai Dinesh

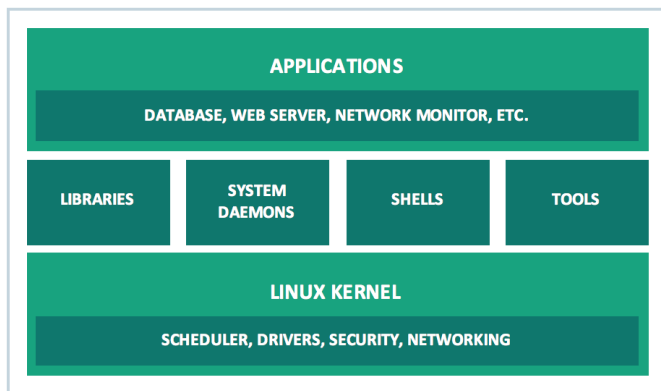
Department of Electronics & Communications Engineering

Central University of Karnataka

yvsdinesh@ieee.org

- 1) Explain Linux OS Architecture.
- 2) Explain Process life cycle and Context switching in details.
- 3) Explain Linux process creation using fork and exec with example.
- 4) Explain CPU Scheduling Criteria and implementation of Round Robin algorithm.
- 5) Explain IPC implementation using pipe on Linux.
- 6) Implement two ways IPC using FIFO on Linux.

I. ANSWER FOR Q1



A. Kernel

Kernel is the core of the Linux based operating system. It virtualizes the common hardware resources of the computer to provide each process with its virtual resources. This makes the process seem as if it is the sole process running on the machine. The kernel is also responsible for preventing and mitigating conflicts between different processes. Different types of the kernel are:

- Monolithic Kernel
- Hybrid kernels
- Exo kernels
- Micro kernels

B. System Library

Is the special types of functions that are used to implement the functionality of the operating system.

C. Shell

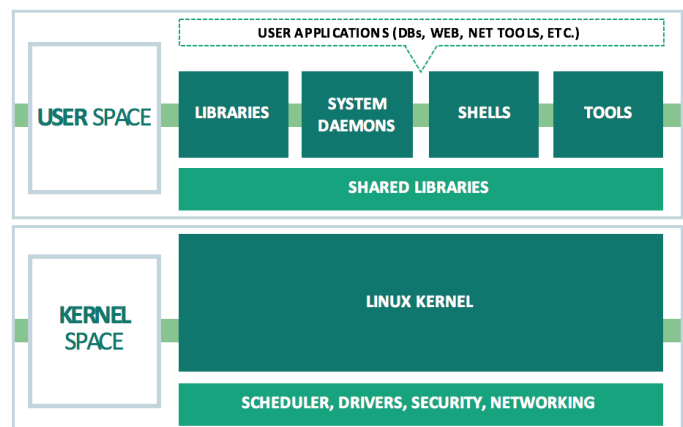
It is an interface to the kernel which hides the complexity of the kernel's functions from the users. It takes commands from the user and executes the kernel's functions.

D. Hardware Layer

This layer consists all peripheral devices like RAM/HDD/CPU etc.

E. System Utility

It provides the functionalities of an operating system to the user.



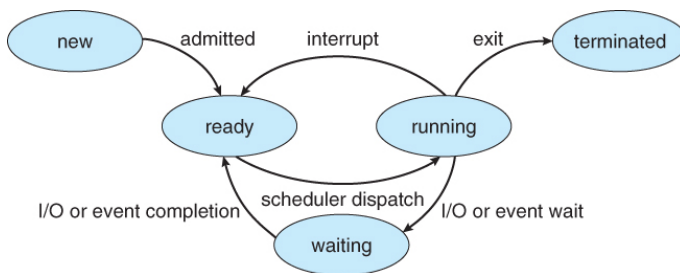
II. ANSWER FOR Q2

A process is normally more than just program code which is often called as text section. It also contains the current activity. Which is implied by the value of program counter and the contents of the processor's register.

A process changes its state during its execution which is called the process life cycle. A process life cycle consists of five stages which are:

State	Description
New	The process which is being created
Running	Instructions being executed
Waiting	The process is waiting for an event to get occur
Ready	The process is waiting to be assigned to a processor
Terminated	The process completed its execution

The process life cycle can be defined by a state diagram. Which has states representing the execution status of process at various time and transitions. That shows the changes in the execution status. To maintain the management information about a process the operating system uses the process control block (PCB).



Context Switching involves storing the context or state of a process so that it can be reloaded when required and execution can be resumed from the same point as earlier. This is a feature of a multitasking operating system and allows a single CPU to be shared by multiple processes.

There are three major triggers for context switching. These are given as follows

- **Multitasking:** In a multitasking environment, a process is switched out of the CPU so another process can be run. The state of the old process is saved and the state of the new process is loaded. On a pre-emptive system, processes may be switched out by the scheduler.
- **Interrupt Handling:** The hardware switches a part of the context when an interrupt occurs. This happens automatically. Only some of the context is changed to minimize the time required to handle the interrupt.
- **User and Kernel Mode Switching:** A context switch may take place when a transition between the user mode and kernel mode is required in the operating system.

III. ANSWER FOR Q3

Every application(program) comes into execution through means of process, process is a running instance of a program. Processes are created through different system calls, most popular are fork() and exec()

fork() creates a new process by duplicating the calling process, The new process, referred to as child, is an exact duplicate of the calling process, referred to as parent, except for the following :

- The child has its own unique process ID, and this PID does not match the ID of any existing process group.
- The child's parent process ID is the same as the parent's process ID.
- The child does not inherit its parent's memory locks and semaphore adjustments.

The exec() family of functions replaces the current process image with a new process image. It loads the program into the current process space and runs it from the entry point.

The parent would fork a child process and the latter would exec the child program in it. The parent would wait for the child to do its work and terminate.

Parent

// parent.c: the parent program

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>
```

```
int main (int argc, char **argv)
{
    int i = 0;
    long sum;
    int pid;
    int status, ret;
    char *myargs [] = { NULL };
    char *myenv [] = { NULL };

    printf ("Parent: Hello, World!\n");
```

```
pid = fork ();
if (pid == 0) {
    // I am the child

    execve ("child", myargs, myenv);
}
```

// I am the parent

```
printf("Parent:Waiting Child to complete.");
```

```

if ((ret = waitpid
(pid, &status, 0)) == -1)

printf ("parent:error");

if (ret == pid)
printf("Parent:Child process waited for.");
}

```

Child

// child.c: the child program

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

```

```

#define A 500
#define B 600
#define C 700

```

```

int main (int argc, char **argv)
{
    int i, j;
    long sum;

    // Some arbitrary work
    //done by the child

    printf ("Child: Hello, World!\n");

    for (j = 0; j < 30; j++ ) {
        for (i =0; i < 900000; i++) {
            sum = A * i + B * i * i + C;
            sum %= 543;
        }
    }

    printf ("Child: Work completed!\n");
    printf ("Child: Bye now.\n");

    exit (0);
}

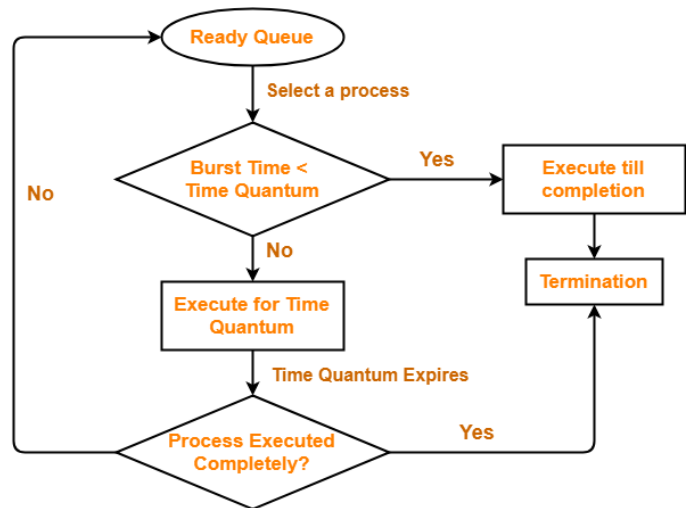
```

IV. ANSWER FOR Q4

There are many different criterias to check when considering the "best" scheduling algorithm, they are:

- **CPU Utilization**// To make out the best use of CPU and not to waste any CPU cycle, CPU would be working most of the time(Ideally 100% of the time). Considering a real system, CPU usage should range from 40% (lightly loaded) to 90% (heavily loaded.)
- **Throughput**
It is the total number of processes completed per unit time or rather say total amount of work done in a unit of time. This may range from 10/second to 1/hour depending on the specific processes.

- **Turnaround Time** It is the amount of time taken to execute a particular process, i.e. The interval from time of submission of the process to the time of completion of the process(Wall clock time).
- **Waiting Time**
The sum of the periods spent waiting in the ready queue amount of time a process has been waiting in the ready queue to acquire get control on the CPU.
- **Load Average** It is the average number of processes residing in the ready queue waiting for their turn to get into the CPU.
- **Response Time**
Amount of time it takes from when a request was submitted until the first response is produced. Remember, it is the time till the first response and not the completion of process execution(final response).



Round Robin Scheduling

Round Robin Example:

Process	Duration	Order	Arrival Time
P1	3	1	0
P2	4	2	0
P3	3	3	0

Suppose time quantum is 1 unit.

P1	P2	P3	P1	P2	P3	P1	P2	P3	P2
0									10

P1 waiting time : 4

P2 waiting time: 6

P3 waiting time: 6

The average waiting time(AWT) : $(4+6+6)/3=5.33$

V. ANSWER FOR Q5

```

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

void main()
{
    int pdfs[2];
    char buf[30];

    if (pipe(pdfs)==-1)
    {
        perror("Pipe not connected");
        exit(1);
    }

    if (!fork())
    {
        printf("CHILD: Writing to the pipe");
        write(pdfs[1], "I am child", 23);
        printf("CHILD:Exiting");
        exit(0);
    }

    else
    {
        printf("PARENT:reading from pipe");
        read(pdfs[0],buf,23);
        printf("PARENT:Received :%s",buf);
    }
}

```

A pipe is a connection between two processes, such that the standard output from one process becomes the standard input of the other process. In UNIX Operating System, Pipes are useful for communication between related processes(inter-process communication).

- Pipe is one-way communication only i.e we can use a pipe such that One process write to the pipe, and the other process reads from the pipe. It opens a pipe, which is an area of main memory that is treated as a “virtual file”.
- The pipe can be used by the creating process, as well as all its child processes, for reading and writing. One process can write to this “virtual file” or pipe and another related process can read from it.
- If a process tries to read before something is written to the pipe, the process is suspended until something is written.
- The pipe system call finds the first two available positions in the process’s open file table and allocates them for the read and write ends of the pipe.

VI. ANSWER FOR Q6

IPC using FIFO and process creation using fork() system call

```

//Write First then Read
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

int main()
{
    int fd,fd1;
    char * myfifo = "/tmp/myfifo";
    mkfifo(myfifo, 0666);
    char str1[80], str2[80];
    char arr1[80], arr2[80];

    while (1)
    {
        if(!fork())
        {
            fd1 = open(myfifo,O_RDONLY);
            read(fd1, str1, 80);
            printf("CHILD: reading from the pipe\n");
            printf("Received data: %s\n", str1);
            close(fd1);

            printf("CHILD: writing from the pipe\n");
            fd = open(myfifo, O_WRONLY);
            fgets(arr2, 80, stdin);

            write(fd, arr2, strlen(arr2)+1);
            close(fd);
        }

        else
        {
            printf("PARENT: writing from the pipe\n");
            fd1 = open(myfifo,O_WRONLY);
            fgets(str2, 80, stdin);
            write(fd1, str2, strlen(str2)+1);
            close(fd1);

            fd = open(myfifo, O_RDONLY);
            read(fd, arr1, sizeof(arr1));

            printf("PARENT: reading from the pipe\n");
            printf("Received data: %s\n", arr1);
            close(fd);
        }
    }

    return 0;
}

```

LT_EX