

Lab 1 Admin

3.1 Basic Linux Operations

How to add user with Capital letter in linux:

```
wajdabdal-hadee@wajdabdal-hadee:~$ sudo adduser --allow-bad-names Network
info: Allowing use of questionable username.
info: Adding user `Network' ...
info: Selecting UID/GID from range 1000 to 59999 ...
info: Adding new group `Network' (1002) ...
info: Adding new user `Network' (1002) with group `Network (1002)' ...
info: Creating home directory `/home/Network' ...
info: Copying files from `/etc/skel' ...
New password:
BAD PASSWORD: The password is shorter than 8 characters
Retype new password:
passwd: password updated successfully
Changing the user information for Network
Enter the new value, or press ENTER for the default
    Full Name []:
    Room Number []:
    Work Phone []:
    Home Phone []:
    Other []:
Is the information correct? [Y/n]
info: Adding new user `Network' to supplemental / extra groups `users' ...
info: Adding user `Network' to group `users' ...
```

Get root privileges using the command:

```
wajdabdal-hadee@wajdabdal-hadee:~$ sudo su
root@wajdabdal-hadee:/home/wajdabdal-hadee# whoami
root
```

3.2 Command Line Shell

الأمر	وظيفته
<code>cd</code>	التنقل بين المجلدات. <code>.</code> تعني المجلد الحالي، <code>..</code> تعني المجلد السابق، <code>~</code> تعني المجلد الرئيسي.
<code>ls</code>	عرض الملفات والمجلدات داخل المجلد الحالي.
<code>cp</code>	نسخ الملفات والمجلدات.
<code>mv</code>	نقل الملفات أو إعادة تسميتها.
<code>rm</code>	حذف الملفات والمجلدات.
<code>mkdir</code>	إنشاء مجلد جديد.
<code>rmdir</code>	حذف مجلد فارغ.
<code>touch</code>	إنشاء ملف فارغ جديد.
<code>less</code>	عرض محتوى الملفات الكبيرة بشكل تدريجي.
<code>cat</code>	طباعة محتوى ملف نصي.
<code>echo</code>	طباعة نص معين أو متغير بيئي.
<code>pwd</code>	طباعة المسار الحالي.
<code>wc</code>	عدّ الكلمات أو الأسطر أو الأحرف داخل ملف.
<code>sort</code>	ترتيب محتويات ملف نصيًا.
<code>></code>	إعادة توجيه الخرج إلى ملف جديد.
<code>ps</code>	عرض العمليات الجارية.
<code>&</code>	تشغيل عملية في الخلفية.
<code>Ctrl-c</code>	إيقاف العملية الجارية.
<code>Ctrl-z</code>	إيقاف العملية مؤقتًا.
<code>bg</code>	إعادة تشغيل عملية موقفة في الخلفية.
<code>fg</code>	إعادة تشغيل عملية موقفة في الواجهة.

3.3 Aliases, Variables, and Environmental Variables

a. Create an alias to the command `ls` and show that it is working.

```
alias ll='ls -la'
```

```
ll # Test the alias
```

```
wajdabdal-hadee@wajdabdal-hadee:~$ alias ll='ls -l'
wajdabdal-hadee@wajdabdal-hadee:~$ ll
total 48
drwxr-xr-x 2 root root 4096 Mar 15 03:52 containerdir
drwxr-xr-x 3 wajdabdal-hadee wajdabdal-hadee 4096 Mar 15 03:45 Desktop
drwxrwxr-x 2 wajdabdal-hadee wajdabdal-hadee 4096 Mar 15 04:20 dockerExample
drwxr-xr-x 2 wajdabdal-hadee wajdabdal-hadee 4096 Oct 2 19:56 Documents
drwxr-xr-x 2 wajdabdal-hadee wajdabdal-hadee 4096 Mar 15 03:45 Downloads
drwxrwxr-x 2 wajdabdal-hadee wajdabdal-hadee 4096 Mar 15 06:24 grp-members
drwxr-xr-x 2 wajdabdal-hadee wajdabdal-hadee 4096 Oct 2 19:56 Music
drwxr-xr-x 3 wajdabdal-hadee wajdabdal-hadee 4096 Nov 7 03:25 Pictures
drwxr-xr-x 2 wajdabdal-hadee wajdabdal-hadee 4096 Oct 2 19:56 Public
drwx----- 8 wajdabdal-hadee wajdabdal-hadee 4096 Mar 15 03:13 snap
drwxr-xr-x 2 wajdabdal-hadee wajdabdal-hadee 4096 Oct 2 19:56 Templates
drwxr-xr-x 2 wajdabdal-hadee wajdabdal-hadee 4096 Oct 2 19:56 Videos
```

b. Create a variable whose value is the absolute path to your current directory.

```
MYDIR=$(pwd) == `pwd`
```

c. Print the value of the variable you have created.

```
echo $MYDIR
```

```
wajdabdal-hadee@wajdabdal-hadee:~$ MYDIR=$(pwd)
wajdabdal-hadee@wajdabdal-hadee:~$ echo $MYDIR
/home/wajdabdal-hadee
```

d. Define an environmental named MYENVVAR. Set the value of this variable to be the date of today.

```
export MYENVVAR=$(date | awk '{print $7"-"$2"-"$3}') → 2025-03-23
```

e. Show that your variable has been successfully created.

```
echo $MYENVVAR
```

```
wajdabdal-hadee@wajdabdal-hadee:~$ export MYENVVAR=$(date | awk '{print $7"-"$2"-"$3}')
wajdabdal-hadee@wajdabdal-hadee:~$ echo $MYENVVAR
2025-Mar-23
```

يستخدم الكوماند ادناه لعرض جميع المتغيرات البيئية:

```
env # لعرض جميع المتغيرات البيئية
```

f. Print all the environmental variables defined in the system.

```
printenv # or env
```

g. What is the difference between an environmental variable and other variables?

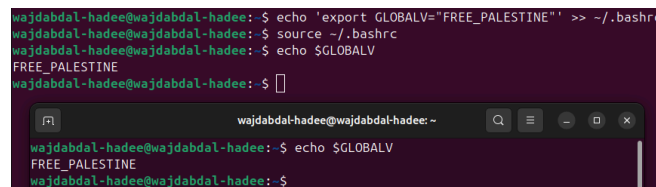
- Environmental variables are available to all processes started from the shell.
- Regular variables exist only within the current shell session.

h. Open another terminal and try to show the value of the environmental variable you have created. Can you see its value? Why?

`echo $MYENVVAR` # It won't be available because it's not persistent.

i. Create an environmental variable and make sure that it can be seen by all terminals? Show your steps in details

```
echo 'export GLOBALVAR="Free_Palestine"' >> ~/.bashrc
source ~/.bashrc
echo $GLOBALVAR
```

A screenshot of a terminal window with a dark background. The prompt is 'wajdabdal-hadee@wajdabdal-hadee: ~'. The user enters the command 'echo 'export GLOBALV="FREE_PALESTINE"' >> ~/.bashrc'. The prompt changes to 'wajdabdal-hadee@wajdabdal-hadee: ~' and the user enters 'source ~/.bashrc'. The prompt changes to 'wajdabdal-hadee@wajdabdal-hadee: ~' and the user enters 'echo \$GLOBALV'. The output is 'FREE_PALESTINE'. The prompt changes to 'wajdabdal-hadee@wajdabdal-hadee: ~' and the user enters 'echo \$GLOBALV'. The output is 'FREE_PALESTINE'. The prompt changes to 'wajdabdal-hadee@wajdabdal-hadee: ~' and the user enters '\$'. The output is '\$'.

3.4 File and Directory Validation

Write a bash shell script that receives two arguments: The first argument is the absolute path of a file you have created. The second argument is the absolute path of a directory you have created. Your script should validate both inputs by making sure that both of them exist. Then, if the validation succeeds, your script should copy the file into that directory. Otherwise, you have to print an error message stating which argument causes validation failure.

```
#!/bin/bash
```

#Check if two arguments are provided

```
if [ "$#" -ne 2 ]; then
echo "Usage: $0 <file_path> <directory_path>"
exit 1
fi
```

#Assign arguments to variables

```
FILE_PATH=$1
DIR_PATH=$2
```

#Validate file existence

```
if [ ! -f "$FILE_PATH" ]; then
echo "Error: File '$FILE_PATH' does not exist."
exit 1
fi
```

#Validate directory existence

```
if [ ! -d "$DIR_PATH" ]; then
echo "Error: Directory '$DIR_PATH' does not exist."
exit 1
fi
```

#Copy the file into the directory

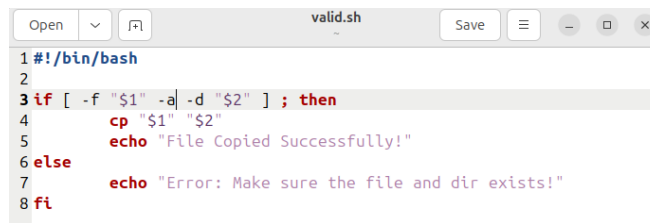
```
cp "$FILE_PATH" "$DIR_PATH"
```

#Check if copy operation was successful

```

if [ $? -eq 0 ]; then
echo "File '$FILE_PATH' successfully copied to '$DIR_PATH'."
else
echo "Error: Failed to copy file."
exit 1
fi

```



```

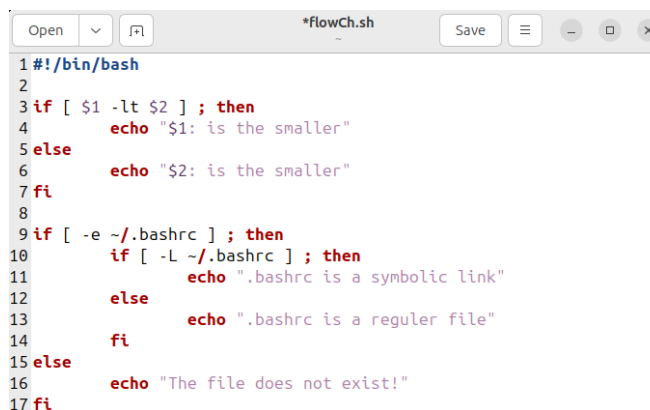
1 #!/bin/bash
2
3 if [ -f "$1" -a -d "$2" ] ; then
4     cp "$1" "$2"
5     echo "File Copied Successfully!"
6 else
7     echo "Error: Make sure the file and dir exists!"
8 fi

```

3.5 Flow Control

Write a bash shell script that works as follows:

- Receives the values of two integers (X and Y) from the command line.
- Compare both variables and print the smaller one.
- Checks whether the file .bashrc exists.
- If the file exists, the script checks whether it is a symbolic link or a regular file. Based on that check, the script has to be print its type.
- If the file does not exist, the script should state that.



```

1 #!/bin/bash
2
3 if [ $1 -lt $2 ] ; then
4     echo "$1: is the smaller"
5 else
6     echo "$2: is the smaller"
7 fi
8
9 if [ -e ~/.bashrc ] ; then
10     if [ -L ~/.bashrc ] ; then
11         echo ".bashrc is a symbolic link"
12     else
13         echo ".bashrc is a reguler file"
14     fi
15 else
16     echo "The file does not exist!"
17 fi

```

1. شرح e- و L-

- e- : يتحقق مما إذا كان الملف موجودًا أم لا، بغض النظر عن نوعه (عادي، دليل، رابط رمزي، إلخ).
- L- : يتحقق مما إذا كان الملف عبارة عن رابط رمزي (symbolic link).

3.6 Process Parameters

a. Show all the running processes in your machine.

`ps aux`

b. Write the PIDs for all processes whose owner is only the root on a file named rootPIDs.txt

`ps aux | grep '^root' | awk '{print $2}' > rootPIDs.txt`

حيث أن:

الجزء `'root^'` في `grep` يستخدم التعبير النمطي (Regular Expression)، والرمز `^` في `Regex` يعني بداية السطر.

تفسير `root^`

- `^` → يطابق بداية السطر.
- `root` → يجب أن يكون أول شيء في السطر هو `root`.

or

`ps -u root -o pid > rootPIDs.txt`

```
wajdabdal-hadee@wajdabdal-hadee:~$ ps -u root -o pid > rootPIDs.txt
wajdabdal-hadee@wajdabdal-hadee:~$ ls
containerdir  Documents  grp-members  Public      Templates  Videos
Desktop       Downloads  Music        rootPIDs.txt test
dockerExample flowCh.sh  Pictures     snap        valid.sh
wajdabdal-hadee@wajdabdal-hadee:~$ cat rootPIDs.txt
PID
1
2
3
4
5
6
7
10
11
```

حيث أن

```
ps -u root -o pid > rootPIDs.txt
```

تحليل الأمر بشكل مرتب ومفهوم

ps 1

• هو أمر في لينكس/يونكس يستخدم لعرض العمليات الجارية في النظام.

u root- 2

• يحدد أن العمليات التي سيتم عرضها تخص المستخدم root فقط.

o pid- 3

• -o (output format) : تحدد الأعمدة التي سيتم عرضها.

• pid : تعني عرض معرف العملية (Process ID) فقط، دون أي معلومات إضافية.

rootPIDs.txt < 4

• < : تعيد توجيه الإخراج إلى ملف بدلاً من عرضه على الشاشة.

• rootPIDs.txt هو الملف الذي سيتم حفظ معرفات الع (PIDs) فيه.

. root الخاصة بالمستخدم PIDs كلا الأمرين يؤديان إلى نفس الهدف: استخراج

⚡ الفرق الرئيسي:

- root يستخدم الطريقة المباشرة لتحديد العمليات الخاصة بـ
- awk و grep يعتمد على تصفية الإخراج يدويًا باستخدام

c. Write the PIDs for all processes whose parent is the init process on a file named initPIDs.txt

```
ps -ef | awk '$3 == 1 {print $2}' > initPIDs.txt
```

```
wajdabdal-hadee@wajdabdal-hadee:~$ ps -ef | awk '$3 == 1 {print $2}' > initPIDs.txt
wajdabdal-hadee@wajdabdal-hadee:~$ ls
containerdir  Documents  grp-members  Pictures      snap      valid.sh
Desktop       Downloads  initPIDs.txt  Public        Templates Videos
dockerExample flowCh.sh  Music         rootPIDs.txt  test
wajdabdal-hadee@wajdabdal-hadee:~$ cat initPIDs.txt
274
350
452
458
463
985
986
990
995
998
```

/// **Explanation:** The `init` process (or `systemd` in newer systems) always has **PID 1**.

d. Manually run the Firefox and pick the process associated with it. Explain all the attributes of that process.

1. `firefox &`
2. `ps aux | grep firefox`

1. Run Firefox from the terminal:

```
bash
```

[Copy](#)[Edit](#)

```
firefox &
```

2. Find the Firefox process:

```
bash
```

[Copy](#)[Edit](#)

```
ps aux | grep firefox
```

3. Explanation of attributes (Example output):

```
swift
```

[Copy](#)[Edit](#)

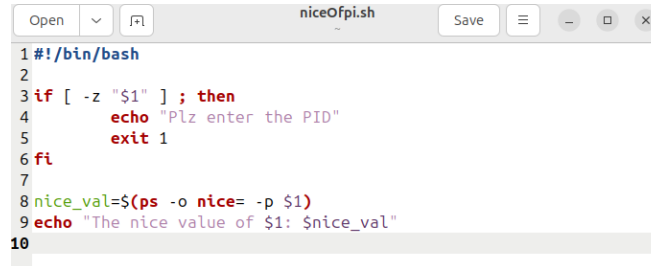
```
user      1234  3.2  1.5  824512 128456 ?        S1   10:45   1:20 /usr/lib/firefox/firefox
```

- `user` → The owner of the process.
- `1234` → PID (Process ID).
- `3.2` → %CPU usage.
- `1.5` → %Memory usage.
- `824512` → Virtual memory size (VSZ) in KB.
- `128456` → Resident memory size (RSS) in KB.
- `?` → The terminal that launched the process (`?` means no associated terminal).
- `S1` → Process state (`S`: sleeping, `1`: multi-threaded).
- `10:45` → Time when the process started.
- `1:20` → Total CPU time used by the process.
- `/usr/lib/firefox/firefox` → The full path of the process.

e. Get the PID of the terminal process you are running.

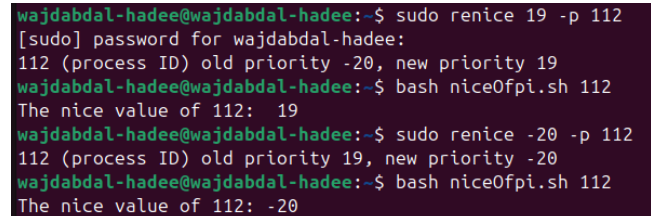
```
echo $$
```

f. Write a bash shell script that receives a process PID as an argument through the command line. Your script should return the nice value of that process.

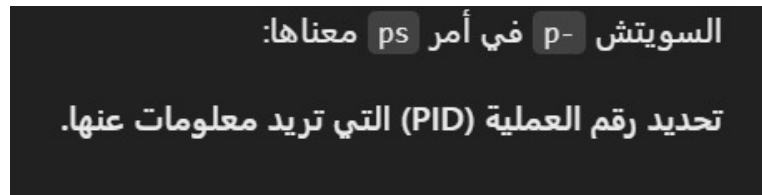


```
1#!/bin/bash
2
3if [ -z "$1" ] ; then
4    echo "Plz enter the PID"
5    exit 1
6fi
7
8nice_val=$(ps -o nice= -p $1)
9echo "The nice value of $1: $nice_val"
10
```

checking:



```
wajdabdal-hadee@wajdabdal-hadee:~$ sudo renice 19 -p 112
[sudo] password for wajdabdal-hadee:
112 (process ID) old priority -20, new priority 19
wajdabdal-hadee@wajdabdal-hadee:~$ bash niceOfpi.sh 112
The nice value of 112: 19
wajdabdal-hadee@wajdabdal-hadee:~$ sudo renice -20 -p 112
112 (process ID) old priority 19, new priority -20
wajdabdal-hadee@wajdabdal-hadee:~$ bash niceOfpi.sh 112
The nice value of 112: -20
```



السويتش `p-` في أمر `ps` معناها:
تحديد رقم العملية (PID) التي تريد معلومات عنها.

g. What is a nice value?

- **The "nice" value** determines the priority of a process in Linux.
- It ranges from **-20 (highest priority) to 19 (lowest priority)**.
- A lower nice value means higher priority for CPU resources.
- Users can set the nice value using the `nice` or `renice` commands.

h. Write a command that maximizes the nice value of a randomly chosen running process.

```
wajdabdal-hadee@wajdabdal-hadee:~$ sudo renice 19 -p 112
[sudo] password for wajdabdal-hadee:
112 (process ID) old priority -20, new priority 19
```

♦ ما هو nice value؟

- **nice value** هو رقم يتراوح بين -20 (الأولوية الأعلى) و 19 (الأولوية الأقل).
- كلما زادت قيمة **nice**، قلت أولوية العملية على المعالج.
- يمكن تغيير قيمة **nice** باستخدام الأمر **renice**.

♦ كيفية اختيار عملية عشوائية؟

1. نستخدم **ps** لجلب قائمة العمليات.
2. نستخدم **shuf -n 1** لاختيار PID عشوائي من القائمة.
3. نستخدم **renice** لضبط قيمة **nice** إلى 19 (القيمة القصوى).

♦ الأمر المطلوب:

```
renice 19 -p $(ps -eo pid --no-headers | shuf -n 1)
```

🔍 شرح الأمر:

1. **ps -eo pid --no-headers** → يجلب قائمة بمعرفات العمليات (PIDs) بدون ترويسة.
2. **shuf -n 1** → يختار PID عشوائي من القائمة.

```
renice 19 -p $(ps -eo pid --no-headers | shuf -n 1)
```

3.7 Process Monitoring and Signaling

- a. Write a command that dynamically monitors all the running processes.

top

b. Write a C code for a process that forks a child process every minute. Trace the running execution of that process and capture when the fork system call is invoked.

The code:

```
Open  edit.c  Save  [Icons]
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <stdlib.h>
4
5 int main(){
6     while (1){
7         pid_t pid = fork();
8         if (pid == 0 ){
9             printf("Child process created with PID: %d\n", getpid());
10            exit (0);
11        }
12
13        sleep (60);
14    }
15    return 0;
16 }
```

The use of gcc:

`gcc edit.c -o editFork`

```
wajdabdal-hadee@wajdabdal-hadee:~$ gcc edit.c -o editFork
```

1. gcc

- هذا هو المترجم (Compiler) الذي يقوم بتحويل كود C إلى ملف تنفيذي.

2. edit.c

- هذا هو ملف الكود المصدري (Source Code) بلغة C، والذي نريد تحويله إلى برنامج قابل للتشغيل.

3. o editFork-

- الخيار -o يحدد اسم الملف التنفيذي الناتج، في هذه الحالة سيكون اسمه `editFork` بدلاً من الاسم الافتراضي (a.out).

How to run the code after using gcc:

`./editFork`

```
wajdabdal-hadee@wajdabdal-hadee:~$ ./editFork
Chiled proccess created with PID: 27710
Chiled proccess created with PID: 27713
```

c. Besides your running terminal (main terminal), open two other terminals. Trace the execution of your main using one of the other terminals. Then, send a STOP signal to the original terminal using the third terminal. Make sure to capture that signal in the tracing terminal.

(c) Trace execution of the main terminal and send a STOP signal

1. Find the PID of your main terminal:

```
bash
echo $$
```

Copy Edit

2. Open a second terminal and trace execution of the main terminal:

```
bash
strace -p <Main_Terminal_PID>
```

Copy Edit

3. Open a third terminal and send a STOP signal to the main terminal:

```
bash
kill -STOP <Main_Terminal_PID>
```

Copy Edit

4. Observe in the tracing terminal that the process is stopped.

`sudo strace -p <PID>`

```
wajdabdal-hadee@wajdabdal-hadee:~  
wajdabdal-hadee@wajdabdal-hadee:~$ sudo strace -p 27725  
strace: Process 27725 attached  
pselect6(1, [0], NULL, NULL, NULL, {sigmask=[], sigsetsize=8}) = 1 (in [0])  
read(0, "e", 1) = 1  
pselect6(1, [0], NULL, [0], {tv_sec=0, tv_nsec=0}, {sigmask=NULL, sigsetsize=8}) = 0 (T  
(timeout)  
write(2, "e", 1) = 1  
pselect6(1, [0], NULL, NULL, NULL, {sigmask=[], sigsetsize=8}) = 1 (in [0])  
read(0, "c", 1) = 1  
pselect6(1, [0], NULL, [0], {tv_sec=0, tv_nsec=0}, {sigmask=NULL, sigsetsize=8}) = 0 (T  
  
wajdabdal-hadee@wajdabdal-hadee:~  
wajdabdal-hadee@wajdabdal-hadee:~$ echo $$  
27725  
wajdabdal-hadee@wajdabdal-hadee:~$ echo "hi"  
hi  
wajdabdal-hadee@wajdabdal-hadee:~$  
  
wajdabdal-hadee@wajdabdal-hadee:~  
wajdabdal-hadee@wajdabdal-hadee:~$ echo $$  
27749  
wajdabdal-hadee@wajdabdal-hadee:~$
```

STOP the terminal by other:

`sudo kill -STOP <PID>`

```
wajdabdal-hadee@wajdabdal-hadee:~  
wajdabdal-hadee@wajdabdal-hadee:~$ sudo strace -p 27725  
strace: Process 27725 attached  
pselect6(1, [0], NULL, NULL, NULL, {sigmask=[], sigsetsize=8}) = 1 (in [0])  
read(0, "e", 1) = 1  
pselect6(1, [0], NULL, [0], {tv_sec=0, tv_nsec=0}, {sigmask=NULL, sigsetsize=8}) = 0 (T  
(timeout)  
write(2, "e", 1) = 1  
pselect6(1, [0], NULL, NULL, NULL, {sigmask=[], sigsetsize=8}) = 1 (in [0])  
read(0, "c", 1) = 1  
pselect6(1, [0], NULL, [0], {tv_sec=0, tv_nsec=0}, {sigmask=NULL, sigsetsize=8}) = 0 (T  
  
wajdabdal-hadee@wajdabdal-hadee:~  
wajdabdal-hadee@wajdabdal-hadee:~$ echo $$  
27725  
wajdabdal-hadee@wajdabdal-hadee:~$ echo "hi"  
hi  
wajdabdal-hadee@wajdabdal-hadee:~$  
  
wajdabdal-hadee@wajdabdal-hadee:~  
wajdabdal-hadee@wajdabdal-hadee:~$ echo $$  
27749  
wajdabdal-hadee@wajdabdal-hadee:~$ sudo kill -STOP 27725  
[sudo] password for wajdabdal-hadee:  
wajdabdal-hadee@wajdabdal-hadee:~$
```

d. Send a CONT signal to the original terminal.

`kill -CONT <PID>`

```
wajdabdal-hadee@wajdabdal-hadee:~$ sudo strace -p 27725
strace: Process 27725 attached
pselect6(1, [0], NULL, NULL, NULL, {sigmask=[], sigsetsize=8}) = 1 (in [0])
read(0, "e", 1) = 1
pselect6(1, [0], NULL, [0], {tv_sec=0, tv_nsec=0}, {signmask=NULL, sigsetsize=8}) = 0
(ineout)
write(2, "e", 1) = 1
pselect6(1, [0], NULL, NULL, NULL, {sigmask=[], sigsetsize=8}) = 1 (in [0])
read(0, "c", 1) = 1
pselect6(1, [0], NULL, [0], {tv_sec=0, tv_nsec=0}, {signmask=NULL, sigsetsize=8}) = 0

wajdabdal-hadee@wajdabdal-hadee:~$ echo $$
27725
wajdabdal-hadee@wajdabdal-hadee:~$ echo "hi"
hi
wajdabdal-hadee@wajdabdal-hadee:~$ dj

27749
wajdabdal-hadee@wajdabdal-hadee:~$ sudo kill -STOP 27725
[sudo] password for wajdabdal-hadee:
wajdabdal-hadee@wajdabdal-hadee:~$ sudo kill -CONT 27725
wajdabdal-hadee@wajdabdal-hadee:~$
```

e. send a KILL signal to the other two terminals using the original terminal. Show all your steps in details.

(e) Kill the other two terminals from the original terminal

1. Find the PIDs of the other two terminals:

```
bash
ps aux | grep terminal
```

(Look for the processes corresponding to the opened terminals)

2. Kill both terminals:

```
bash
kill -9 <Terminal1_PID> <Terminal2_PID>
```

or

```
bash
pkill -9 -f terminal
```

sudo kill -KILL <PID>

```
wajdabdal-hadee@wajdabdal-hadee: ~  
wajdabdal-hadee@wajdabdal-hadee:~$ sudo kill -KILL 27725  
wajdabdal-hadee@wajdabdal-hadee:~$ sudo kill -KILL 27749  
wajdabdal-hadee@wajdabdal-hadee:~$
```

f. What is the difference between kill and terminate signals?

(f) Difference between `kill` and `terminate` signals

Signal	Description
<code>SIGTERM</code> (15)	Gracefully terminates a process, allowing it to clean up resources.
<code>SIGKILL</code> (9)	Immediately forces a process to stop, without allowing cleanup.

- `kill -15 <PID>` sends `SIGTERM`, allowing the process to terminate properly.
- `kill -9 <PID>` sends `SIGKILL`, forcing termination without cleanup.

- `kill` can send any signal, including `SIGKILL`, which immediately terminates the process.
- `terminate` means sending `SIGTERM`, a signal that gracefully terminates the process.

g. Write a bash shell script that gets a process PID as an argument from the command line. Then, your script should find the number of threads this process is running using the information provided under `/proc` directory.

```
numThr.sh  
1#!/bin/bash  
2  
3if [ -z "$1" ] ; then  
4    echo "Plz enter the PID!"  
5    exit 1  
6fi  
7  
8thrCount=`cat /proc/$1/status | grep Threads | awk '{print $2}'`  
9echo "The number of Threads for the PID $1: $thrCount"  
10
```

FOR checking:


```
wajdabdal-hadee@wajdabdal-hadee:/proc/27852$ cat status | grep "Threads"
Threads: 87
wajdabdal-hadee@wajdabdal-hadee:/proc/27852$ cd
wajdabdal-hadee@wajdabdal-hadee:~$ bash numThr.sh 27852
The number of Threads for the PID 27852: 87
```

3.8 Processes Communication with Socket

- From moodle, download the files client.c and server.c.
- Briefly explain both codes.

1. كود الخادم (server.c):

- يقوم بإنشاء **Socket** باستخدام `()socket`.
- يربطه بعنوان IP محلي باستخدام `()bind`.
- يستمع للطلبات الواردة من العملاء عبر `()listen`.
- يقبل اتصالاً وارداً باستخدام `()accept`.
- يقرأ رسالة من العميل باستخدام `()read`.
- يطبع الرسالة المستقبلية على الشاشة.
- يغلق الاتصال مع العميل ويظل في حلقة لا نهائية لقبول الاتصالات الجديدة.

2. كود العميل (client.c):

- يحدد عنوان الـ IP للخادم (إما محدد في الكود أو يتم قراءته من المستخدم).
- يحصل على عنوان الخادم باستخدام `()gethostbyname`.
- ينشئ **Socket** باستخدام `()socket`.
- يتصل بالخادم باستخدام `()connect`.
- يرسل رسالة "hello world" للخادم باستخدام `()write`.
- يغلق الاتصال بعد إرسال الرسالة.

- Compile the server file.

gcc server.c -o server

```
wajdabdal-hadee@wajdabdal-hadee:~/Desktop$ gcc server.c -o server
server.c: In function 'main':
server.c:29:6: warning: implicit declaration of function 'exit' [-Wimplicit-function-declaration]
   29 |     exit(1);
      |     ^~~~~
server.c:7:1: note: include '<stdlib.h>' or provide a declaration of 'exit'
    6 | #include <arpa/inet.h>
      | ~~~~~
+++ |+#include <stdlib.h>
    7 |
```

d. Compile the client file.

gcc client.c -o client

```
wajdabdal-hadee@wajdabdal-hadee:~/Desktop$ gcc client.c -o client
client.c: In function 'main':
client.c:32:6: warning: implicit declaration of function 'exit' [-Wimplicit-function-declaration]
   32 |     exit(1);
      |     ^~~~~
client.c:9:1: note: include '<stdlib.h>' or provide a declaration of 'exit'
    8 | #include <netdb.h>
      | ~~~~~
+++ |+#include <stdlib.h>
    9 |
```

e. Run the server file.

./server

```
wajdabdal-hadee@wajdabdal-hadee:~/Desktop$ ./server
```

f. Run the client file.

./client localhost

```
wajdabdal-hadee@wajdabdal-hadee:~/Desktop$ ./client localhost
Hellooooo
wajdabdal-hadee@wajdabdal-hadee:~/Desktop$ ./client localhost
Hellooooo
wajdabdal-hadee@wajdabdal-hadee:~/Desktop$
```

g. Modify the client code to send its process PID to the server to be printed with the hello message.

on the client:

```
int pid = getpid();
char message[100];
sprintf(message, "\nPID: %d - Hello World", pid);
write(sockfd, message, strlen(message) + 1);
```

```
/* write a message to the server */
fprintf(stderr, "Helloooo");
write(sockfd, "hello world22222", sizeof("hello world"));

int pid = getpid();
char message[256];
sprintf(message, "\nPID: %d - Hello World", pid);
write(sockfd, message, strlen(message) + 1);

close(sockfd);

return 0;
```

on the server:

```
printf("Received from client: %s\n", string);
```

```
/* read a message from the client */
len = read(newsockfd, string, MAX_SIZE);
/* make sure it's a proper string */
string[len] = 0;
printf("%s\n", string);
printf("Received from Client: %s\n", string);
close(newsockfd);
```

The Result:

```
wajdabdal-hadee@wajdabdal-hadee: ~/Desktop
wajdabdal-hadee@wajdabdal-hadee:~/Desktop$ ./server
hello world2
PID: 30090 - Hello World
Received from Client: hello world2
PID: 30090 - Hello World

wajdabdal-hadee@wajdabdal-hadee:~/Desktop
wajdabdal-hadee@wajdabdal-hadee:~/Desktop$ ./client localhost
Helloooo
```

h. What interprocess communication technique is being used?

The technique used here is Sockets over TCP/IP, which is one of the most common Inter-Process Communication (IPC) methods for communication between processes on different machines or within the same machine.

ج. تحديد تقنية الاتصال بين العمليات (IPC) المستخدمة
التقنية المستخدمة هنا هي Sockets عبر TCP/IP، والتي تعتبر أحد أشهر طرق Inter-Process Communication (IPC) بين الأجهزة المختلفة أو حتى العمليات داخل نفس الجهاز.

♦ ما هو معنى "Interprocess"؟

Interprocess Communication (IPC) أو التواصل بين العمليات هو مصطلح يشير إلى الطرق أو التقنيات التي تُستخدم لتبادل البيانات والمعلومات بين عمليتين أو أكثر تعملان على نفس النظام أو جهاز الكمبيوتر.

- عملية (Process) هي برنامج يعمل في الذاكرة ويكون له معرف فريد (PID).
- التواصل بين العمليات (IPC) يعني كيفية نقل البيانات بين عمليات مختلفة تعمل في نفس الوقت.

i. Use (Ctrl+Z) to stop the server process.

Ctrl+Z or Ctrl+c

j. Try to run the server again. What did you get? Why

```
[1]+  Stopped                  ./server
wajdabdal-hadee@wajdabdal-hadee:~/Desktop$ ./server
can't bind local address: Address already in use
```

Reason: The port is still in use. This can be resolved by forcing the system to release the port.

قد تظهر رسالة خطأ بسبب أن المنفذ لا يزال مشغولاً:
can't bind local address
السبب: أن المنفذ لم يُحرر بعد، ويمكن حل هذه المشكلة بإجبار النظام على تحرير المنفذ باستخدام الأمر:

بتقدر انك تقتل العملية عن طريق هاد الكوماند

```
wajdabdal-hadee@wajdabdal-hadee:~/Desktop$ pgrep server
30088
wajdabdal-hadee@wajdabdal-hadee:~/Desktop$ sudo kill -9 30088
[1]+  Killed                  ./server
```

3.9 To Do (مش جاهز)

Modify the code you have used in the previous section to work as follows:

a. The client process sends its PID to the server process.

(Already implemented in Section 3.8).

b. The server process sends its PID back to the client process.

on the server:

```
int server_pid = getpid();
char response[100];
sprintf(response, "Server PID: %d", server_pid);
write(newsockfd, response, strlen(response) + 1);
```

```
/* Retrieve the server's PID using getpid() and send it to the client */
int serv_pid = getpid();
char respon[256];
sprintf(respon, "Server PID: %d", serv_pid);
write(newsockfd, respon, strlen(respon) + 1);
close(newsockfd);
```

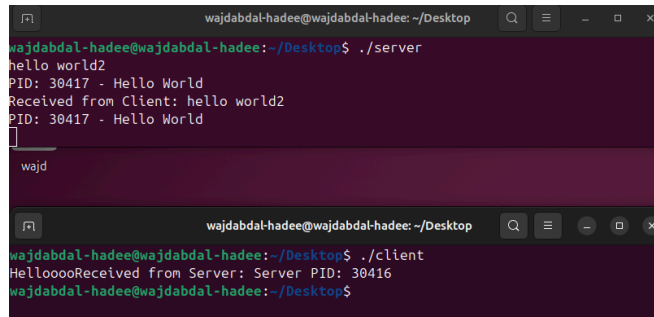
on the client:

```
char response[100];
read(sockfd, response, sizeof(response));
printf("Received from server: %s\n", response);
```

```
/* Received from Server the message */
char respon[256];
read(sockfd, respon, sizeof(respon));
printf("Received from Server: %s\n", respon);
close(sockfd);

return 0;
```

The Result:



```
wajdabdal-hadee@wajdabdal-hadee: ~/Desktop
wajdabdal-hadee@wajdabdal-hadee:~/Desktop$ ./server
hello world2
PID: 30417 - Hello World
Received from Client: hello world2
PID: 30417 - Hello World

wajd
wajdabdal-hadee@wajdabdal-hadee: ~/Desktop
wajdabdal-hadee@wajdabdal-hadee:~/Desktop$ ./client
HelloooooReceived from Server: Server PID: 30416
wajdabdal-hadee@wajdabdal-hadee:~/Desktop$
```

c. The client process sends USR1 and USR2 signals to the server process.

on the server:

```
#include <signal.h>

void handle_sigusr1(int sig) {
printf("Hello, I received SIG1 from the client.\n");
}

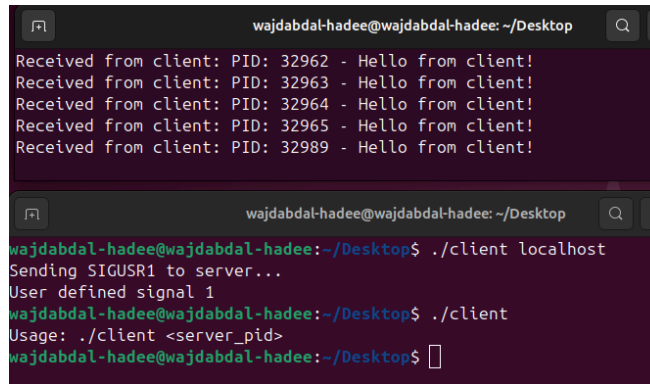
void handle_sigusr2(int sig) {
printf("Hello, I received SIG2 from the client.\n");
}

int main() {
signal(SIGUSR1, handle_sigusr1);
signal(SIGUSR2, handle_sigusr2);
}
```

on the client:

```
kill(server_pid, SIGUSR1);
sleep(1);
kill(server_pid, SIGUSR2);
```

the Result before kill any signal:



The image shows a terminal window with two panes. The top pane displays five lines of output from a server process: "Received from client: PID: 32962 - Hello from client!", "Received from client: PID: 32963 - Hello from client!", "Received from client: PID: 32964 - Hello from client!", "Received from client: PID: 32965 - Hello from client!", and "Received from client: PID: 32989 - Hello from client!". The bottom pane shows a shell prompt "wajdabdal-hadee@wajdabdal-hadee: ~/Desktop\$". The first command entered is "./client localhost", which results in "Sending SIGUSR1 to server..." and "User defined signal 1". The second command is "./client", which shows "Usage: ./client <server_pid>". The prompt is then followed by a cursor.

```
wajdabdal-hadee@wajdabdal-hadee: ~/Desktop
Received from client: PID: 32962 - Hello from client!
Received from client: PID: 32963 - Hello from client!
Received from client: PID: 32964 - Hello from client!
Received from client: PID: 32965 - Hello from client!
Received from client: PID: 32989 - Hello from client!

wajdabdal-hadee@wajdabdal-hadee: ~/Desktop
wajdabdal-hadee@wajdabdal-hadee:~/Desktop$ ./client localhost
Sending SIGUSR1 to server...
User defined signal 1
wajdabdal-hadee@wajdabdal-hadee:~/Desktop$ ./client
Usage: ./client <server_pid>
wajdabdal-hadee@wajdabdal-hadee:~/Desktop$
```

- d. When the server process receives USR1 it prints the statement Hello, I received SIG1 from the client..
- e. When the server process receives USR2, it prints the statement Hello, I received SIG2 from the client..
- f. The client process sends kill signal to the server process.
- g. The client process terminates.