INSTITUTE OF ENGINEERING AND TECHNOLOGY , DAVV INDORE

DEPARTMENT OF INFORMATION TECHNOLOGY

LAB ASSIGNMENT OF OPERATING SYSTEM

SUBJECT CODE: 4ITRC2

LAB ASSIGNMENT - 04

NAME : AAYUSH SAHU

ROLLNO : 23I4101

CLASS : BE 2ND YEAR

1. Process Management System Calls

These system calls are used to create, manage, and terminate processes. They allow the kernel and user programs to handle multitasking by controlling how processes are started, executed, paused, resumed, and ended.

fork()

Definition: Creates a new process by duplicating the calling process.
Purpose: Used to create child processes.
Code:

```
#include <stdio.h>
#include <unistd.h>

int main() {
   pid_t pid = fork();
   if (pid == 0)
      printf("Child process\n");
   else
      printf("Parent process\n");
   return 0;
}
```

Output:

```
Parent process
Child process
```

exec()

Definition: Replaces the current process image with a new process image.
Purpose: Used to run a new program.
Code:

```
#include <stdio.h>
#include <unistd.h>

int main() {
   printf("Before exec\n");
   execl("/bin/ls", "ls", NULL);
   printf("This won't be printed if exec is successful.\n");
   return 0;
}
```

Output:

```
Before exec
<directory contents shown>
```

wait()

Definition: Suspends execution of the parent process until the child terminates.
Purpose: To wait for child process termination.
Code:

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

int main() {
   pid_t pid = fork();
   if (pid == 0) {
      printf("Child process\n");
   } else {
      wait(NULL);
      printf("Parent waited for child\n");
   }
   return 0;
}
```

Output:

```
Child process
Parent waited for child
```

exit()

Definition: Terminates the calling process.
Purpose: Cleanly exit a process.
Code:

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

int main() {
   printf("Exiting process\n");
   exit(0);
}
```

Output:

```
Exiting process
```

---

2. File Management System Calls

File management system calls are responsible for operations on files such as creating, opening, reading, writing, and closing them. These calls provide the interface between user applications and the file system.

open()

Definition: Opens a file and returns a file descriptor.
Purpose: To access files.

Code:

```
#include <fcntl.h>
#include <stdio.h>

int main() {
   int fd = open("test.txt", O_CREAT | O_WRONLY, 0644);
   if (fd >= 0)
      printf("File opened successfully\n");
   return 0;
}
```

Output:

File opened successfully

read()

Definition: Reads data from a file descriptor.

Purpose: To read contents from a file.

Code:

```
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>

int main() {
   char buf[100];
   int fd = open("test.txt", O_RDONLY);
   int n = read(fd, buf, 100);
   buf[n] = '\0';
   printf("Read: %s\n", buf);
   close(fd);
   return 0;
}
```

Output:

Read: Hello world

write()

Definition: Writes data to a file descriptor.

Purpose: To write contents to a file.

Code Example:

```
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>

int main() {
   int fd = open("test.txt", O_WRONLY | O_CREAT, 0644);
   write(fd, "Hello World", 11);
```

```
    close(fd);
    return 0;
}
```

Output:

(Contents written to "test.txt")

close()

Definition: Closes a file descriptor.

Purpose: Releases resources.

Code Example:

```
#include <fcntl.h>
#include <unistd.h>

int main() {
    int fd = open("test.txt", O_RDONLY);
    close(fd);
    return 0;
}
```

Output:

(Nothing visible, file closed successfully)

---

3. Device Management System Calls

Device management system calls are used to manage and communicate with hardware devices like keyboards, printers, and disks. These calls allow programs to read from or write to devices and modify device parameters.

read() & write()

Definition: Used to transfer data to/from devices (also used for files).

Purpose: Communicate with device drivers or files.

Code:

```
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>

int main() {
    int fd = open("test.txt", O_RDWR);
    write(fd, "Device data", 11);
    char buf[100];
    int n = read(fd, buf, 100);
```

```
    buf[n] = '\0';
    printf("Read: %s\n", buf);
    close(fd);
    return 0;
}
```

Output:

Read: Device data

ioctl()

Definition: Performs device-specific input/output operations.

Purpose: Used to configure devices.

Code (conceptual example):

```
#include <sys/ioctl.h>
#include <stdio.h>

int main() {
    int device = open("/dev/mydevice", O_RDONLY);
    ioctl(device, 0x1234, NULL);
    close(device);
    return 0;
}
```

Output:

Custom ioctl command sent (depends on driver implementation).

select()

Definition: Monitors multiple file descriptors.

Purpose: Waits for data on multiple I/O sources.

Code:

```
#include <sys/select.h>
#include <stdio.h>

int main() {
    fd_set rfds;
    struct timeval tv;
    FD_ZERO(&rfds);
    FD_SET(0, &rfds);
    tv.tv_sec = 5;
    tv.tv_usec = 0;
    int retval = select(1, &rfds, NULL, NULL, &tv);
    if (retval)
        printf("Data is available\n");
    else
        printf("No data within five seconds\n");
    return 0;
}
```

Output:
(Data availability based on input)

---

4. Network Management System Calls

These calls provide access to networking features. They enable programs to communicate over a network using sockets. Operations include creating sockets, establishing connections, and sending/receiving data.

socket()

Definition: Creates an endpoint for communication.
Purpose: Initialize network communication.
Code:

```
#include <sys/socket.h>
#include <stdio.h>

int main() {
   int sockfd = socket(AF_INET, SOCK_STREAM, 0);
   if (sockfd >= 0)
      printf("Socket created\n");
   return 0;
}
```

Output:
Socket created

connect()

Definition: Initiates a connection on a socket.
Purpose: Connect client to server.
Code (partial):

```
#include <arpa/inet.h>
#include <sys/socket.h>

struct sockaddr_in serv_addr;
connect(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr));
```

Output:
Connected to server (if server is available)

send() & recv()

Definition: Used to transmit and receive data on a socket.
Purpose: Network data transmission.
Code:

```
send(sockfd, "Hello", strlen("Hello"), 0);
recv(sockfd, buffer, sizeof(buffer), 0);
```

Output:
Send and receive message over network

---

5. System Information Management System Calls

System information calls are used to retrieve details about the system's environment and state, such as the process ID, user ID, system uptime, hostname, and available resources.

getpid()

Definition: Returns the process ID of the calling process.
Purpose: Identify processes.
Code:

```
#include <stdio.h>
#include <unistd.h>

int main() {
   printf("PID: %d\n", getpid());
   return 0;
}
```

Output:
PID: <number>

getuid()

Definition: Returns the user ID of the calling process.
Purpose: Security checks.
Code:

```
#include <stdio.h>
#include <unistd.h>

int main() {
   printf("UID: %d\n", getuid());
   return 0;
}
```

Output:
UID: <number>

gethostname()

Definition: Gets the name of the current host.
Purpose: Identify machine in a network.
Code:

```
#include <unistd.h>
#include <stdio.h>
```

```
int main() {
   char hostname[1024];
   gethostname(hostname, 1024);
   printf("Hostname: %s\n", hostname);
   return 0;
}
```

Output:

Hostname: <machine_name>

sysinfo()

Definition: Returns information on overall system statistics.

Purpose: System monitoring.

Code:

```
#include <sys/sysinfo.h>
#include <stdio.h>

int main() {
   struct sysinfo info;
   sysinfo(&info);
   printf("Uptime: %ld seconds\n", info.uptime);
   return 0;
}
```

Output:

Uptime: <seconds>
