Week 1:

1. Write a program to create a child process using system call fork().

```
#include <stdio.h>

#include <unistd.h>

int main() {

  // Create a child process

  pid_t pid = fork();

  if (pid < 0) {

    // Fork failed

    printf("Fork failed!\n");

    return 1;

  } else if (pid == 0) {

    // This is the child process

    printf("Hello from the Child Process! PID: %d\n", getpid());

  } else {

    // This is the parent process

    printf("Hello from the Parent Process! PID: %d, Child PID: %d\n", getpid(), pid);

  }

  return 0;

}
```

2. Write a program to print process Id's of parent and child process i.e. parent should print its own and its child process id while child process should print its own and its parent process id. (use getpid(), getppid())

```
#include <stdio.h>

#include <unistd.h>


int main() {

  // Create a child process

  pid_t pid = fork();


  if (pid < 0) {

    // Fork failed

    printf("Fork failed!\n");

    return 1;

  } else if (pid == 0) {

    // This is the child process

    printf("Child Process:\n");

    printf("PID: %d, Parent PID: %d\n", getpid(), getppid());

  } else {

    // This is the parent process

    printf("Parent Process:\n");

    printf("PID: %d, Child PID: %d\n", getpid(), pid);

  }


  return 0;

}
```

3. Write a program to create child process which will list all the files present in your system. Make sure that parent process waits until child has not completed its execution. (use wait(), exit())

What will happen if parent process dies before child process? Illustrate it by creating one more child of parent process.

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

int main() {
  pid_t pid1, pid2;

  // Create the first child process
  pid1 = fork();

  if (pid1 < 0) {
    // Fork failed
    printf("Fork failed!\n");
    return 1;
  } else if (pid1 == 0) {
    // This is the first child process
    printf("Child Process 1 (PID: %d): Listing files...\n", getpid());
    execlp("ls", "ls", "-l", (char *)NULL);
    exit(0);
  } else {
    // Parent process waits for the first child to complete
    wait(NULL);
    printf("Parent Process (PID: %d): First child completed.\n", getpid());

    // Create the second child process
    pid2 = fork();

    if (pid2 < 0) {
      // Fork failed
      printf("Fork failed!\n");
      return 1;
    } else if (pid2 == 0) {
      // This is the second child process
      printf("Child Process 2 (PID: %d): I am the second child.\n", getpid());
      sleep(5);  // Simulate some work
      printf("Child Process 2 (PID: %d): Work done.\n", getpid());
      exit(0);
    } else {
      // Parent process dies before the second child finishes
      printf("Parent Process (PID: %d): Exiting now.\n", getpid());
      exit(0);
    }
  }
}
```