Assignment – 3

Q1. Create Child Process and Display PID and Termination Status

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

int main() {
    pid_t child_pid;
    int status;

    child_pid = fork(); // Create a child process

    if (child_pid == -1) {
        perror("Fork failed");
        exit(EXIT_FAILURE);
    }

    if (child_pid == 0) {
        // Child process
        printf("Child process PID: %d\n", getpid());
        exit(EXIT_SUCCESS);
    } else {
        // Parent process
        waitpid(child_pid, &status, 0); // Wait for child process to terminate
        if (WIFEXITED(status)) {
            printf("Child terminated with status: %d (0x%x)\n", WEXITSTATUS(status),
WEXITSTATUS(status));
        }
    }

    return 0;
}
```

Output:

```
chandan@CHANDAN:/mnt/c/Users/hp/Desktop/CodeForces$ gcc ass.c
chandan@CHANDAN:/mnt/c/Users/hp/Desktop/CodeForces$ ./a.out
Child process received: Hello from parent!#?
chandan@CHANDAN:/mnt/c/Users/hp/Desktop/CodeForces$
```

Q2. Parent-Child Process Communication via Pipe

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

int main() {
    int pipe_fd[2];
    pid_t child_pid;
    char buffer[100];

    if (pipe(pipe_fd) == -1) {
        perror("Pipe creation failed");
        exit(EXIT_FAILURE);
    }

    child_pid = fork();

    if (child_pid == -1) {
        perror("Fork failed");
        exit(EXIT_FAILURE);
    }

    if (child_pid == 0) {
        // Child process
        close(pipe_fd[1]); // Close write end of the pipe in the child process
        read(pipe_fd[0], buffer, sizeof(buffer));
        printf("Child process received: %s", buffer);
        close(pipe_fd[0]);
    } else {
        // Parent process
        close(pipe_fd[0]); // Close read end of the pipe in the parent process
        write(pipe_fd[1], "Hello from parent!", 18);
        close(pipe_fd[1]);
        wait(NULL); // Wait for child to finish
    }

    return 0;
}
```

Output :

```
chandan@CHANDAN:/mnt/c/Users/hp/Desktop/CodeForces$ gcc ass.c
chandan@CHANDAN:/mnt/c/Users/hp/Desktop/CodeForces$ ./a.out
Child process PID: 71
Child terminated with status: 0 (0x0)
chandan@CHANDAN:/mnt/c/Users/hp/Desktop/CodeForces$
```

Q3. Use the ps, ps lx, pstree, and -aux, command to display the attributes.

```
chandan@CHANDAN:/mnt/c/Users/hp/Desktop/CodeForces$ ps
  PID TTY          TIME CMD
    9 tty1     00:00:00 bash
   97 tty1     00:00:00 ps
chandan@CHANDAN:/mnt/c/Users/hp/Desktop/CodeForces$ ps lx
F   UID   PID  PPID PRI  NI    VSZ   RSS WCHAN  STAT TTY        TIME COMMAND
0  1000     9     8  20   0  18076  3644 -      S    tty1       0:00 -bash
0  1000    98     9  20   0  18584  1744 -      R    tty1       0:00 ps lx
chandan@CHANDAN:/mnt/c/Users/hp/Desktop/CodeForces$ pstree
init─┬─init───bash───pstree
     └─2*[{init}]
chandan@CHANDAN:/mnt/c/Users/hp/Desktop/CodeForces$ ps -aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0   9848   564 ?        Ssl  22:11   0:00 /init
root         8  0.0  0.0   9868   312 tty1     Ss   22:11   0:00 /init
chandan      9  0.0  0.0  18076  3644 tty1     S    22:11   0:00 -bash
chandan    100  0.0  0.0  18904  2048 tty1     R    22:23   0:00 ps -aux
chandan@CHANDAN:/mnt/c/Users/hp/Desktop/CodeForces$
```

Q4. In a C program, print the address of the variable and enter into a long loop (say using while(1))

a) Start three to four processes of the same program and observe the printed address values.

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

int main() {
    int variable = 42;

    pid_t child_pid;

    printf("Process ID: %d\n", getpid());

    while (1) {
        printf("Address of variable: %p\n", (void*)&variable);
        sleep(1);
    }

    return 0;
}
```

```
chandan@CHANDAN:/mnt/c/Users/hp/Desktop/CodeForces$ gcc ass.c
chandan@CHANDAN:/mnt/c/Users/hp/Desktop/CodeForces$ ./a.out
Process ID: 137
Address of variable: 0x7fffd9109e74
Address of variable: 0x7fffd9109e74
Address of variable: 0x7fffd9109e74
Address of variable: 0x7fffd9109e74
Address of variable: 0x7fffd9109e74
Address of variable: 0x7fffd9109e74
Address of variable: 0x7fffd9109e74
Address of variable: 0x7fffd9109e74
Address of variable: 0x7fffd9109e74
Address of variable: 0x7fffd9109e74
Address of variable: 0x7fffd9109e74
^C
chandan@CHANDAN:/mnt/c/Users/hp/Desktop/CodeForces$
```

b) Show how two processes which are members of the relationship parent-child are concurrent from execution point of view, initially the child is copy of the parent, but every process has its own data.

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

int main() {
    int variable = 42;

    pid_t child_pid = fork();

    if (child_pid == -1) {
        perror("Fork failed");
```

```c
        return 1;
    }

    if (child_pid == 0) {
        // Child process
        printf("Child Process - PID: %d\n", getpid());
        printf("Child Process - Variable Address: %p\n", (void*)&variable);
        variable = 100; // Modify the child's copy of the variable
    } else {
        // Parent process
        printf("Parent Process - PID: %d\n", getpid());
        printf("Parent Process - Variable Address: %p\n", (void*)&variable);
        variable = 200; // Modify the parent's copy of the variable
    }

    // Both parent and child continue executing from here

    while (1) {
        printf("Process - PID: %d, Variable Value: %d\n", getpid(), variable);
        sleep(1);
    }

    return 0;
}
```

```
chandan@CHANDAN:/mnt/c/Users/hp/Desktop/CodeForces$ gcc ass.c
chandan@CHANDAN:/mnt/c/Users/hp/Desktop/CodeForces$ ./a.out
Parent Process - PID: 147
Child Process - PID: 148
Parent Process - Variable Address: 0x7fffe3566140
Child Process - Variable Address: 0x7fffe3566140
Process - PID: 147, Variable Value: 200
Process - PID: 148, Variable Value: 100
Process - PID: 147, Variable Value: 200
Process - PID: 148, Variable Value: 100
Process - PID: 148, Variable Value: 100
Process - PID: 147, Variable Value: 200
Process - PID: 148, Variable Value: 100
Process - PID: 147, Variable Value: 200
Process - PID: 148, Variable Value: 100
Process - PID: 147, Variable Value: 200
Process - PID: 148, Variable Value: 100
Process - PID: 147, Variable Value: 200
Process - PID: 148, Variable Value: 100
Process - PID: 147, Variable Value: 200
^C
```

Q5. Implement inter process communication when

a) **Two process are related**

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

int main() {
    int pipe_fd[2];
    pid_t child_pid;
    char message[] = "Hello, child!";

    if (pipe(pipe_fd) == -1) {
        perror("Pipe creation failed");
        exit(EXIT_FAILURE);
    }

    child_pid = fork();

    if (child_pid == -1) {
        perror("Fork failed");
        exit(EXIT_FAILURE);
    }

    if (child_pid == 0) {
        // Child process
        close(pipe_fd[1]); // Close write end of the pipe in the child process
        char buffer[100];
        read(pipe_fd[0], buffer, sizeof(buffer));
        printf("Child received: %s\n", buffer);
        close(pipe_fd[0]);
    } else {
        // Parent process
        close(pipe_fd[0]); // Close read end of the pipe in the parent process
        write(pipe_fd[1], message, sizeof(message));
        close(pipe_fd[1]);
        wait(NULL); // Wait for child to finish
    }

    return 0;
}
```

Output :

```
chandan@CHANDAN:/mnt/c/Users/hp/Desktop/CodeForces$ gcc ass.c
chandan@CHANDAN:/mnt/c/Users/hp/Desktop/CodeForces$ ./a.out
Child received: Hello, child!
```

b) Two process are not related

```c
// This is the writer program

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

int main() {
    int fd;
    char *pipeName = "/tmp/myfifo";

    mkfifo(pipeName, 0666);
    fd = open(pipeName, O_WRONLY);

    char message[] = "Hello from the writer!";
    write(fd, message, sizeof(message));
    close(fd);

    return 0;
}
```

```c
// This is the reader program

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

int main() {
    int fd;
    char *pipeName = "/tmp/myfifo";

    fd = open(pipeName, O_RDONLY);

    char buffer[100];
    read(fd, buffer, sizeof(buffer));
    printf("Received: %s\n", buffer);
    close(fd);

    return 0;
}
```

```
chandan@CHANDAN:/mnt/c/Users/hp/Desktop/CodeForces$ gcc reader.c -o reader
chandan@CHANDAN:/mnt/c/Users/hp/Desktop/CodeForces$ ./writer & ./reader
[1] 251
Received:
Received:
[1]+  Done                    ./writer
chandan@CHANDAN:/mnt/c/Users/hp/Desktop/CodeForces$
```