Assignment 1 sigsegv.c

WAP to print the address which causing segmentation fault.
Pre-requisites:-

- Knowledge about system calls, How to read and understand 'man pages'.
- Good knowledge about signals and signal handling.
- Working of sigaction system calls.

Objective: -

- To understand working of signal handling.

Requirements: -

1. Write a signal handler function to print address which caused seg-fault(SIGSEGV).
2. Use sigaction system call to register signal.
3. Use struct siginfo from sa_sigaction to print address (Read man page).
4. Create a segmentation fault from code.
5. When error occurs program should print address and exit.

Sample execution: -

1. **./sigsegv**
```
Segmentation fault ..!!
Address 0x123456 caused error
```

Assignment 2 alarm.c

WAP to implement alarm with snooze for given time and date using **SIGALRM**
Pre-requisites:-

- Knowledge about system calls, How to read and understand 'man pages'.
- Good knowledge about signals and signal handlers.
- Working of alarm system calls.

Objective: -

- To understand signals and time related system calls.

Requirements: -

1. User gives the time and date from command-line arguments.
2. Validate the time eg: Do not go behind the current time.
3. Date is an option, if user not providing date consider it as today.
4. In handler, avoid all user communication(**printf, scanf** etc) and loops.
   Make it minimal as possible.
5. After the alarm expires, display a ALARM message along with date and time.
6. Prompt the user whether he wants to stop or snooze.
7. If user selects stop, terminate the program.
8. If user selects snooze, prompt for snooze time in minutes.
   > If user enters the time, reset the alarm to the entered time in minutes
   > If user doesn't enter time, default the snooze time to 1 mins

Sample execution: -

1. `./alarm` (No arguments)
```
Error: No arguments passed
```

```
Usage: ./alarm <hh:mm> [dd/mm/yy]
```

2. `./alarm 30:15`
   ```
   Error: Invalid time
   ```
3. `./alarm 22:10`
   ```
   set alarm for 10:10 PM for today
   ```
4. `./alarm 22:10 02/03/16`
   ```
   set alarm for 10:10 PM for today
   ```
5. When alarm occurs

   ```
   Wake-up...Alarm..!!
   1. Snooze     2. Exit
   ```
   if user select 1
   ```
   Enter snooze time
   2
   ```
   After 2 mins above process will repeat until user gives exit

Hints:

- Use strptime to convert string to time (Refer man page)
- Other useful functions are localtime, mktime, strptime

Assignment 3 block_signal.c

WAP to block certain signals from being received from command-line
Pre-requisites:-
- Knowledge about system calls, How to read and understand 'man pages'.
- Good knowledge about signals and signal handling.
- Working of sigaction system call and signal masking.

Objective: -
- To understand importance of signal masking.

Requirements: -
1. Write a signal handler function for any signal, say SIGINT .
2. While its running inside handler (use loop) block other signals(atleast 3) being received .
3. Use ***sa_mask*** from ***struct sigaction*** to set signals to be blocked (Refer man).
4. To create a signal set use variable of ***sigset_t***.
5. To add or remove signals from set use ***sigaddset***, ***sigdelset*** functions (refer man).
6. Process will receive blocked signals once its out from handler.

Sample execution: -

1. **./block_signal**
   ```
   Process 2345 waiting for signal..
   ```
   press ctrl + c from terminal.
   ```
   SIGINT received
   Inside handler
   Inside handler
   .
   .
   ```

-

2. Now open another terminal and send signal to process using kill command.
   ```
   Bash$ kill -SIGUSR1 2345
   Bash$ kill -SIGTERM 2345
   Bash$ kill -SIGABRT 2345
   ```
3. After exiting from handler will respond to blocked signal.

Assignment 4 avoid_zombie.c

WAP to avoid a child become zombie by using signal handlers.
Implement it with two different method.
Pre-requisites:-

- Knowledge about system calls, How to read and understand 'man pages'.
- Good knowledge about signals and signal handling.
- Working of sigaction system call.

Objective: -

- To understand, how to avoid zombie asynchronously .

Requirements: -

Write two separate programs for both methods.

<u>Method 1</u>

1. Create a child process.
2. Write a signal handler function for **SIGCHLD** to avoid child become zombie (Do **man 7 signal** for SIGCHLD) .
3. Write code inside handler to avoid zombie and print child exit status.

<u>Method 2</u>

4. Create a child process.
5. Use ***sa_flag*** from ***struct sigaction*** to avoid zombie (Refer man).
6. Prints the child exit status inside handler.

Sample execution: -

1.
   ```
   ./avoid_zombie
   child <pid> terminated with exit code 0.
   ```