3.1 WAP to overlay the image of the current process with some other process by calling any of the exec functions.

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

int main()
{
     char *filename = "ls";
     char *arg1 = "-a";
     char *arg2 = "-s";

     printf("Before calling exec function.");

     execlp(filename, arg1, arg2, NULL);

     printf("\nWill this line be printed? Check!");

     return 0;
}
```

3.2 WAP to overlay the image of the child process with **ps –Tl** using execl() function.

Spawning of a Process: When the image of of a (child) process, created by calling fork(), is overlayed with the process image of some other process by calling any function of the exec family, then it is called the spawning of a process.

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

int main()
{
     pid_t pid;
     pid = fork();
     char *bin_path = "/bin/ps";
     char *arg1 = "-Tl";

     if(pid<0)
     {
          printf("fork failed");
          return 1;
     }
     else if(pid == 0)
     {
          execl(bin_path, bin_path, arg1, NULL);
     }
     else
     {
          wait(NULL);
          printf("child complete\n");
     }
     return 0;
}
```

3.3 WAP to overlay the image of the child process with **ls -a -l** using execlp() function.

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

int main()
{
     char *filename = "ls";
     char *arg1 = "-a";
     char *arg2 = "-s";

     pid_t pid;
     pid = fork();

     if(pid < 0)
     {
          printf("fork failed!");
          return -1;
     }
     else if(pid == 0)
     {
          execlp(filename, arg1, arg2, NULL);
     }
     else
     {
          wait(NULL);
          printf("child complete\n");
     }
     return 0;
}
```

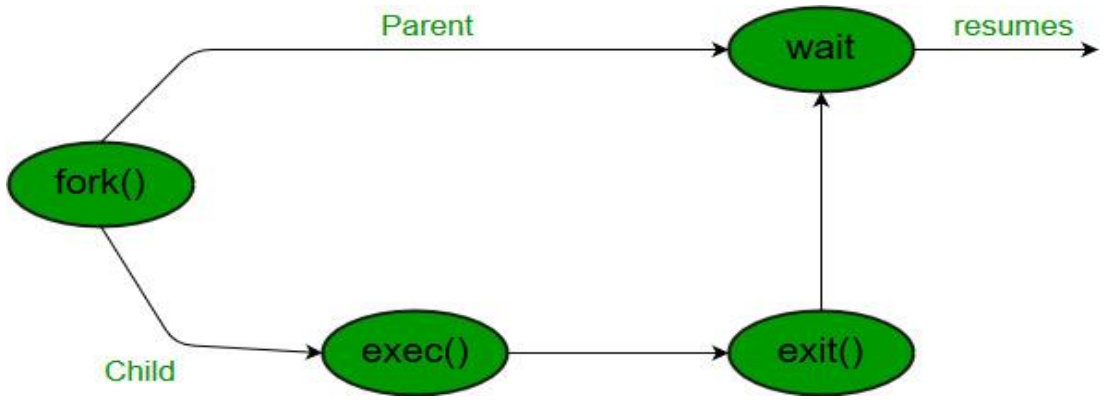
3.4 WAP to avoid the zombie or orphan status of a child process.

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

int main()
{
     int stat;

     // This exit status (99) is reported by WEXITSTATUS
     if (fork() == 0)
          exit(99);
     else
          wait(&stat);    // Parent waits for child to exit.
     if (WIFEXITED(stat))
          printf("Exit status: %d\n", WEXITSTATUS(stat));
     else if (WIFSIGNALED(stat))
          printf("\nExit signal");

     return 0;
}
```

Note: A call to wait() blocks the calling process until one of its child processes exits or a signal is received. After child process terminates, parent continues its execution after wait system call instruction.

Child process may terminate due to any of these:

- It calls exit();
- It returns (an int) from main
- It receives a signal (from the OS or another process) whose default action is to terminate.



The wait function takes one argument **status** and returns a process ID of the dead children.

```
pid_t wait(int *stat_loc);
```

3.5 WAP to make the parent process wait for a specific child. The parent process should wait for multiple children one by one.

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

int main()
{
     int i, stat;
     pid_t pid[5];
     for (i=0; i<5; i++)
     {
          if ((pid[i] = fork()) == 0)
          {
               sleep(1);
               exit(0 + i);
          }
     }

     // Usage of waitpid() to wait for a specific child
     for (i=0; i<5; i++)
     {
          pid_t cpid = waitpid(pid[i], &stat, 0);
          if (WIFEXITED(stat))
               printf("Child %d terminated with status: %d\n",
                    cpid, WEXITSTATUS(stat));
     }
     return 0;
}
```

Ask yourself.

What is the difference between **system("...")** call and the functions under exec family?