

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	[@KLWKS_BOT THANOS]

Experiment Title: Process API

Aim/Objective:

The objective of Process API is to provide a set of functions and tools that allow programmers to manage and control processes within the operating system environment.

Description:

The Process API typically includes functions for creating new processes, terminating existing processes, querying information about processes, managing process attributes (such as process ID, parent process ID, and process state), and controlling process execution.

Pre-Requisites:

6. Analysing the concept of fork()
7. Use of the wait system calls for parent and child processes.
8. Retrieving the PID for the parent and the child.
9. Concepts of dup(), dup2().
10. Understanding various types of exec calls.
11. The init process.

Pre-Lab:

1. Write brief description and prototypes in the space given below for the following process subsystem call EX: -"\$man <system call name>"

1. fork()

Description: Creates a child process by duplicating the parent process.

Prototype: `pid_t fork(void);`

Course Title	OPERATING SYSTEMS	ACADEMIC YEAR: 2024-25
Course Code(s)	23CS2104A	Page 28 of 226

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	[@KLWKS_BOT THANOS]

2. getpid (), getppid () system calls

Description:

- `getpid()` returns process ID.
- `getppid()` returns parent process ID.

Prototypes:

```
pid_t getpid(void);
```

```
pid_t getppid(void);
```

3. exit() system call

Description: Terminates the process and returns status to the parent.

Prototype: `void exit(int status);`

Course Title	OPERATING SYSTEMS	ACADEMIC YEAR: 2024-25
Course Code(s)	23CS2104A	Page 29 of 226

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	[@KLWKS_BOT THANOS]

4. shmget()

Description: Allocates shared memory.

Prototype: `int shmget(key_t key, size_t size, int shmflg);`

5. wait()

Description: Parent waits for child process to terminate.

Prototype: `pid_t wait(int *status);`

6. sleep()

Description: Pauses process execution for a specified time.

Prototype: `unsigned int sleep(unsigned int seconds);`

Course Title	OPERATING SYSTEMS	ACADEMIC YEAR: 2024-25
Course Code(s)	23CS2104A	Page 30 of 226

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	[@KLWKS_BOT THANOS]

7. exec()

Description: Replaces current process with a new one.

Prototype: `int exec(const char *path, char *const argv[]);`

8. waitpid()

Description: Parent waits for specific child to terminate.

Prototype: `pid_t waitpid(pid_t pid, int *status, int options);`

9. _exit()

Description: Terminates process without cleanup.

Prototype: `void _exit(int status);`

Course Title	OPERATING SYSTEMS	ACADEMIC YEAR: 2024-25
Course Code(s)	23CS2104A	Page 31 of 226

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	[@KLWKS_BOT THANOS]

10. opendir()

Description: Opens a directory stream.

Prototype: `DIR *opendir(const char *name);`

11. Readdir()

Description: Reads a directory entry.

Prototype: `struct dirent *readdir(DIR *dirp);`

Course Title	OPERATING SYSTEMS	ACADEMIC YEAR: 2024-25
Course Code(s)	23CS2104A	Page 32 of 226

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	[@KLWKS_BOT THANOS]

12. execlp(),execvp(),exec(),exec()execv() system calls

Description: Replaces current process image with a new one.

Prototypes:

```
int execlp(const char *file,
const char *arg, ...);
int execvp(const char *file,
char *const argv[]);
int exec(const char *path, char
*const argv[]);
int execv(const char *path, char
*const argv[]);
```

Course Title	OPERATING SYSTEMS	ACADEMIC YEAR: 2024-25
Course Code(s)	23CS2104A	Page 33 of 226

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	[@KLWKS_BOT THANOS]

In Lab:

1. write a program for implementing process management using the following system calls of the UNIX operating system: fork, exec, getpid, exit, wait, close.

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

int main() {
    pid_t child_pid;
    int status;

    child_pid = fork();
    if (child_pid < 0) {
        perror("Fork failed");
        return 1;
    }
    if (child_pid == 0) {
        printf("Child process: My PID is %d\n", getpid());
        execl("/bin/ls", "ls", "-l", (char *)NULL);
        perror("Exec failed");
        return 1;
    } else {
        printf("Parent process: My PID is %d, Child PID is %d\n", getpid(), child_pid);
        wait(&status);
        printf("Parent process: Child process exited with status %d\n", status);
    }
    return 0;
}
```

Course Title	OPERATING SYSTEMS	ACADEMIC YEAR: 2024-25
Course Code(s)	23CS2104A	Page 34 of 226

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	[@KLWKS_BOT THANOS]

1. To write a program for implementing Directory management using the following system calls of the UNIX operating system: opendir, readdir.

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

```
int main() {
    DIR *dir;
    struct dirent *entry;

    dir = opendir(".");
    if (dir == NULL) {
        perror("opendir");
        return 1;
    }

    printf("Contents of the current directory:\n");
    while ((entry = readdir(dir)) != NULL) {
        printf("%s\n", entry->d_name);
    }

    closedir(dir);
    return 0;
}
```

Course Title	OPERATING SYSTEMS	ACADEMIC YEAR: 2024-25
Course Code(s)	23CS2104A	Page 35 of 226

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	[@KLWKS_BOT THANOS]

2 Write a program for implementing process management using the following system calls of the UNIXoperating system: fork, exec, getpid, exit, wait, close.

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

int main() {
    pid_t child_pid;
    int status;

    child_pid = fork();
    if (child_pid < 0) {
        perror("Fork failed");
        return 1;
    }
    if (child_pid == 0) {
        printf("Child process: My PID is %d\n", getpid());
        execl("/bin/ls", "ls", "-l", (char *)NULL);
        perror("Exec failed");
        return 1;
    } else {
        printf("Parent process: My PID is %d, Child PID is %d\n", getpid(), child_pid);
        wait(&status);
        printf("Parent process: Child process exited with status %d\n", status);
    }

    return 0;
}
```

Course Title	OPERATING SYSTEMS	ACADEMIC YEAR: 2024-25
Course Code(s)	23CS2104A	Page 36 of 226

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	[@KLWKS_BOT THANOS]

3 T-series creates a text document (song.txt) that contains the lyrics of a song. They want to know how many lines and words are present in the song.txt. They want to utilize Linux directions and system calls to accomplish their objective. Help T-T-series to finish their task by utilizing a fork system call. Print the number of lines in song.txt using the parent process and print the number of words in it using the child process.

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

int main() {
    pid_t child_pid;
    int status;

    child_pid = fork();
    if (child_pid < 0) {
        perror("Fork failed");
        return 1;
    }
    if (child_pid == 0) {
        FILE *file = fopen("song.txt", "r");
        if (file == NULL) {
            perror("Failed to open song.txt");
            exit(1);
        }
        int wordCount = 0;
        char ch;
        int inWord = 0;
        while ((ch = fgetc(file)) != EOF) {
            if (ch == ' ' || ch == '\n' || ch == '\t') {
                inWord = 0;
            } else if (!inWord) {
                wordCount++;
                inWord = 1;
            }
        }
    }
}
```

Course Title	OPERATING SYSTEMS	ACADEMIC YEAR: 2024-25
Course Code(s)	23CS2104A	Page 37 of 226

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	[@KLWKS_BOT THANOS]

```

printf("Child process: Number of words in song.txt: %d\n", wordCount);
fclose(file);
} else {
    wait(&status);
    if (WIFEXITED(status)) {
        printf("Parent process: Child process exited with status %d\n", WEXITSTATUS(status));
    } else {
        printf("Parent process: Child process did not exit normally\n");
    }
    FILE *file = fopen("song.txt", "r");
    if (file == NULL) {
        perror("Failed to open song.txt");
        exit(1);
    }
    int lineCount = 0;
    char ch;
    while ((ch = fgetc(file)) != EOF) {
        if (ch == '\n') {
            lineCount++;
        }
    }
    printf("Parent process: Number of lines in song.txt: %d\n", lineCount);
    fclose(file);
}
return 0;
}

```

Course Title	OPERATING SYSTEMS	ACADEMIC YEAR: 2024-25
Course Code(s)	23CS2104A	Page 38 of 226

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	[@KLWKS_BOT THANOS]

POST LAB

1. Write a program to display the user ID, group ID, parent ID, and process id.

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

int main() {
    uid_t uid = getuid();
    gid_t gid = getgid();
    pid_t ppid = getppid();
    pid_t pid = getpid();

    printf("User ID (UID): %d\n", uid);
    printf("Group ID (GID): %d\n", gid);
    printf("Parent Process ID (PPID): %d\n", ppid);
    printf("Process ID (PID): %d\n", pid);

    return 0;
}
```

Course Title	OPERATING SYSTEMS	ACADEMIC YEAR: 2024-25
Course Code(s)	23CS2104A	Page 39 of 226

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	[@KLWKS_BOT THANOS]

2. Write a program to display process statements before and after forking

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

int main() {
    printf("Before forking: This is the parent process (PID: %d)\n", getpid());
    pid_t child_pid = fork();

    if (child_pid < 0) {
        perror("Fork failed");
        return 1;
    }

    if (child_pid == 0) {
        printf("In the child process (PID: %d), after forking\n", getpid());
    } else {
        printf("In the parent process (PID: %d), after forking child process (Child PID: %d)\n",
            getpid(), child_pid);
    }

    return 0;
}
```

Course Title	OPERATING SYSTEMS	ACADEMIC YEAR: 2024-25
Course Code(s)	23CS2104A	Page 40 of 226

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	[@KLWKS_BOT THANOS]

3. Write a program to display the child process ID, parent process ID, process ID before and after forking.

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

int main() {
    printf("Before forking: This is the process (PID: %d) with parent (PPID: %d)\n", getpid(),
getppid());
    pid_t child_pid = fork();

    if (child_pid < 0) {
        perror("Fork failed");
        return 1;
    }

    if (child_pid == 0) {
        printf("In the child process (PID: %d) with parent (PPID: %d) after forking\n", getpid(),
getppid());
    } else {
        printf("In the parent process (PID: %d) with child (Child PID: %d) after forking\n", getpid(),
child_pid);
    }

    return 0;
}
```

Course Title	OPERATING SYSTEMS	ACADEMIC YEAR: 2024-25
Course Code(s)	23CS2104A	Page 41 of 226

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	[@KLWKS_BOT THANOS]

4. write a program to create a child process that sleeps for 5 seconds and after 5 seconds kills the child process with process-id.

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

int main() {
    pid_t child_pid;
    child_pid = fork();

    if (child_pid < 0) {
        perror("Fork failed");
        return 1;
    }

    if (child_pid == 0) {
        printf("Child process (PID: %d) sleeping for 5 seconds...\n", getpid());
        sleep(5);
        printf("Child process (PID: %d) woke up after 5 seconds\n", getpid());
    } else {
        printf("Parent process (PID: %d) created child process (Child PID: %d)\n", getpid(),
child_pid);
        sleep(1);
        kill(child_pid, SIGKILL);
        printf("Parent process: Killed the child process (Child PID: %d)\n", child_pid);
    }

    return 0;
}
```

Course Title	OPERATING SYSTEMS	ACADEMIC YEAR: 2024-25
Course Code(s)	23CS2104A	Page 42 of 226

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	[@KLWKS_BOT THANOS]

5 Write a C program to create a process in Unix (using fork()).

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

int main() {
    pid_t child_pid;
    child_pid = fork();

    if (child_pid < 0) {
        perror("Fork failed");
        return 1;
    }

    if (child_pid == 0) {
        printf("Child process (PID: %d) is running\n", getpid());
    } else {
        printf("Parent process (PID: %d) created a child process (Child PID: %d)\n", getpid(),
child_pid);
    }

    return 0;
}
```

Course Title	OPERATING SYSTEMS	ACADEMIC YEAR: 2024-25
Course Code(s)	23CS2104A	Page 43 of 226

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	[@KLWKS_BOT THANOS]

- **Data and Results:**

Data

The program demonstrates process creation using `fork` to create a child process.

Result

Child and parent processes display their respective PIDs after forking.

- **Analysis and Inferences:**

Analysis

The code illustrates how a child process is created and how both parent and child processes behave independently.

Inferences

Forking creates independent processes with separate execution contexts.

Course Title	OPERATING SYSTEMS	ACADEMIC YEAR: 2024-25
Course Code(s)	23CS2104A	Page 44 of 226

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	[@KLWKS_BOT THANOS]

Sample VIVA-VOCE Questions (In-Lab):

1. What is the role of a process control block (PCB) in managing processes?

The PCB stores information about processes, including their state, ID, program counter, registers, and memory usage.

2. What is a process in the context of an operating system?

A process is a program in execution, including the program code, data, and execution state.

3. What are the main functions of a process API in an operating system?

The process API manages process creation, termination, scheduling, and inter-process communication.

Course Title	OPERATING SYSTEMS	ACADEMIC YEAR: 2024-25
Course Code(s)	23CS2104A	Page 45 of 226

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	[@KLWKS_BOT THANOS]

4. Explain the concept of process termination and the role of the `exit()` system call.

Process termination happens when a process finishes execution. `exit()` ends the process and returns status to the OS.

5. What is the purpose of the `fork()` system call in the process API?

`fork()` creates a new child process by duplicating the parent process.

Evaluator Remark (if any):	Marks Secured ____ out of 50
	Signature of the Evaluator with Date

Note: Evaluator MUST ask Viva-voce before signing and posting marks for each experiment.

Course Title	OPERATING SYSTEMS	ACADEMIC YEAR: 2024-25
Course Code(s)	23CS2104A	Page 46 of 226