1. Printenv prints out all the EVs, and if you want to get specific, you can use “printenv ‘name’” to get a specific EV. To set or unset EVs, we use export and unset which are only found in bash:

```
pazy@ubuntu-lab:~$ env
SHELL=/bin/bash
SESSION_MANAGER=local/ubuntu-lab:@/tmp/.ICE-unix/1738,unix/ubuntu-lab:/tmp/.ICE-
unix/1738
QT_ACCESSIBILITY=1
COLORTERM=truecolor
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
SSH_AGENT_LAUNCHER=gnome-keyring
XDG_MENU_PREFIX=gnome-
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
GNOME_SHELL_SESSION_MODE=ubuntu
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
XMODIFIERS=@im=ibus
DESKTOP_SESSION=ubuntu
GTK_MODULES=gail:atk-bridge
PWD=/home/pazy
```

2. When the program is first run, it prints all the EVs and then saves to output1. After swapping from the child to the parent process, the output is the same, meaning they share the same EVs. The diff command shows the differences between two files. After running it, we see no differences because the child and parent share the same EVs.

3. When null is used, user/bin/env prints nothing or just defaults. The third parameter controls what environment the program receives. After changing null to environ, it now prints all the environment variables from the calling process. The difference between fork and exceve is that fork doesn't need to pass EVs explicitly:

4. This is convenient but creates a security issue: users can control environment variables that affect the shell's behavior.:

5. Path is inherited, myvar is inherited, LD_Library_path is NOT inherited, this is because the dynamic linker unsets potentially dangerous EVs for set-UID programs to prevent an attacker from getting control of the path:

6. sudo ln -sf /bin/zsh /bin/sh # Needed to bypass dash countermeasure
   export PATH=/tmp:$PATH # Make /tmp first in PATH
   cp ls /tmp/ # Put malicious ls in /tmp

   - Why this worked: System(“ls”) calls /bin/sh -c “ls” then the shell looks for ls in PATH, PATH gets set to /tmp:$PATH so /tmp/ls is found first. Malicious ls is executed then with root privileges

7. -1: When a normal program is run with a normal user, “I am not sleeping.” LD_Preload overrides the sleep() function
   -2:when a normal user runs a set UID program, the program sleeps. LD_preload gets unset for the set-UID program due to security measures
   -3:when setUID root, exported in root account, “I am not sleeping”. When real UID = effective UID, LD_preload is allowed.
   -4:when setUID user1, exported as a different user, the program sleeps. When real UID does NOT = effective UID, LD_preload is filtered out.
   - The dynamic linker checks if the real and effective UIDs differ. If they do (Set-UID case), it unsets dangerous LD_* variables for security.

8. 1: Yes, it would be very easy with the System() method to add a semicolon and get access to the root user
   2: I get an error message, and the attack does not work. This is because Execve() executes /bin/cat directly with the entire string as a single filename argument. No shell interpretation occurs.
   - Execve() is safer than System()

9. # In the spawned shell
   echo "HACKED" >&3
   cat /etc/zzz
   - This attack worked, and it printed into the file. When dropping privileges, close all file descriptors opened while privileged, or use fcntl(fd, F_SETFD, FD_CLOEXEC) to mark them close-on-exec.