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.

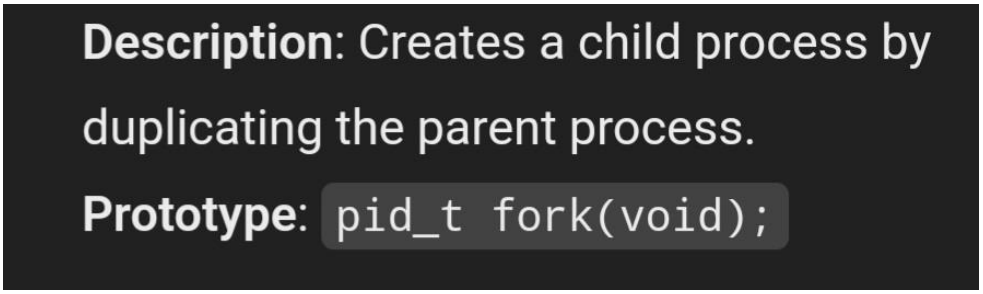
Pre-Requisites:

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.

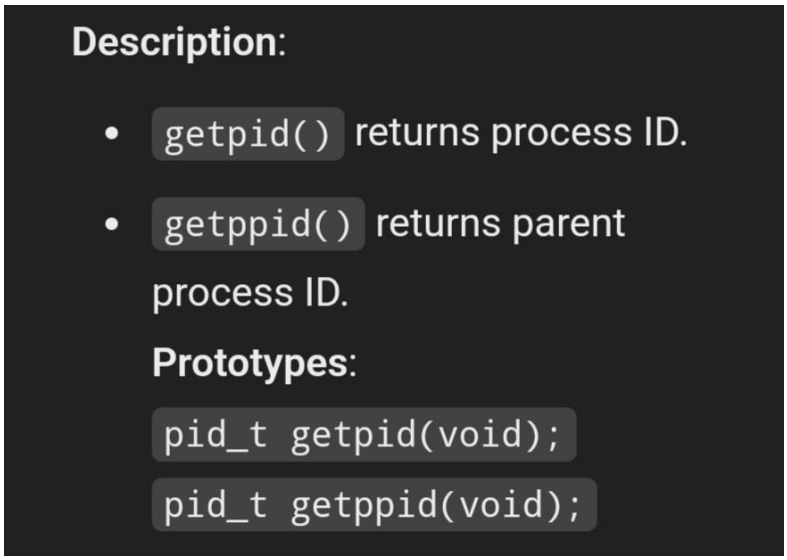
Pre-Lab:

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

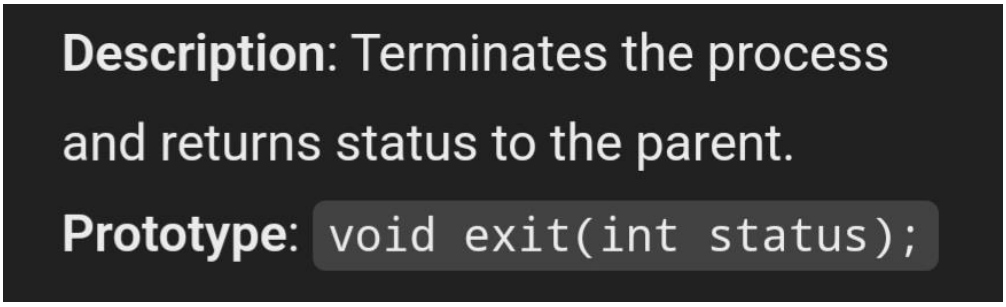
1. fork()



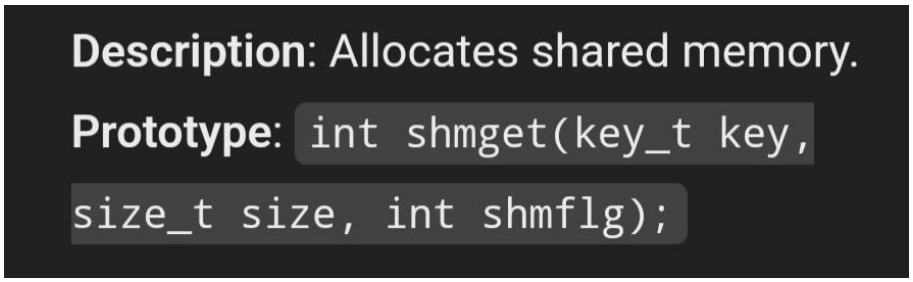
2. getpid (), getppid () system calls



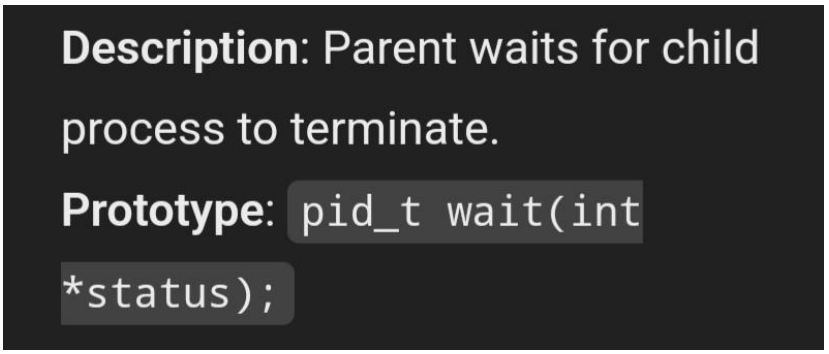
3. exit() system call



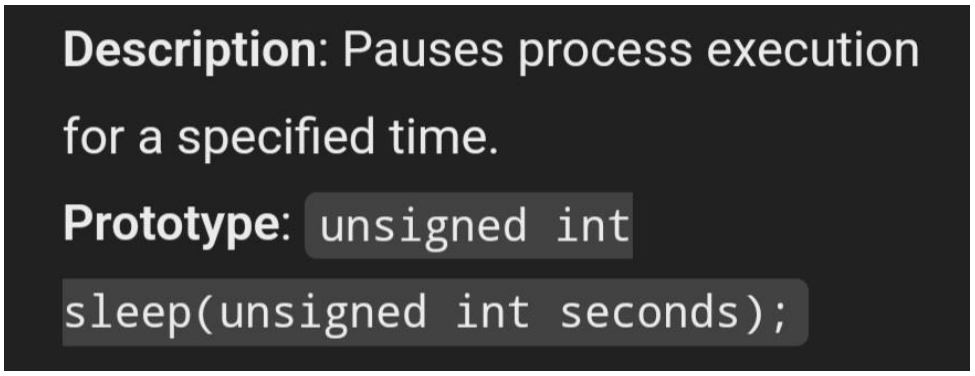
4. shmget()



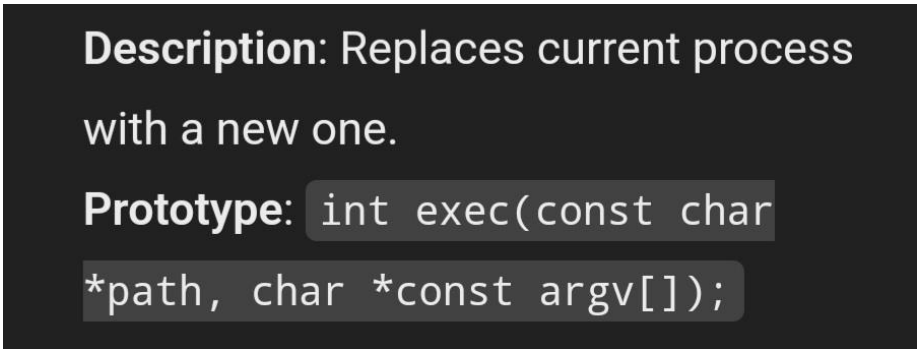
5. wait()



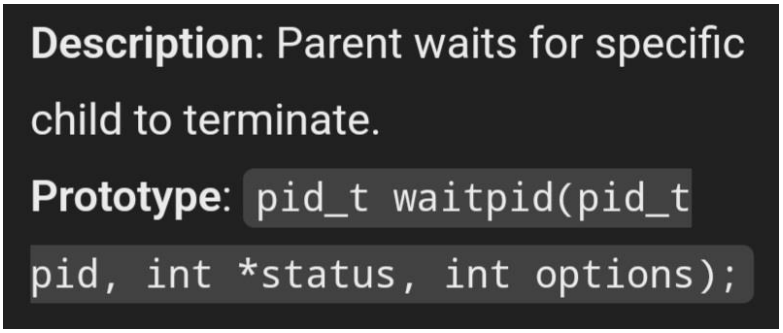
6. sleep()



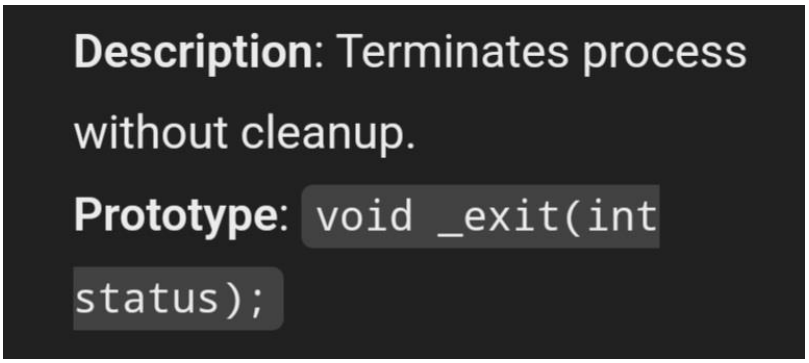
7. exec()



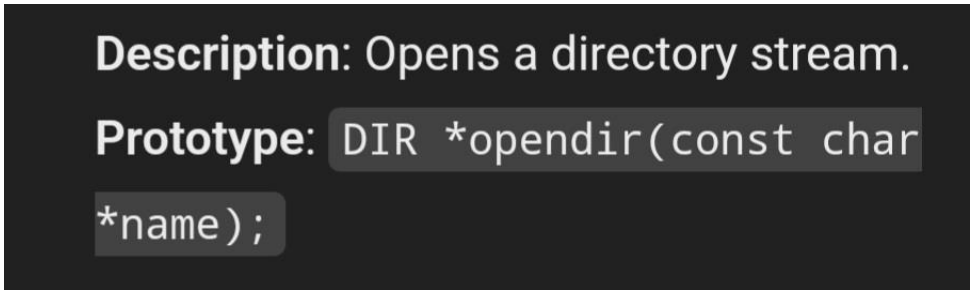
8. waitpid()



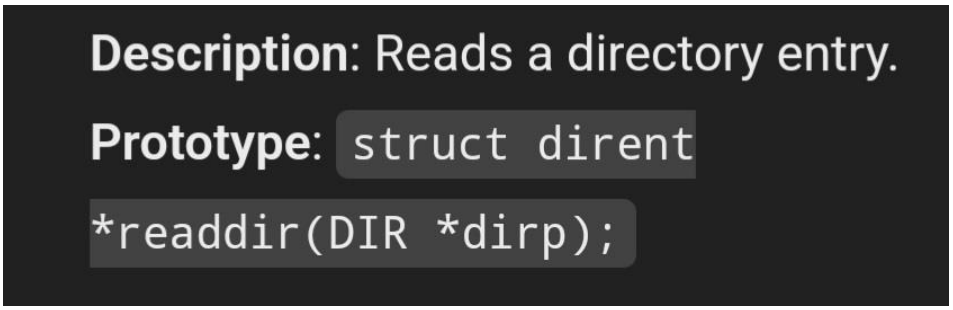
9. _exit()



10. opendir()



11. Readdir()



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

In Lab:

1. write a program for implementing process management using the following system calls of theUNIX 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;
}
```

1. To write a program for implementing Directory management using the following system calls of the UNIXoperating 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;
}
```

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;
}
```

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;
         }
      }
```

```
    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;
}
```

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;
}
```

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;
}
```

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;
}
```

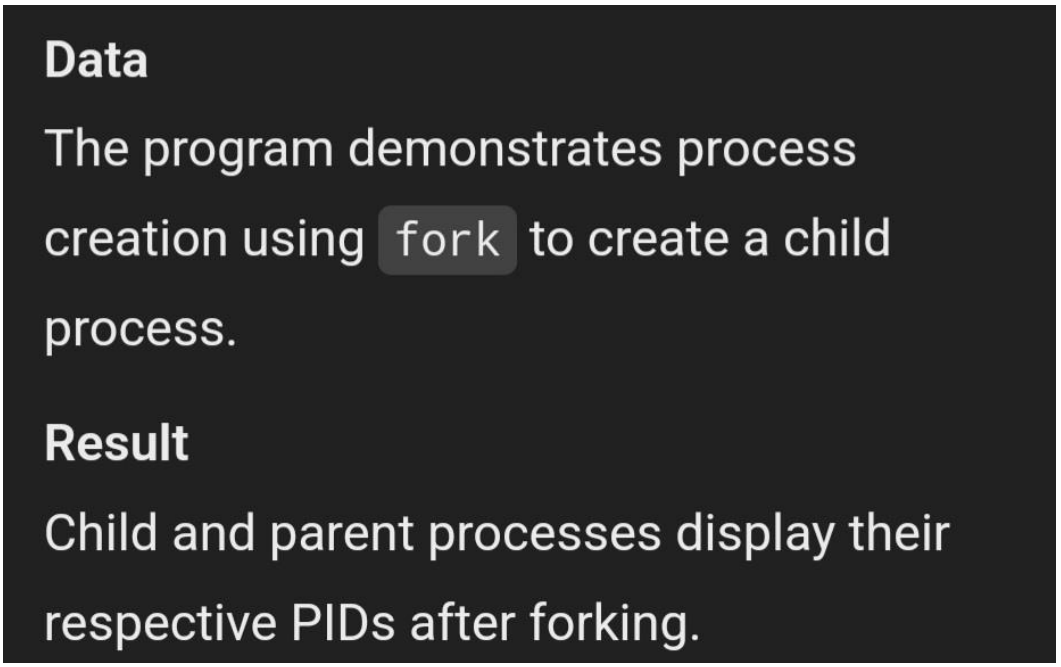
4. write a program to create a child process that sleeps for 5 seconds and after 5 seconds kills thechild process with process-id.

```
#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;
}
```

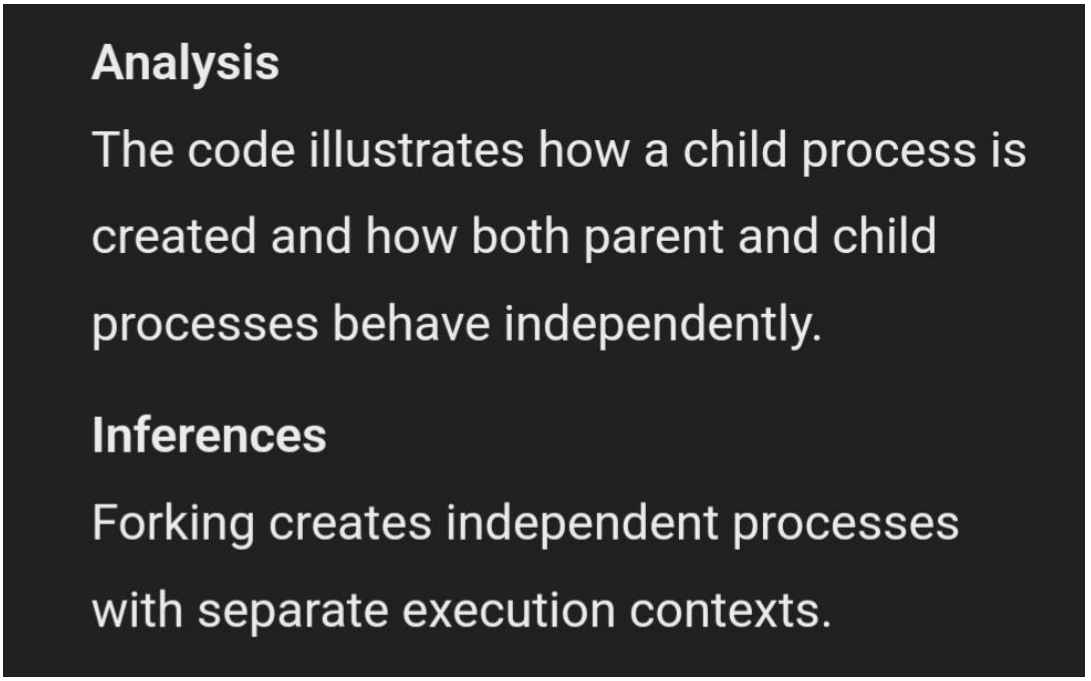
5 Write a C program to create a process in Unix (using fork()).

```
#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;
}
```

- **Data and Results:**

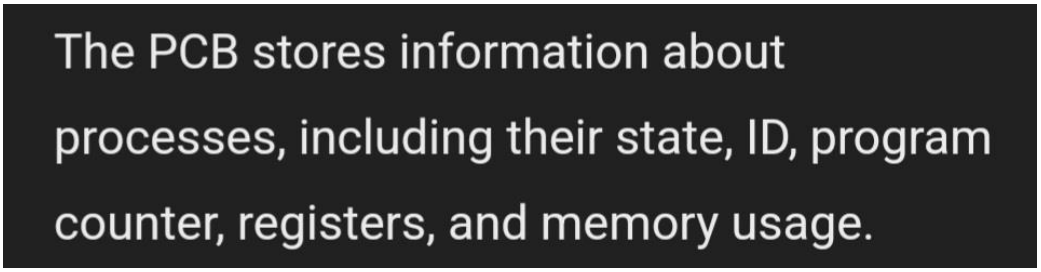


- **Analysis and Inferences:**

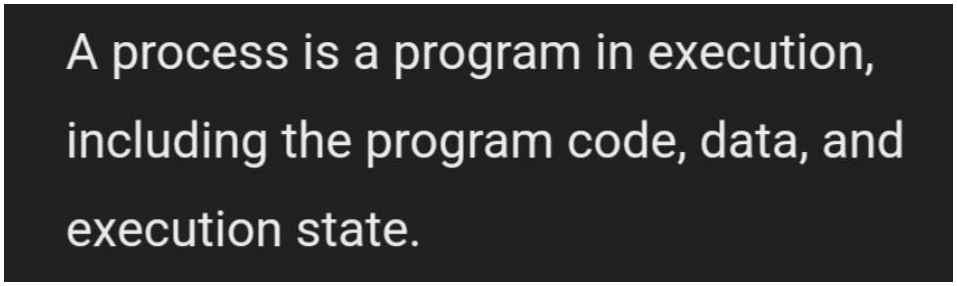


Sample VIVA-VOCE Questions (In-Lab):

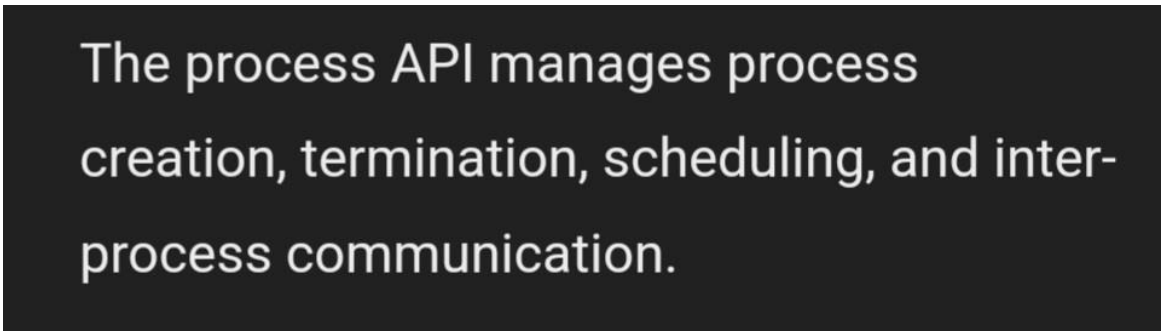
1. **What is the role of a process control block (PCB) in managing processes?**



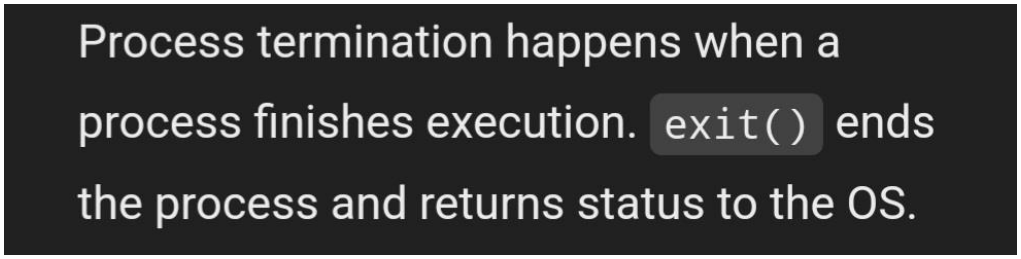
2. **What is a process in the context of an operating system?**



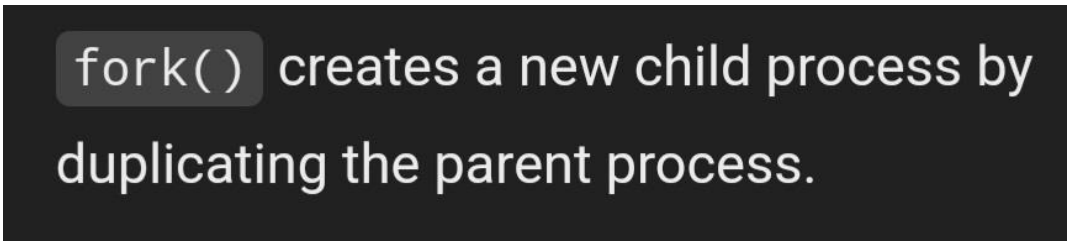
3. **What are the main functions of a process API in an operating system?**



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



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



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.