Write a Menu driven program to demonstrate zombie process and orphan process.

#!/bin/bash

# Function to create a zombie process

create\_zombie() {

echo "Creating a Zombie Process..."

(sleep 10) & # Child process sleeps for 10 seconds

sleep 5 # Parent sleeps to let child become a zombie

echo "Parent process exiting."

}

# Function to create an orphan process

create\_orphan() {

echo "Creating an Orphan Process..."

(while true; do echo "Orphan PID: $$"; sleep 2; done) & # Child runs indefinitely

echo "Parent process exiting. Child will become an orphan."

}

# Main menu loop

while true; do

echo "Menu:"

echo "1) Create Zombie Process"

echo "2) Create Orphan Process"

echo "3) Exit"

read -p "Choose an option: " choice

case $choice in

1) create\_zombie ;;

2) create\_orphan ;;

3) echo "Exiting..."; exit ;;

\*) echo "Invalid option. Try again." ;;

esac

read -p "Press [Enter] to continue..."

clear

done

# Function to create a zombie process

create\_zombie() {

This line defines a function named create\_zombie. The comments above are for human readers, explaining what the function will do.

echo "Creating a Zombie Process..."

This line outputs a message to the terminal indicating that a Zombie Process is about to be created.

(sleep 10) & # Child process sleeps for 10 seconds

This line starts a subshell that runs the command sleep 10 in the background (indicated by the &). This creates a child process that will "die" after 10 seconds, without its parent process waiting for it to finish, thus allowing it to become a zombie.

sleep 5 # Parent sleeps to let child become a zombie

The parent process sleeps for 5 seconds, which is enough time for the child process to run for 10 seconds. This means that the child process will complete its execution while the parent is still running, allowing it to become a zombie.

echo "Parent process exiting."

After the sleep period, this line outputs a message indicating that the parent process is exiting.

}

This closes the create\_zombie function.

# Function to create an orphan process

create\_orphan() {

This line begins a new function definition named create\_orphan, again with a comment to describe what the function does.

echo "Creating an Orphan Process..."

This line outputs a message signaling that an Orphan Process is about to be created.

(while true; do echo "Orphan PID: $$"; sleep 2; done) & # Child runs indefinitely

This line starts a subshell that runs an infinite loop in the background. Inside the loop, it prints the current process ID (PID) of the orphan process ($$ refers to the PID of the current process). The loop then sleeps for 2 seconds before repeating. Because the parent will exit before this process can be reaped, the child will become an orphan.

echo "Parent process exiting. Child will become an orphan."

After starting the orphan process, this line indicates that the parent process is exiting, leading the child to become an orphan.

}

This closes the create\_orphan function.

# Main menu loop

while true; do

This begins an infinite loop, allowing the script to repeatedly display a menu until the user decides to exit.

echo "Menu:"

echo "1) Create Zombie Process"

echo "2) Create Orphan Process"

echo "3) Exit"

These lines output the menu options for the user. The user can choose to create a Zombie Process, create an Orphan Process, or exit the script.

read -p "Choose an option: " choice

This line prompts the user to enter their choice, storing it in the variable choice.

case $choice in

This begins a case statement that evaluates the value of choice.

1) create\_zombie ;;

If the user enters 1, the script calls the create\_zombie function.

2) create\_orphan ;;

If the user enters 2, the script calls the create\_orphan function.

3) echo "Exiting..."; exit ;;

If the user enters 3, it outputs a message saying "Exiting..." and then terminates the script using exit.

\*) echo "Invalid option. Try again." ;;

If the user enters anything other than 1, 2, or 3, this line executes, outputting an error message indicating that the choice was invalid.

esac

This marks the end of the case statement.

read -p "Press [Enter] to continue..."

This line pauses the script, prompting the user to press Enter before proceeding.

clear

This line clears the terminal screen, preparing it for the display of the menu again.

done

This marks the end of the while true loop, which will repeat indefinitely until the user chooses to exit.