Assignment 5 zomb_orph.c

Create a scenario to make zombie process become orphan, print status of each state.
Pre-requisites:-

- Knowledge about system calls, How to read and understand 'man pages'.
- Good knowledge about processes, zombie and orphan.
- Working of fork system call.

Objective: -

- To understand different states of a process.

Requirements: -

1. Create a child process and print status that process is running
2. After some time print status that process is zombie state
3. After some time print zombie process cleared by init.
4. To print status use help of **/proc/<pid>/status** file.
   For eg: if child pid is 1234, open file /proc/1234/status and print first 3 lines
5. If that file is not available means that process is cleared.

Sample execution: -

1. ./zomb_orph

```
A child created with pid 1234

Name:       ./zomb_orph
State:     S (sleeping)

Name:       /zomb_orph    (After some time)
State:     Z (zombie)

Process 1234 cleared by init   (After some time)
```

Assignment 6 nonblock_wait.c

WAP to avoid a child become zombie with out blocking the parent.
Pre-requisites:-

- Knowledge about system calls, How to read and understand 'man pages'.
- Good knowledge about processes & zombie process.
- Working of fork & wait system call.

Objective: -

- To understand different states of a process.

Requirements: -

1. Create a child process avoid it become a zombie.
2. To avoid zombie we need to call wait(), but this block parent until child terminates.
3. So we need to use waitpid() with proper arguments (Read man page).
4. When child is working parent has to continuously print some message.
5. When ever child terminates parent has to print child terminated and print exit status of child process.

Sample execution: -

1. ./nonblock_wait

```
A child created with pid 1234

parent is running
parent is running
parent is running
.
.
Child 1234 terminated normally with exit status 0
parent terminating
```

Assignment 7 exe_child.c

WAP to create a child process which will execute command passed through command-line arguments.

Pre-requisites:-

- Knowledge about system calls, How to read and understand 'man pages'.
- Good knowledge about processes.
- Working of fork, wait and exec system calls.

Objective: -

- To understand how to use exec system calls.

Requirements: -

1. Create child and execute a command passed from command-line arguments.
2. If no arguments passed print a usage message.
3. In case any wrong command passed, print an error message.
4. After child terminates print the exit status of child process.

Sample execution: -

1. *No args passed (Print usage info)*

```
./exe_child
Usage:
./exe_child args...
```

2. *Valid arg. passed*

```
./exe_child date
This is the CHILD process, with id 11612
Wed Apr  4 13:27:19 IST 2012
Child exited with status 0
```

3. *Child terminated using SIGKILL (kill -9)*

```
./exe_child sleep 20 &
[1] 11617
$ This is the CHILD process, with id 11618

$ kill -9 11618
Child terminated abnormally
Child exited with code 9
```

Assignment 8 three_child.c

WAP to create three child processes from same parent.

Pre-requisites:-

- Knowledge about system calls, How to read and understand 'man pages'.
- Good knowledge about processes.
- Working of fork & wait system calls.

Objective: -

- To understand how to use fork system calls.

Requirements: -

1. Create three child process from same parent.
2. Parent has to wait for all three child process.
3. Print exit status of each child when they terminates.

Sample execution: -

1. ./three_child

```
Child 1 with pid 2020 created
Child 2 with pid 2021 created
Child 3 with pid 2022 created
Child 2020 is terminated with code 0
Child 2021 is terminated with code 0
Child 2022 is terminated with code 0
```