Operations on Processes

Code and Output Documentation:

1. `fork1:`

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

int main() {
        fork();
        printf("Hello \n");
        return 0;
}
```

. Compile and Run `fork1.c`:

```
gcc -o fork2 fork1.c

./fork2
```

Output:
Hello
Hello

2. `Getpid`:

```
#include<stdio.h>
#include <unistd.h>
#include<sys/types.h>
int main()
{
printf("I am in hi.c \n ");
printf("pid of hi.c is %d \n" , getpid());
return 0;
}
```

Output:

```
$ gcc -o hihi hi.c
```

$ ./hihi

I am in hi.c
pid of hi.c is 6643

Getpid.c code

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

int main() {
        pid_t pid = fork();  // Create a new process

        if (pid < 0) {
        // Fork failed
        perror("fork");
        exit(1);
        } else if (pid == 0) {
        // Child process
        printf("Hello, I am the child process.\n");
        printf("My PID is %d\n", getpid());
        exit(0);
        } else {
        // Parent process
        printf("Hello, I am the parent process.\n");
        printf("My PID is %d\n", getpid());
        exit(0);
        }
}
```

- **Compile and Run `getpid.c`:**

```
gcc -o getpid getpid.c

./getpid
```

OUTPUT:

Hello, I am the parent process.
My PID is 5609
Hello, I am the child process.

My PID is 5610

3. Nice :

$ cat > nice3.c

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <errno.h>  // Include for errno

int main() {
        pid_t pid;
        int retnice;

        printf("Press DEL to stop the process\n");

        pid = fork();  // Create a new process

        if (pid < 0) {
        // Handle fork error
        perror("fork failed");
        return 1;
        }

        for (;;) {
        if (pid == 0) {
        // Child process
        retnice = nice(-5);  // Increase priority (lower niceness)
        if (retnice == -1) {
                if (errno == EPERM) {
                printf("Child: Operation not permitted, may need higher privileges\n");
                } else {
                perror("Child: nice failed");
                }
                return 1;
        }
        printf("Child gets higher CPU priority, new niceness: %d\n", retnice);
        sleep(1);  // Sleep for 1 second
        } else {
        // Parent process
```

```
        retnice = nice(4);  // Decrease priority (higher niceness)
        if (retnice == -1) {
                if (errno == EPERM) {
                printf("Parent: Operation not permitted, may need higher privileges\n");
                } else {
                perror("Parent: nice failed");
                }
                return 1;
        }
        printf("Parent gets lower CPU priority, new niceness: %d\n", retnice);
        sleep(1);  // Sleep for 1 second
        }
        }

        return 0;
}
```

$ gcc -o nice_a nice3.c

$ ./nice_a

```
Press DEL to stop the process
Parent gets lower CPU priority, new niceness: 4
Child: Operation not permitted, may need higher privileges
Parent gets lower CPU priority, new niceness: 8
Parent gets lower CPU priority, new niceness: 12
Parent gets lower CPU priority, new niceness: 16
Parent gets lower CPU priority, new niceness: 19
Parent gets lower CPU priority, new niceness: 19
Parent gets lower CPU priority, new niceness: 19
^Z
[1]+  Stopped                 ./nice_a
```

~$ sudo ./nice_a

4.Orphan process

$ cat > shell2.c

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

```
int main() {
        pid_t pid;

        pid = fork();

        if (pid > 0) {
            // Child process
        printf("Parent process\n");
        printf("ID : %d\n", getpid());
        printf("Parent ID : %d\n\n", getppid());
        wait(NULL);
        } else if (pid == 0) {
        printf("Child process\n");
        printf("ID : %d\n", getpid());
        printf("Parent ID : %d\n", getppid());

        sleep(20);          // Sleep to simulate long-running child process

       // After 10 seconds, parent process might have finished
        printf("\nChild process (after sleep)\n");
        printf("ID : %d\n", getpid());
        printf("Parent ID : %d\n", getppid());
    // Optionally, exit the child process
        exit(0);
        } else {
        perror("Failed to create child process");
        exit(EXIT_FAILURE);
        }

        return 0;
}
```

$ gcc -o shell_a shell2.c

$ ./shell_a

```
Parent process
ID : 10334
Parent ID : 10222

Child process
ID : 10335
Parent ID : 10334
```

Child process (after sleep)
ID : 10335
Parent ID : 10334

5. Wait ()

~$ cat > wait.c

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

int main() {
        pid_t pid;
        int status;

        pid = fork();

        if (pid > 0) {
        // Parent process
        printf("Parent process: waiting for child to finish...\n");
        pid_t child_pid = wait(&status);

        if (WIFEXITED(status)) {
        printf("Child process %d exited with status %d\n", child_pid, WEXITSTATUS(status));
        } else if (WIFSIGNALED(status)) {
        printf("Child process %d killed by signal %d\n", child_pid, WTERMSIG(status));
        }

        } else if (pid == 0) {
        // Child process
        printf("Child process: doing some work...\n");
        sleep(2);  // Simulate work
        printf("Child process: exiting...\n");
        exit(42);  // Exit with status 42
        } else {
        perror("fork failed");
        exit(EXIT_FAILURE);
        }
```

```
        return 0;
}
```

~$ gcc -o waitt wait.c

~$ ./waitt

```
Parent process: waiting for child to finish...
Child process: doing some work...
Child process: exiting...
Child process 10559 exited with status 42
```

6.Clock

$ cat > clock.c

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

int main() {
        clock_t start, end;
        double cpu_time_used;

        start = clock();

        for (volatile long i = 0; i < 1000000000; i++);

        end = clock();

        if (start == (clock_t)-1 || end == (clock_t)-1) {
        perror("clock failed");
        exit(EXIT_FAILURE);
        }

        cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
        printf("Time taken: %f seconds\n", cpu_time_used);

        return 0;
}
```

$ gcc -o clockk clock.c

$ ./clockk

Time taken: 1.785825 seconds

`execlp()` :function

`execlp()` to execute the `ls` command with the `-l` option:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main() {
        printf("Executing `ls -l`...\n");

        // Replace the current process image with `ls -l`
        execlp("ls", "ls", "-l", NULL);

        // If execlp() fails, print an error message
        perror("execlp failed");

        return 0;
}
```

```
$ gcc -o execlp_b execlp.c

$ ./execlp_b
```

Output:

```
Executing `ls -l`...
total 280
-rw-rw-r-- 1 bcasci-19 bcasci-19   174 Aug  2 09:08 addition.c
drwxrwxr-x 5 bcasci-19 bcasci-19  4096 Aug  2 11:14 addition.java
-rw-rw-r-- 1 bcasci-19 bcasci-19   880 Aug  2 09:58 additionjava.class
-rw-rw-r-- 1 bcasci-19 bcasci-19   194 Aug  2 09:57 additionjava.java
-rw-rw-r-- 1 bcasci-19 bcasci-19   354 Aug  2 09:29 ascii.java
-rw-rw-r-- 1 bcasci-19 bcasci-19   356 Aug  2 09:30 AsciiValue
```

execv.c

$ cat > execv.c

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
int main()
 {
  printf(" I am in execv.c \n");
  printf("pid of hi.c is %d \n" , getpid());
 char *args[]={"./hi" , NULL};
 execv(args[0],args);
  printf("Coming back to execv.c ");
  return 0;
}
```

$ gcc -o execv execv.c

$ ./execv

```
 I am in execv.c
pid of hi.c is 6889
Coming back to execv.c
```