



Yaman_199273_C2

Haneen_197899_C2

Lubna_175170_C1

Dr. Majeda Asaad

السؤال الأول:

► أنشئ مجلدين باسم one, two ضمن مجلدك الخاص.



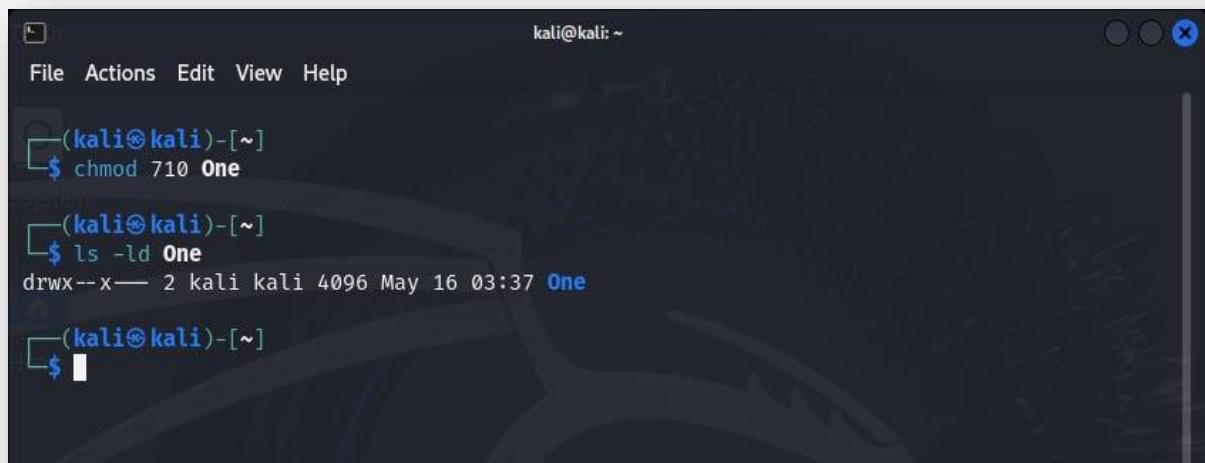
```
kali㉿kali:[~]
$ mkdir One Two

(kali㉿kali)-[~]
$ ls
Desktop Documents Downloads Music One Pictures Public Templates Two Videos

(kali㉿kali)-[~]
$
```

► عدل سماحية المجلد one قراءة وكتابة وتنفيذ للملك، وتنفيذ للمجموعة ولا

شيء للبلق .



```
kali㉿kali:[~]
$ chmod 710 One

(kali㉿kali)-[~]
$ ls -ld One
drwx--x-- 2 kali kali 4096 May 16 03:37 One

(kali㉿kali)-[~]
$
```

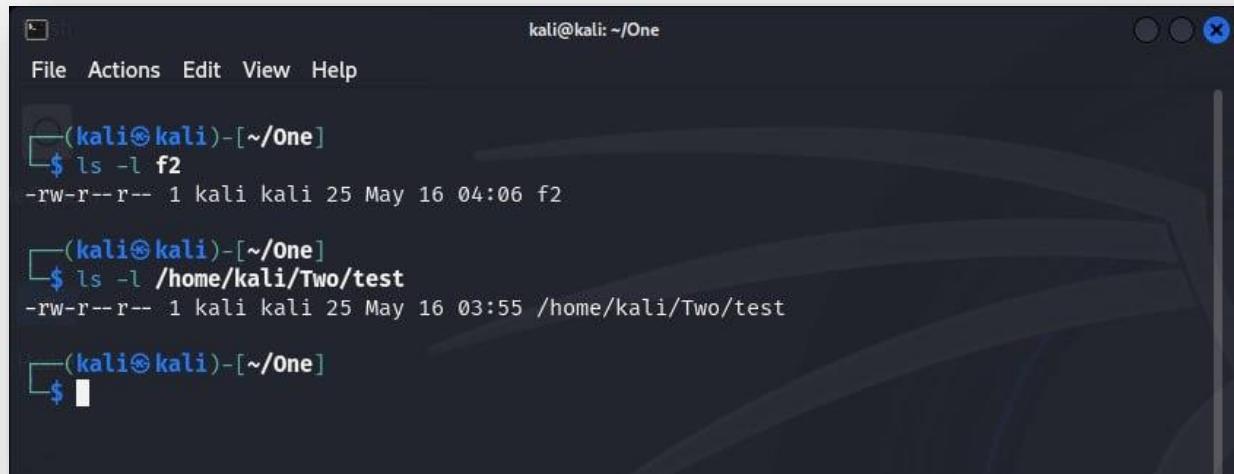
► أنشئ ملف test ضمن مجلد Two واتب بداخله اسمك ورقمك الجامعي.

```
kali㉿kali:[~] $ cd Two
kali㉿kali:[~/Two] $ cat > test
Yaman Hamed
Yaman_199273
kali㉿kali:[~/Two] $ ls
test
kali㉿kali:[~/Two] $ cat test
Yaman Hamed
Yaman_199273
kali㉿kali:[~/Two]
```

► انسخ الملف test إلى المجلد one مع تغيير الاسم إلى f2 .

```
kali㉿kali:[~] $ cp test /home/kali/One/f2
kali㉿kali:[~/Two] $ cd /home/kali/One
kali㉿kali:[~/One] $ ls
f2
kali㉿kali:[~/One] $ cat f2
Yaman Hamed
Yaman_199273
kali㉿kali:[~/One]
```

► أظهر time stamp لكلا الملفين.



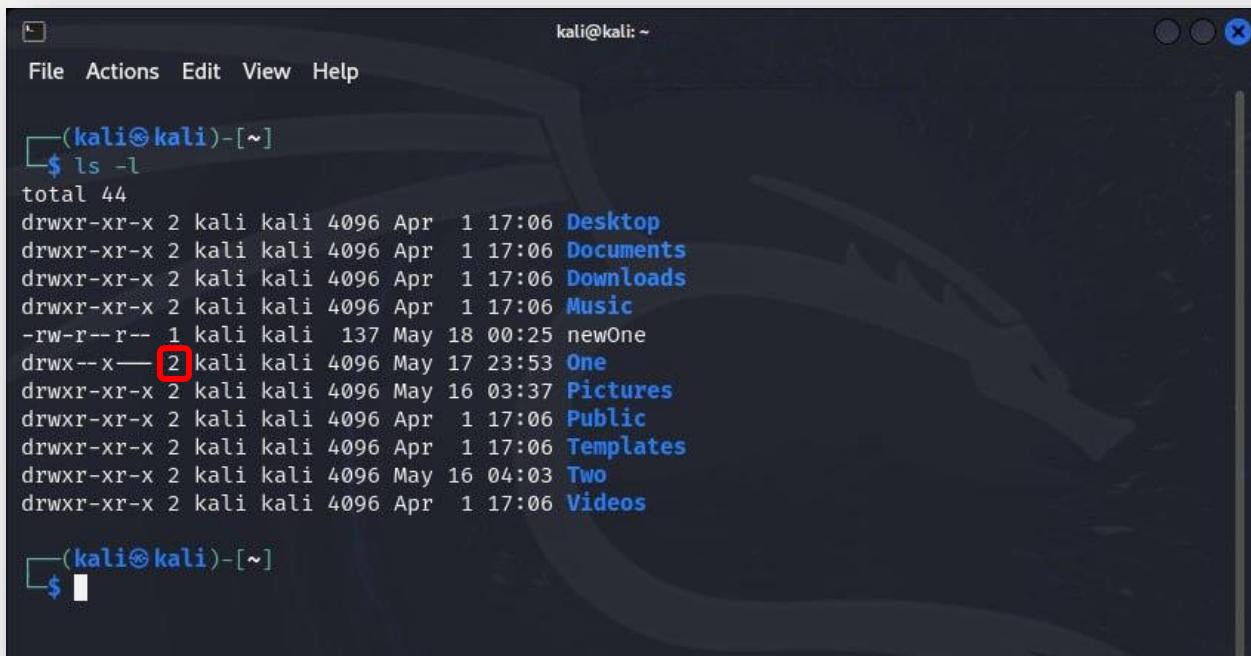
```
kali@kali: ~/One
File Actions Edit View Help

[(kali㉿kali)-[~/One]
$ ls -l f2
-rw-r--r-- 1 kali kali 25 May 16 04:06 f2

[(kali㉿kali)-[~/One]
$ ls -l /home/kali/Two/test
-rw-r--r-- 1 kali kali 25 May 16 03:55 /home/kali/Two/test

[(kali㉿kali)-[~/One]
$
```

► أظهر عدد الارتباطات . one

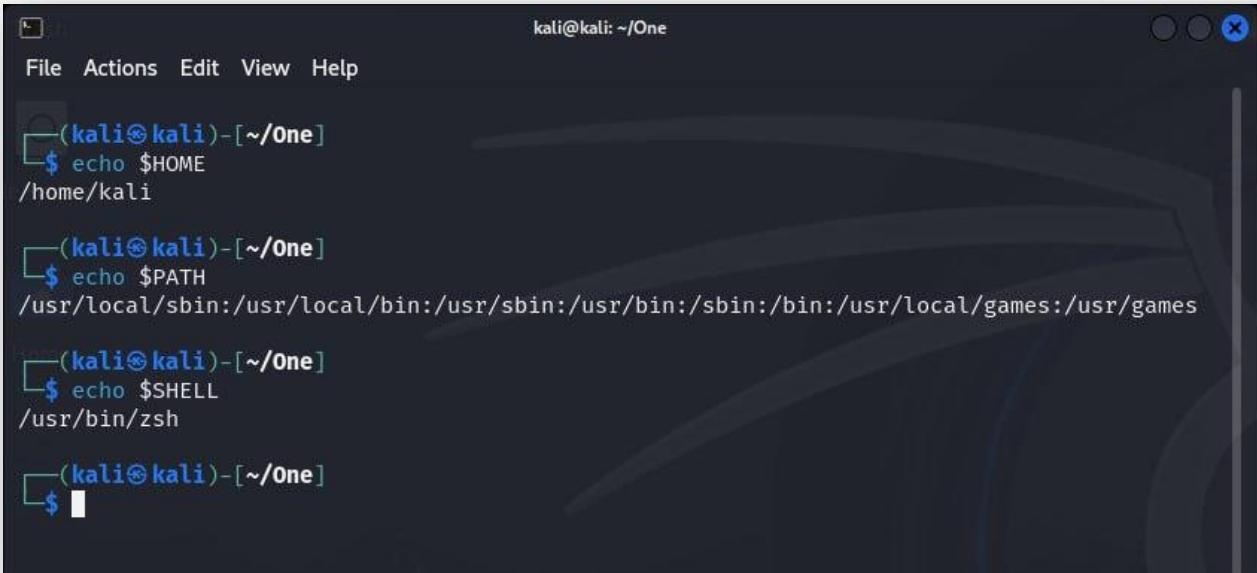


```
kali@kali: ~
File Actions Edit View Help

[(kali㉿kali)-[~]
$ ls -l
total 44
drwxr-xr-x 2 kali kali 4096 Apr  1 17:06 Desktop
drwxr-xr-x 2 kali kali 4096 Apr  1 17:06 Documents
drwxr-xr-x 2 kali kali 4096 Apr  1 17:06 Downloads
drwxr-xr-x 2 kali kali 4096 Apr  1 17:06 Music
-rw-r--r-- 1 kali kali 137 May 18 00:25 newOne
drwx--x--- 2 kali kali 4096 May 17 23:53 One
drwxr-xr-x 2 kali kali 4096 May 16 03:37 Pictures
drwxr-xr-x 2 kali kali 4096 Apr  1 17:06 Public
drwxr-xr-x 2 kali kali 4096 Apr  1 17:06 Templates
drwxr-xr-x 2 kali kali 4096 May 16 04:03 Two
drwxr-xr-x 2 kali kali 4096 Apr  1 17:06 Videos

[(kali㉿kali)-[~]
$
```

► أظهر قيمة متغيرات البيئة . PATH, HOME, SHELL



A screenshot of a terminal window titled "kali@kali: ~/One". The window shows the following command outputs:

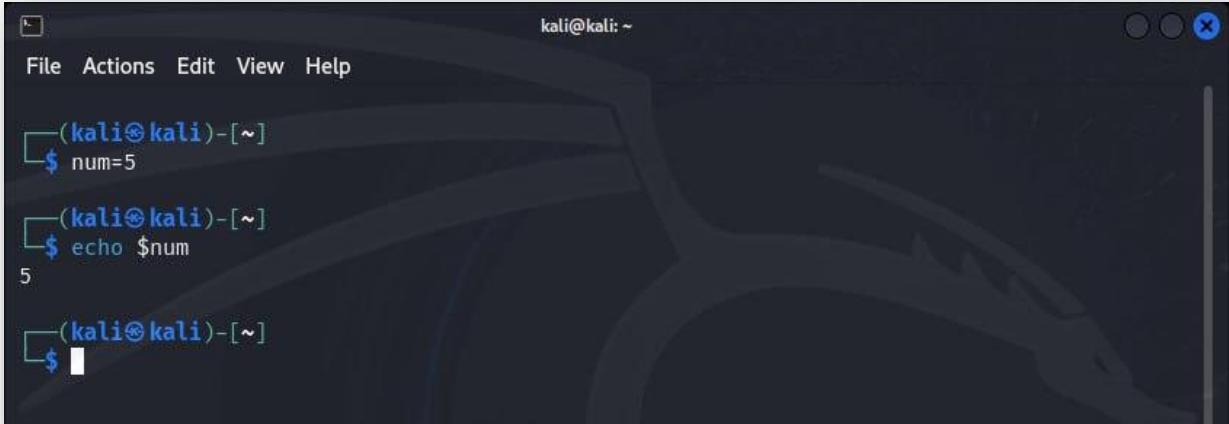
```
(kali㉿kali)-[~/One]
$ echo $HOME
/home/kali

(kali㉿kali)-[~/One]
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/local/games:/usr/games

(kali㉿kali)-[~/One]
$ echo $SHELL
/usr/bin/zsh

(kali㉿kali)-[~/One]
$
```

► عَزَّفَ مُتَحَولٌ جَدِيدٌ يُعْنِي num فِي shell وَأَدْخَلَ قِيمَةً 5 لِلمُتَحَول.



A screenshot of a terminal window titled "kali@kali: ~". The window shows the following command outputs:

```
(kali㉿kali)-[~]
$ num=5

(kali㉿kali)-[~]
$ echo $num
5

(kali㉿kali)-[~]
$
```

► اكتب برنامج بلغة shell يقوم بإدخال عدد وإظهار رسالة العدد زوجي أم فردي.

```
kali㉿kali: ~
File Actions Edit View Help
[(kali㉿kali)-[~]
$ touch newOne

[(kali㉿kali)-[~]
$ nano newOne
```

```
kali㉿kali: ~
File Actions Edit View Help
GNU nano 7.2                               newOne
echo "Enter any number:"
read number
if [ `expr $number % 2` -ne 0 ]; then
    echo "$number is odd"
else
    echo "$number is even"
fi

[ Read 8 lines ]
^G Help      ^O Write Out   ^W Where Is   ^K Cut        ^T Execute   ^C Location
^X Exit     ^R Read File   ^\ Replace    ^U Paste      ^J Justify   ^/ Go To Line
```

```
File Actions Edit View Help
[(kali㉿kali)-[~]
$ touch newOne

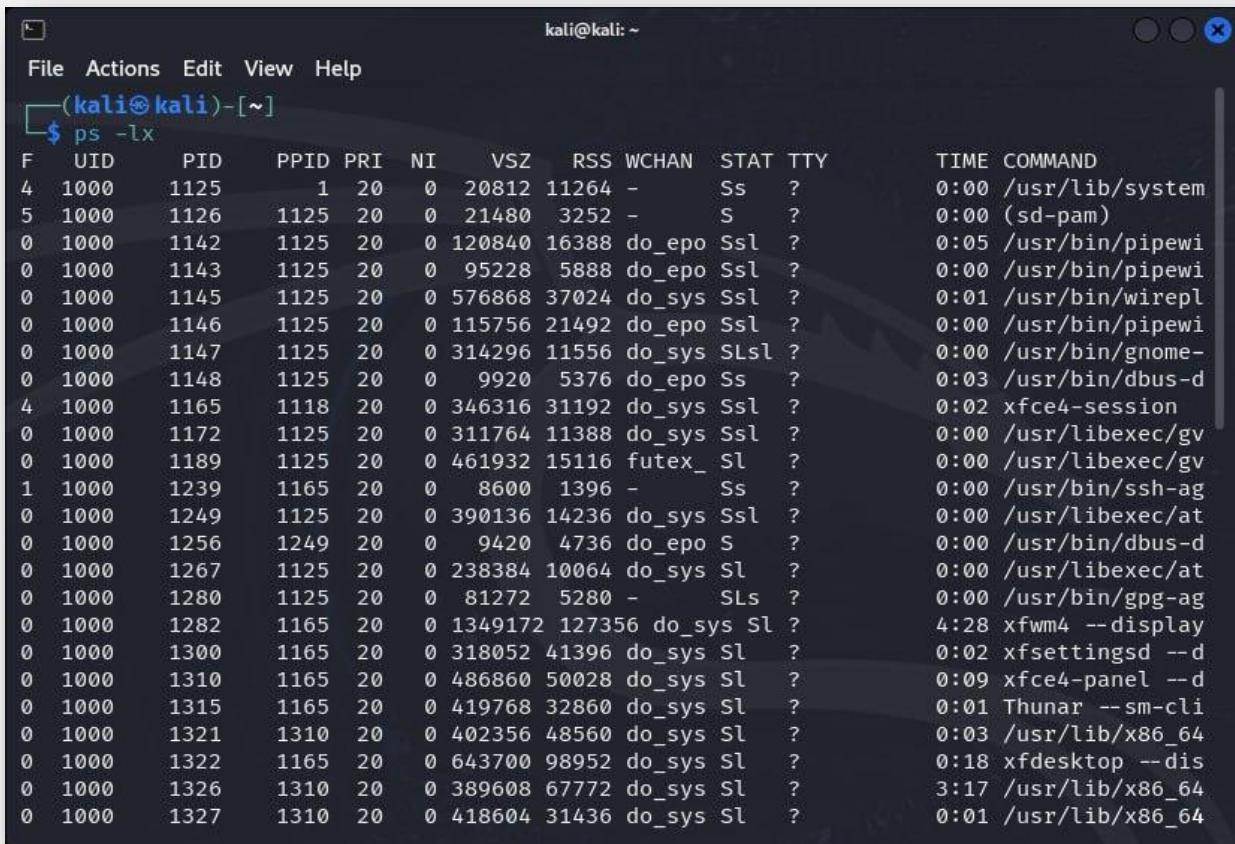
[(kali㉿kali)-[~]
$ nano newOne

[(kali㉿kali)-[~]
$ sh newOne
Enter any number:
7
7 is odd

[(kali㉿kali)-[~]
$ sh newOne
Enter any number:
952
952 is even

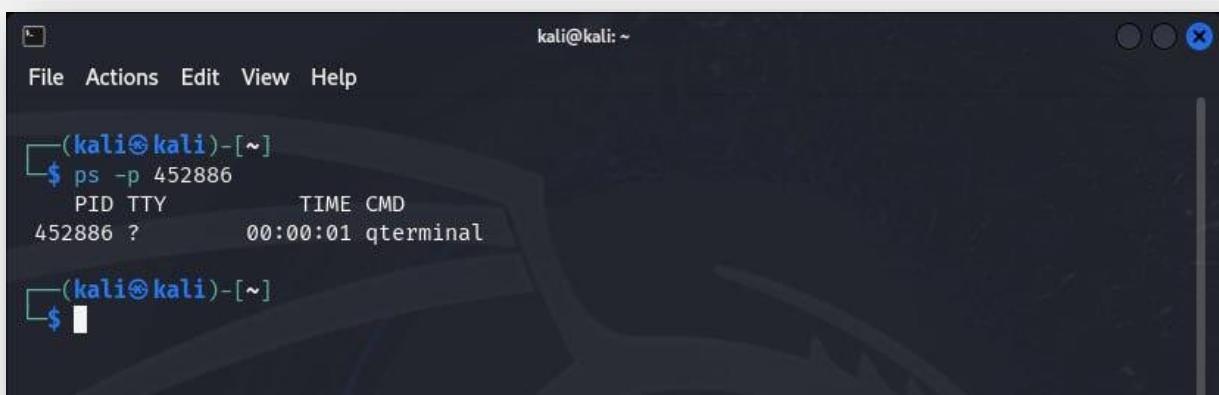
[(kali㉿kali)-[~]
$
```

► كيف يمكن الحصول على تفاصيل جميع الإجرائيات العاملة في النظام الآن.



```
kali㉿kali:[~]
$ ps -lx
F  UID      PID  PPID PRI  NI    VSZ   RSS WCHAN STAT TTY          TIME COMMAND
4 1000     1125     1  20   0 20812 11264 -      Ss    ?        0:00 /usr/lib/system
5 1000     1126   1125  20   0 21480 3252 -      S    ?        0:00 (sd-pam)
0 1000     1142   1125  20   0 120840 16388 do_epo Ssl  ?        0:05 /usr/bin/pipewi
0 1000     1143   1125  20   0 95228 5888 do_epo Ssl  ?        0:00 /usr/bin/pipewi
0 1000     1145   1125  20   0 576868 37024 do_sys Ssl  ?        0:01 /usr/bin/wirepl
0 1000     1146   1125  20   0 115756 21492 do_epo Ssl  ?        0:00 /usr/bin/pipewi
0 1000     1147   1125  20   0 314296 11556 do_sys SLs  ?        0:00 /usr/bin/gnome-
0 1000     1148   1125  20   0 9920  5376 do_epo Ss  ?        0:03 /usr/bin/dbus-d
4 1000     1165   1118  20   0 346316 31192 do_sys Ssl  ?        0:02 xfce4-session
0 1000     1172   1125  20   0 311764 11388 do_sys Ssl  ?        0:00 /usr/libexec/gv
0 1000     1189   1125  20   0 461932 15116 futex_ SL  ?        0:00 /usr/libexec/gv
1 1000     1239   1165  20   0 8600  1396 -      Ss  ?        0:00 /usr/bin/ssh-ag
0 1000     1249   1125  20   0 390136 14236 do_sys Ssl  ?        0:00 /usr/libexec/at
0 1000     1256   1249  20   0 9420  4736 do_epo S  ?        0:00 /usr/bin/dbus-d
0 1000     1267   1125  20   0 238384 10064 do_sys SL  ?        0:00 /usr/libexec/at
0 1000     1280   1125  20   0 81272  5280 -      SLs  ?        0:00 /usr/bin/gpg-ag
0 1000     1282   1165  20   0 1349172 127356 do_sys SL  ?        4:28 xfwm4 --display
0 1000     1300   1165  20   0 318052 41396 do_sys SL  ?        0:02 xfsettingsd --d
0 1000     1310   1165  20   0 486860 50028 do_sys SL  ?        0:09 xfce4-panel --d
0 1000     1315   1165  20   0 419768 32860 do_sys SL  ?        0:01 Thunar --sm-cli
0 1000     1321   1310  20   0 402356 48560 do_sys SL  ?        0:03 /usr/lib/x86_64
0 1000     1322   1165  20   0 643700 98952 do_sys SL  ?        0:18 xfdesktop --dis
0 1000     1326   1310  20   0 389608 67772 do_sys SL  ?        3:17 /usr/lib/x86_64
0 1000     1327   1310  20   0 418604 31436 do_sys SL  ?        0:01 /usr/lib/x86_64
```

► كيف يمكن البحث عن العمليات بواسطة PID .



```
kali㉿kali:[~]
$ ps -p 452886
 PID TTY          TIME CMD
452886 ?        00:00:01 qterminal

(kali㉿kali:[~]
$
```

السؤال الثاني:

اكتب برنامج يقوم بما يلي:

► تعريف متتحول x وتعيين قيمة له ولتكن 100 :

► إنشاء إجرائية ابن:

```
File Edit Search View Document Help
File Actions Edit View Help
(kali㉿kali)-[~/Desktop]
$ cd Desktop
(kali㉿kali)-[~/Desktop]
$ gcc Ques2.c -o Ques2
(kali㉿kali)-[~/Desktop]
$ ./Ques2
- Child x before change value = 100
- Child x after change value = 40
- Parent x before change value = 100
- Parent x after change value = 50
```

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/types.h>
4 #include <wait.h>
5 #include <unistd.h>
6
7 int main() {
8     int x = 100;
9     pid_t PID = fork();
10    if (PID == 0) {
11        printf("- Child x before change value = %d\n", x);
12        x = 40;
13        printf("- Child x after change value = %d\n", x);
14    }
15    else if (PID == -1) {
16        printf("Fork Failed!");
17        return -1;
18    }
19    else {
20        wait(NULL);
21        printf("_____\n");
22        printf("- Parent x before change value = %d\n", x);
23        x = 50;
24        printf("- Parent x after change value = %d\n", x);
25    }
26    return 0;
27 }
28 }
```

► شرح الكود:

. - مكتبة الادخال والإخراج القياسية. **1** *<stdio.h> Standard Input/Output*

. - مكتبة للاستخدامات العامة تحوي عدة توابع خاصة. **2** *<stdlib.h> Standard Library*

بحجز الذاكرة وتتابع أخرى للقيام بعمليات معينة ورياضية أيضاً.

. - مكتبة تحوي العديد من التوابع **3** *<sys/types.h> Typedef Symbols And Structures*

الخاصة بالإجرائيات والنياسب وبعض التوابع الأخرى.

4 - مكتبة تحوي العديد من التوابع للتعامل مع أبناء الإجرائيات.

5 - مكتبة تحتوي الكثير من التوابع للتعامل مع واجهات أنظمة التشغيل.

يقوم البرنامج بداية بتعريف متغير x صحيح وإسناد قيمة 100 له.

بعدها يقوم بتعريف متغير PID من نوع pid_t وهو نوع يتعامل مع معرفات الإجرائيات العائدة من بعض التوابع مثل `fork()`, `wait()`.

هنا في المثال تم استدعاء التابع `fork` وهوتابع يقوم بإنشاء نسخة طبق الأصل من الإجرائية فيصبح لدينا إجرائيتين متطابقتين بينهما فرق رئيسي وهو رقم الإجرائية الخاص PID.

بعد الاستدعاء مباشرة تبدأ العمليتين بالتنفيذ الابن والأب على التوازي.

في حال عودة القيمة 0 من التابع هذا يعني أنها ضمن الإجرائية الابن عندها يتم طباعة قيمة x الحالية وبعدها يتم تغيير قيمة x إلى 40 ثم يتم طباعة القيمة الجديدة.

أما في حال عودة معرف الابن أي PID الخاص بالابن هذا يعني أنها ضمن الأب وبالتالي يتم تنفيذ التعليمة `(null) wait` والتي تجعل الإجرائية الأب تنتظر تنفيذ الإجرائية الابن وانتهائها.

بعدها يتم طباعة قيمة x الحالية وتعديل قيمته ضمن الأب إلى 50 ثم يتم طباعة القيمة الجديدة.

► ما هي قيمة المتغير في الإجرائية الابن.

قيمتها 100 قبل التعديل أما بعد التعديل تصبح 40.

► ماذا يحدث للمتغير عندما تقوم كل من الإجرائية الابن والأب بتغيير قيمة x .

تصبح قيمتها في الإجرائية الابن 40 أما في الأب 50.

► ما هي ملاحظاتك على البرنامج وعلاقة الإجرائية الابن والأم والمتحول.

ملاحظاتي أن المتغير له قيم مستقلة في كل من الإجرائيتين.

قيمة 100 ابتدائية مستقلة ونهاية 50 و 40 مستقلتين.

أما العلاقة بين الإجرائية الابن والأب والمتحول فإن كل من الابن والأب له قيم مختلفة عن الآخر.

السؤال الثالث:

```
File Edit Search View Document Help
X

```
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <stdlib.h>
4 #include <sys/wait.h>
5 #include <string.h>
6
7 #define BUFFER_SIZE 30
8
9 void displayError(const char *message) {
10 perror(message);
11 exit(EXIT_FAILURE);
12 }
13
14 int main() {
15 int pipe1[2]; // Pipe from parent to child 1
16 int pipe2[2]; // Pipe from child 1 to child 2
17 int pipe3[2]; // Pipe from parent to child 2
18
19 if (pipe(pipe1) == -1 || pipe(pipe2) == -1 || pipe(pipe3) == -1) {
20 displayError("pipe");
21 }
22
23 pid_t child1 = fork();
24 if (child1 == -1) {
25 displayError("fork");
26 } else if (child1 == 0) {
27 close(pipe1[1]);
28 close(pipe2[0]);
29 close(pipe3[0]);
30 close(pipe3[1]);
31
32 char messageToChild2[BUFFER_SIZE] = "Hello from Child 1";
33 write(pipe2[1], messageToChild2, BUFFER_SIZE);
34
35 char messageFromParent[BUFFER_SIZE];
36 int bytesRead = read(pipe1[0], messageFromParent, BUFFER_SIZE);
37 if (bytesRead > 0) {
38 printf("Parent received from Child 1: %s\n", messageFromParent);
39 }
40
41
42 pid_t child2_pid = fork();
43 if (child2_pid == -1) {
44 handleError("fork");
45 } else if (child2_pid == 0) {
46 close(pipe3[0]);
47 close(pipe2[1]);
48 close(pipe1[0]);
49 close(pipe1[1]);
50
51 char receivedMsg[BUFFER_SIZE];
52 int bytesRead = read(pipe2[0], receivedMsg, BUFFER_SIZE);
53 if (bytesRead > 0) {
54 printf("Child 2 received: %s\n", receivedMsg);
55 }
56 close(pipe2[0]);
57
58 char msgToParent2[BUFFER_SIZE] = "Message to Parent from Child 2";
59 write(pipe3[1], msgToParent2, BUFFER_SIZE);
60 close(pipe3[1]);
61 exit(EXIT_SUCCESS);
62 }
63
64 close(pipe1[1]);
65 close(pipe2[0]);
66 close(pipe2[1]);
67 close(pipe3[1]);
68
69 waitpid(child2_pid, NULL, 0);
70
71 char child1Msg[BUFFER_SIZE];
72 char child2Msg[BUFFER_SIZE];
73
74 int readBytesChild1 = read(pipe1[0], child1Msg, BUFFER_SIZE);
75 if (readBytesChild1 > 0) {
76 printf("Parent received from Child 1: %s\n", child1Msg);
77 }
78
79 int readBytesChild2 = read(pipe3[0], child2Msg, BUFFER_SIZE);
80 if (readBytesChild2 > 0) {
81 printf("Parent received from Child 2: %s\n", child2Msg);
82 }
83
84 close(pipe1[0]);
85 close(pipe3[0]);
86
87 waitpid(child1_pid, NULL, 0);
88
89 return 0;
90 }
```



kali@kali:~



File Actions Edit View Help



-(kali@kali)-[~]



$ nano pipe.c



-(kali@kali)-[~]



$ gcc -o pipe pipe.c



-(kali@kali)-[~]



$ ./pipe



Child 2 received: Hello from Child 1  
Parent received from Child 1: Hello from Child 1  
Parent received from Child 2: Hello from Child 2



-(kali@kali)-[~]



$



kali@kali:~



File Actions Edit View Help



-(kali@kali)-[~]



$ nano pipe.c



-(kali@kali)-[~]



$ gcc -o pipe pipe.c



-(kali@kali)-[~]



$ ./pipe



Child 2 received: Hello from Child 1  
Parent received from Child 1: Message to Parent from Child 1  
Parent received from Child 2: Message to Parent from Child 2



-(kali@kali)-[~]



$



kali@kali:~



File Actions Edit View Help



-(kali@kali)-[~]



$ nano pipe.c



-(kali@kali)-[~]



$ gcc -o pipe pipe.c



-(kali@kali)-[~]



$ ./pipe



Child 2 received: Hello from Child 1  
Parent received from Child 1: Message to Parent from Child 1  
Parent received from Child 2: Message to Parent from Child 2



-(kali@kali)-[~]



$


```

```
File Actions Edit View Help
└─(kali㉿kali)-[~]
$ nano pipe.c
└─(kali㉿kali)-[~]
$ gcc -o pipe pipe.c
└─(kali㉿kali)-[~]
$ ./pipe
Child 2 received: Hello from Child 1
Parent received from Child 1: Message to Parent from Child 1
Parent received from Child 2: Message to Parent from Child 2
└─(kali㉿kali)-[~]
$
```

► شرح الكود:

الكود ينشئ العملية الأب تقوم بإنشاء عمليتين أبناء تتوصلان عبر قنوات (pipes). الهدف هو إرسال واستقبال رسائل بين العمليات.

- شرح مبدأ عمل الكود:

1. إنشاء القنوات:

- يتم إنشاء ثلاثة قنوات: واحدة من الطفل الأول للأب، واحدة من الطفل الأول للطفل الثاني، وواحدة من الطفل الثاني للأب.

2. العملية الطفل الأول (Child 1):

- ترسل رسالة إلى الطفل الثاني عبر القناة الثانية.
- ترسل رسالة إلى العملية الأب عبر القناة الأولى.
- تنهي عملها.

3. العملية الطفل الثاني (Child 2):

- تستقبل رسالة من الطفل الأول عبر القناة الثانية وتطبعها.
- ترسل رسالة إلى العملية الأب عبر القناة الثالثة.
- تنهي عملها.

4. العملية الأب (Parent):

- تنتظر انتهاء العملية الطفل الثاني.
- تستقبل رسائل من الطفل الأول عبر القناة الأولى وتطبعها.
- تستقبل رسائل من الطفل الثاني عبر القناة الثالثة وتطبعها.
- تنتظر انتهاء العملية الطفل الأولى.

5. ربط المخرج القياسي لأحد الأبناء بالمدخل القياسي للابن الآخر:

يتم إنشاء أنبوب (pipe) بين العمليتين الفرعيتين. عندما يرغب ابن الأول في إرسال بيانات إلى ابن الثاني، يكتب البيانات في المخرج القياسي للأنبوب. ابن الثاني يقرأ هذه البيانات من المدخل القياسي للأنبوب. بهذه الطريقة، يتم تمرير البيانات من ابن الأول إلى ابن الثاني عبر الأنبوب.

6. ربط المدخل القياسي لكلا الإجرائيتين ابن بالمخرج القياسي للإجرائية الأم:

يتم إنشاء أنبوبين منفصلين بين العملية الأم وكل من الأبناء. كل ابن يكتب بياناته في المدخل القياسي للأنبوب المخصص له، والذي تتصل نهايته الأخرى بالمخرج القياسي للأم. هذا يعني أن العملية الأم يمكنها قراءة البيانات التي يرسلها كل ابن من الأنبوب المخصص له. بهذه الطريقة، يتم تمرير البيانات من الأبناء إلى الأم عبر الأنابيب.

- التدفق العام:

- الطفل الأول يرسل رسالة إلى الطفل الثاني والأب.
- الطفل الثاني يستقبل رسالة من الطفل الأول ويرسل رسالة إلى الأب.
- الأب يستقبل الرسائل من كلا الطفلين وينتظر انتهاء عملياتهما الفرعية.

بهذه الطريقة، يتواصل الأطفال مع بعضهم البعض ومع العملية الرئيسية باستخدام القنوات.

السؤال الرابع:

: Main >

The screenshot shows a terminal window titled "kali@kali: ~" with several tabs open. The current tab displays a C program source code. The code handles signal 9 (SIGCHLD) and uses fork() to create two child processes: a receiver and a sender. The receiver prints the PID of the sender to its standard output. The sender prints the PID of the receiver to its standard output. Both processes then exit. Finally, the main process waits for both children to finish using waitpid().

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <sys/types.h>
5 #include <sys/wait.h>
6 #include <signal.h>
7
8 void handle_sigchld(int signo) {
9     // Handle SIGCHLD silently
10 }
11
12 int main() {
13     pid_t pid_sender, pid_receiver;
14
15     signal(SIGCHLD, handle_sigchld); // Set up SIGCHLD handler
16
17     pid_receiver = fork();
18     if (pid_receiver == 0) {
19         execl("./receiver", "receiver", NULL);
20         perror("execl");
21         exit(1);
22     }
23
24     sleep(1); // Allow receiver to initialize
25
26     pid_sender = fork();
27     if (pid_sender == 0) {
28         char pid_receiver_str[10];
29         snprintf(pid_receiver_str, sizeof(pid_receiver_str), "%d", pid_receiver);
30         execl("./sender", "sender", pid_receiver_str, NULL);
31         perror("execl");
32         exit(1);
33     }
34
35     // Wait for both child processes to finish
36     waitpid(pid_receiver, NULL, 0);
37     waitpid(pid_sender, NULL, 0);
38
39     return 0;
40 }
```

The terminal output shows the following sequence of events:

- \$ nano main.c
- \$ gcc -o main main.c
- \$./main
- Sending signal: 9
- \$./main
- Sending signal: 2
- Received and ignored signal 2
- Sending signal: 9
- \$./main
- Sending signal: 9
- \$./main
- Sending signal: 3
- Received and ignored signal 3
- Sending signal: 1
- Received and ignored signal 1
- Sending signal: 9

شرح الكود: ➤

الهدف من هذا البرنامج الرئيسي هو تشغيل برامجين الإجرائيتين المستقلتين هما receiver و sender، حيث يقوم البرنامج الرئيسي بتنظيم تتابع تشغيلهما من خلال استخدام الدالة exec() لتشغيل برماج خارجية.

شرح مبدأ عمل الكود:

١. تعريف معالج الإشارة: يقوم البرنامج بتعريف دالة `handle_sigchld` لمعالجة إشارة `SIGCHLD` بشكل صامت. هذه الإشارة تُرسل إلى العملية الرئيسية عندما تنتهي أي من العمليات الفرعية.

2. تعيين معالج الإشارة: يتم تعيين معالج الإشارة SIGCHLD ليكون الدالة .signal باستخدام الدالة ()

3. إنشاء العملية الفرعية الأولى (المستقبل - receiver):

- يتم استدعاء () fork لإنشاء عملية فرعية.

- إذا كانت القيمة المرتجلة من () fork هي 0، فهذا يعني أننا داخل العملية الفرعية.

- يتم استدعاء (exec ("./receiver", "receiver", NULL - .receiver

- إذا فشل () exec يتم طباعة رسالة خطأ باستخدام perror والخروج من باستخدام العملية الفرعية .exit(1)

4. السماح للمستقبل بالتهيئة: يقوم البرنامج الرئيسي بالنوم لمدة ثانية واحدة باستخدام

sleep (1) لـ إعطاء الوقت الكافي لـ receiver ليتهيأ.

5. إنشاء العملية الفرعية الثانية (المرسل - sender):

- يتم استدعاء () fork لإنشاء عملية فرعية ثانية.

- إذا كانت القيمة المرتجلة من () fork هي 0، فهذا يعني أننا داخل العملية الفرعية.

- يتم تحويل معرف العملية (PID) لـ receiver إلى سلسلة نصية باستخدام () .snprintf

- يتم استدعاء (exec("./sender", "sender", pid_receiver_str, NULL - لـ تشغيل برنامج sender وتمرير معرف عملية المستقبل كمعامل.

- إذا فشل () exec يتم طباعة رسالة خطأ باستخدام perror والخروج من العملية الفرعية باستخدام .exit(1)

6. انتظار انتهاء العمليات الفرعية:

- بعد إنشاء العمليتين الفرعيتين، ينتظر البرنامج الرئيسي حتى تنتهي كل منهما باستخدام .waitpid()

باختصار، يقوم البرنامج الرئيسي بإنشاء عمليتين فرعيتين لتشغيل برنامجي sender و receiver بشكل متسلسل، وينتظر انتهاءهما قبل إنتهاء التنفيذ.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <signal.h>
5 #include <time.h>
6
7 int signals[] = {SIGHUP, SIGINT, SIGQUIT, SIGKILL};
8
9 void send_signal(pid_t pid) {
10     int signal = signals[rand() % 4];
11     kill(pid, signal);
12     printf("Sending signal: %d\n", signal);
13     if (signal == SIGKILL) {
14         exit(0); // Terminate the sender process after sending SIGKILL
15     }
16 }
17
18 int main(int argc, char *argv[]) {
19     if (argc != 2) {
20         fprintf(stderr, "Usage: %s <pid>\n", argv[0]);
21         return 1;
22     }
23
24     pid_t receiver_pid = atoi(argv[1]);
25     srand(time(NULL));
26
27     while (1) {
28         send_signal(receiver_pid);
29         sleep(1);
30     }
31
32     return 0;
33 }

```

- شرح مبدأ عمل الكود:

المرسل يقوم بإرسال إشارات إلى العملية المستقبلة بشكل دوري:

1. تعریف الإشارات: يحتوي البرنامج على مجموعة من الإشارات المحددة التي يمكن إرسالها.

2. اختيار الإشارة العشوائية: يتم اختيار إشارة عشوائية من هذه المجموعة.

3. إرسال الإشارة: يتم إرسال الإشارة المختارة إلى العملية المستقبلة.

4. التصرف عند إرسال SIGKILL: إذا تم إرسال الإشارة SIGKILL، ينهي المرسل نفسه.

5. التكرار الدوري: يستمر البرنامج في إرسال الإشارات كل ثانية.

6. التعليمة kill(pid, signal) تُستخدم في الكود لإرسال إشارة معينة إلى عملية محددة. وظيفتها هي إرسال الإشارة المحددة إلى العملية المعنية بالمعرف pid.

- في الدالة send_signal(pid_t pid), يتم تحديد إشارة عشوائية من بين الإشارات التالية:

.SIGHUP, SIGINT, SIGQUIT, SIGKILL

- بعد تحديد الإشارة العشوائية، يتم استخدام الـ`kill(pid, signal)` لإرسال هذه الإشارة إلى العملية المستقبلة، والتي تم تمرير معرفها `pid` إلى المرسل عند تشغيله.

7. سياق استخدام الـ`kill` في الكود:

- إرسال إشارة: الـ`kill` ترسل إشارة محددة إلى عملية معينة. يمكن أن تكون هذه الإشارة لعلام العملية بتنفيذ إجراء معين، أو لإنهايتها كما في حالة `SIGCHLD`.

- المخرجات:

- عند تشغيل البرنامج، يبدأ المرسل في إرسال إشارات عشوائية إلى المستقبل.
- المستقبل يعالج هذه الإشارات وفقاً للمعالجات المحددة لكل إشارة.

باختصار، الـ`kill` تُستخدم لإرسال إشارات إلى العمليات، وفي هذا الكود، يتم استخدامها لإرسال إشارات من المرسل إلى المستقبل، مما يؤدي إلى تنفيذ المعالجات المناسبة في المستقبل بناءً على نوع الإشارة المستلمة.

8. الإشارة 17: (`SIGCHLD`)

- وظيفتها: الإشارة `SIGCHLD` تُرسل إلى العملية الأم عندما تنتهي إحدى العمليات الفرعية (الأبناء) أو تتوقف مؤقتاً. تستفيد العملية الأم من هذه الإشارة لمعرفة حالة العملية الفرعية دون الحاجة للانتظار النشط.

- التعامل معها في الكود: في الكود، يتم تعريف معالج للإشارة `SIGCHLD` لكي تتعامل العملية الأم مع انتهاء العمليات الفرعية بهدوء.

- كيفية قتل البرنامج في هذا الكود: في الكود، يتم إرسال الإشارة `SIGKILL` من العملية المرسلة (`sender`) إلى العملية المستقبلة (`receiver`) عبر الدالة `kill`. الإشارة `SIGKILL` هي إشارة خاصة تُرسل لإنهاء عملية بشكل فوري وغير قابل للمعالجة. لا يمكن للبرنامج تجاهل أو معالجة هذه الإشارة، مما يؤدي إلى إنهاء العملية المستقبلة فور استلامها.

- مثال في الكود:

1. إرسال الإشارة: في الكود `sender.c` ، يتم اختيار إشارة عشوائية من بين عدة إشارات، وإذا كانت الإشارة هي `SIGKILL` ، يتم إرسالها إلى العملية المستقبلة باستخدام الـ

`.kill(pid, SIGKILL)`

2. استلام الإشارة: في الكود `receiver.c` ، عند استلام الإشارة `SIGKILL` ، يتم إنهاء العملية فوراً لأنها لا تستطيع معالجة هذه الإشارة.

بالتالي، عندما يرسل المرسل الإشارة `SIGKILL` ، يتم إنهاء العملية المستقبلة والبرنامـج مباشرة. هذه العملية تجعل من الممكن قتل البرنامج بشكل فوري باستخدام هذه الإشارة، وبالتالي لن تتم معالجة الإشارة رقم 17 ، بالإضافة إلى أنه كما تم ذكره سابقاً أن الإشارة 17 ترسل إلى العملية الأم عندما تنتهي إحدى العمليـات الفرعية الأبناء ونحن في هذا البرنامج نستخدم عمليتين مستقلتين تماماً وهما المرسل والمستقبل، وبالتالي لن تتم معالجة الإشارة رقم 17 للخروج عند إرسال الإشارة رقم 9 .

```

File Edit Search View Document Help
File Actions Edit View Help
kali@kali: ~ kali@kali: ~ kali@kali: ~
(kali㉿kali)-[~]
$ nano receiver.c
(kali㉿kali)-[~]
$ gcc -o receiver receiver.c
(kali㉿kali)-[~]
$ 

```

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <signal.h>
5
6 void handle_signal(int signo) {
7     switch (signo) {
8         case SIGHUP:
9         case SIGINT:
10        case SIGQUIT:
11            printf("Received and ignored signal %d\n", signo);
12            break;
13        case SIGKILL:
14            printf("Received signal %d (SIGKILL), terminating process.\n", signo);
15            exit(0);
16            break;
17    }
18 }
19
20 int main() {
21     struct sigaction sa;
22     sa.sa_handler = handle_signal;
23     sa.sa_flags = 0;
24     sigemptyset(&sa.sa_mask);
25
26     sigaction(SIGHUP, &sa, NULL);
27     sigaction(SIGINT, &sa, NULL);
28     sigaction(SIGQUIT, &sa, NULL);
29     sigaction(SIGKILL, &sa, NULL);
30
31     while (1) {
32         pause(); // Wait for signals
33     }
34
35     return 0;
36 }

```

• شرح مبدأ عمل الكود:

المستقبل يعالج الإشارات التي يستقبلها من المرسل:

1. تعریف معالج الإشارات: يقوم البرنامج بتحديد كيفية التعامل مع كل إشارة.

2. معالجة الإشارات المختلفة:

- بالنسبة للإشارات 1 و 2 و 3 تشير إلى SIGINT و SIGHUP و SIGQUIT على التوالي: يتم طباعة رسالة تجاهل الإشارة.

- بالنسبة للإشارة SIGKILL: يتم طباعة رسالة وإنهاء البرنامج.

3. انتظار الإشارات: يدخل البرنامج في حالة انتظار دائمة للإشارات.

باختصار:

- المرسل يرسل إشارات عشوائية إلى المستقبل كل ثانية، وينهي نفسه عند إرسال

- المستقبل يعالج الإشارات عن طريق طباعة رسائل مناسبة وينهي SIGKILL

نفسه عند استلام SIGKILL.

السؤال الخامس:

```
File Edit Search View Document Help
File Edit Search View Document Help
1 #include <stdio.h>
2 #include <semaphore.h>
3 #include <fcntl.h>
4 #include <sys/ipc.h>
5 #include <sys/shm.h>
6 #include <sys/stat.h>
7 #include <sys/mman.h>
8 #include <sys/wait.h>
9 #include <string.h>
10 #define BufferSize 5
11 const char *MemoryName = "My_little_memory";
12 const int MemorySize = 12;
13 char Buffer[BufferSize];
14 int count = 0;
15 char Info[] = "Yaman";
16 void *pointer;
17 sem_t *Mutex;
18 sem_t *Empty;
19 sem_t *Items;
20 void Producer() {
21     sem_wait(Mutex);
22     sem_wait(Empty);
23     Buffer[count] = Info[count];
24     printf("Producer: produces [%c], and add to Buffer!\n\n", Info[count]);
25     count += 1;
26     sem_post(Mutex);
27     sem_post(Items);
28 }
29 void Consumer() {
30     char item;
31     sem_wait(Mutex);
32     sem_wait(Items);
33     item = Buffer[count - 1];
34     printf("Consumer: consumes [%c], and delete from Buffer!\n\n", item);
35     count -= 1;
36     sem_post(Mutex);
37     sem_post(Empty);
38 }
39 int main() {
40     Mutex = sem_open("/mutex", O_CREAT, 0644, 1);
41     Empty = sem_open("/empty", O_CREAT, 0644, BufferSize);
42     Items = sem_open("/items", O_CREAT, 0644, 0);
43     printf("Welcome to (Producer and Consumer) program!\n");
44     int choose;
45     int Get_shmid = shm_open(MemoryName, O_CREAT | O_RDWR, 0666);
46     if (Get_shmid < 0) {
47         printf("Failed to create share memory id!\n");
48         return 1;
49     }
50     pointer = mmap(NULL, MemorySize, PROT_READ | PROT_WRITE, MAP_SHARED, Get_shmid, 0);
51     if (pointer == MAP_FAILED) {
52         printf("Failed to mmap");
53         return 1;
54     }
55     while (1) {
56         printf("Press [1] to produce\nPress [2] to consume\nPress [3] to exit\n");
57         scanf("%d", &choose);
58         if (choose == 1) {
59             if (count >= BufferSize) {
60                 printf("Buffer is full, can't produce!\n\n");
61             } else {
62                 Producer();
63             }
64         }
65         else if (choose == 2) {
66             if (count <= 0) {
67                 printf("Buffer is empty, can't consume!\n\n");
68             } else {
69                 Consumer();
70             }
71         }
72         else if (choose == 3) {
73             break;
74         }
75         else {
76             printf("Invalid choice");
77         }
78     }
79 }
80 munmap(pointer, MemorySize);
81 sem_close(Mutex);
82 sem_close(Items);
83 sem_close(Empty);
84 sem_unlink("/mutex");
85 sem_unlink("/empty");
86 sem_unlink("/items");
87 sem_unlink("/items");
88 }
```

```
kali㉿kali:[~/Desktop]
$ ./synch
Welcome to (Producer and Consumer) program!
Press [1] to produce
Press [2] to consume
Press [3] to exit
1
Producer: produces [M], and add to Buffer!

Press [1] to produce
Press [2] to consume
Press [3] to exit
1
Producer: produces [a], and add to Buffer!

Press [1] to produce
Press [2] to consume
Press [3] to exit
1
Producer: produces [n], and add to Buffer!

Press [1] to produce
Press [2] to consume
Press [3] to exit
1
Producer: produces [a], and add to Buffer!

Press [1] to produce
Press [2] to consume
Press [3] to exit
1
Producer: produces [r], and add to Buffer!

Press [1] to produce
Press [2] to consume
Press [3] to exit
1
Buffer is full, can't produce!

Press [1] to produce
Press [2] to consume
Press [3] to exit
3

```

```
kali㉿kali:[~/Desktop]
$ ./synch
Welcome to (Producer and Consumer) program!
Press [1] to produce
Press [2] to consume
Press [3] to exit
1
Producer: produces [M], and add to Buffer!

Press [1] to produce
Press [2] to consume
Press [3] to exit
2
Consumer: consumes [M], and delete from Buffer!

Press [1] to produce
Press [2] to consume
Press [3] to exit
2
Buffer is empty, can't consume!

Press [1] to produce
Press [2] to consume
Press [3] to exit
1
Producer: produces [M], and add to Buffer!

Press [1] to produce
Press [2] to consume
Press [3] to exit
1
Producer: produces [a], and add to Buffer!

Press [1] to produce
Press [2] to consume
Press [3] to exit
2
Consumer: consumes [a], and delete from Buffer!

Press [1] to produce
Press [2] to consume
Press [3] to exit
3

```

► شرح الكود:

- يقوم البرنامج بداية بفتح السيمافورات الثلاثة Mutex, Empty, Items التي قمنا بتعريفها في المنطقه العامة "Global" خارج التوابع.
- يأخذ الثلاثة صلاحيات قراءة وكتابة للملك وقراءة للمجموعة والآخرين.
- يأخذ كل من السيمافور Mutex قيمة ابتدائية 1، أما السيمافور Items القيمة 0.
- أما السيمافور Empty يأخذ القيمة الابتدائية لحجم الخزان المؤقت المعرف بالمنطقه العامة "Global" وهي 5.
- يقوم بعدها البرنامج بالترحيب بالمستخدم.
- ثم يقوم البرنامج بإنشاء متغير choose وبعدها يقوم بإنشاء ذاكرة مشتركة عن طريق التابع `shm_open` مع تمرير اسم الذاكرة المشتركة الذي قد تم تعريفه مسبقاً كـ "Global" وتمرير القيمة `O_CREAT` ليقوم بإنشاء الذاكرة المشتركة في حال عدم وجودها وتمرير أيضاً `0666` وهي صلاحيات والتي تعطى كالآتي : قراءة وكتابة للملك والمجموعة والآخرين ، أما `0` فهي تعني انه لا يوجد أي صلاحيات إضافية.
- بعدها يتم التحقق في حال كان القيمة العائدة من التابع `shm_open` أصغر تماماً من صفر فإنه قد فشل إنشاء الذاكرة المشتركة ويتم إخبار المستخدم بذلك وإعادة القيمة `1` لإنتهاء البرنامج.
- بعدها يتم استخدام البوينتر الذي تم إنشائه مسبقاً كـ "Global" مع استخدام التابع `mmap` التابع `mmap` يأخذ البارامترات.
- الأول `NULL` وتعني أن النظام هو من يختار العنوان الذي سيكون عنده تعين الذاكرة المشتركة.
- الثاني `MemorySize` وهو متغير يتضمن حجم الذاكرة التي تم إنشائها مسبقاً وإعطائه القيمة `12`.
- الثالث `PROT_READ | PROT_WRITE` وهي قيم تعني أن المنطقه المشتركة ستكون قابلة للقراءة والكتابة.
- الرابع `MAP_SHARED` وهو قيمة تعني أن أي تغيير يحصل في المنطقه المشتركة سيكون ممئي للجميع.

- الخامس Get_shmid وهو المعرف الخاص بالذاكرة المشتركة وقيمه قد تم جلبها من التابع `shm_open` مسبقاً.
- السادس وهو القيمة 0 وهو يعني أننا نريد استعمال جميع مساحة الذاكرة المشتركة من بدايتها.
- بعدها يقوم البرنامج بفحص ما إذا كان هناك أي خطأ حدث أثناء تعيين الذاكرة المشتركة بواسطة التابع `mmap`.
- إذا كانت القيمة العائدة `MAP_FAILED` هذا يعني أن العملية فشلت ويقوم البرنامج بإخبار المستخدم بالفشل والخروج من البرنامج.
- يقوم بعدها البرنامج بالدخول بحلقة نهائية قيمتها 1 والتي تعني `TRUE`, يقوم البرنامج بسؤال المستخدم إذا كان يريد إنتاج فليضغط 1 أما استهلاك فليضغط 2 أما إذا أراد الخروج من البرنامج فليضغط 3.
- ويقوم عن طريق التابع `scanf` بأخذ القيمة من المستخدم وتخزينها بالمتغير `choose` الذي تم تعريفه مسبقاً.
- في حال كانت القيمة المدخلة هي 1 أي إنتاج يقوم البرنامج بفحص المتغير `count` وهو متغير يقوم البرنامج بزيادته في كل مرة نقوم فيها بإنتاج حرف من حروف الاسم المعطى، في حال كان المتغير `count` يساوي أو أكبر من الخزان المؤقت وبالتالي الخزان ممتلئ ولا يمكننا الإنتاج.
- أما في حال كان المتغير `count` أصغر تماماً من الخزان المؤقت وبالتالي الخزان لم يتمتع بعد ويمكننا الإنتاج فيقوم البرنامج باستدعاء التابع `producer` والذي يقوم بدوره كالتالي: يقوم التابع بدايةً بتعليق السيمافور `Mutex` هذا يعني أن هناك إجرائية تقوم بعملية معينة داخل المقطع الحرج وبالتالي لا يمكن لأي عملية أخرى أن تقوم بأي عملية ما لم يتم فك السيمافور `Mutex` عن طريق `sem_post` وهنا يتحقق لدينا مفهوم الاستبعاد المتبادل.
- أيضاً يقوم بتعليق السيمافور `Empty` وهو المسؤول عن عدد الأماكن المتوفرة في الخزان المؤقت.
- إذا كان المخزن المؤقت ممتلئاً، ستنتظر العملية حتى يتم استهلاك عنصر واحد على الأقل.

- يقوم بالسطر التالي `Buffer[count] = Info[count]` بإضافة الحرف المراد إنتاجه من المورد الأساسي وهو مصفوفة المحارف `Info` إلى الخزان المؤقت عن طريق المتغير `.index` والذي يعمل عمل الفهرس أو `count` بعدها يتم إخبار المستخدم أنه تم إنتاج الحرف المعين وإضافته إلى الخزان المؤقت.
- ثم يقوم بزيادة العداد `count` بمقدار 1 كإشارة إلى المكان التالي في الخزان المؤقت.
- بعدها يقوم التابع بتحرير السيمافور `Mutex` أي أنه يمكن لأي إجرائية الآن الدخول إلى المنطقة المشتركة وإجراء عمليات عليها.
- ثم يقوم بتحرير السيمافور `Items` إشارة إلى أن هناك عنصر جديد تم إضافته إلى الخزان.
- وينتهي التابع `Producer`تابع الإنتاج ثم يعود البرنامج إلى التابع الرئيسي.
- عند عودة البرنامج إلى التابع الرئيسي قد يعود المستخدم ويدخل 1 لينتج حرف آخر ولكن قد يدخل 2.
- عند إدخال 2 أي أن المستخدم يريد الاستهلاك من الخزان يقوم البرنامج بفحص العدد في حال كان أصغر أو يساوي 0 وبالتالي الخزان المؤقت فارغ ولم ينتج شيء من قبل، وبالتالي لا يمكن للمستخدم أن يستهلك فيقوم البرنامج بطباعة عبارة تخبر المستخدم أن الخزان فارغ لا يمكنك الاستهلاك.
- أما إذا لم يكن الخزان فارغ وبالتالي قيمة `count` تساوي 1 فأعلى هنا يتم استدعاء التابع `Consumer` والذي يقوم بدوره:
- يقوم التابع بداية بإنشاء المتغير `item` من نوع `char` لتخزين العنصر الذي سيتم استهلاكه.
- يقوم التابع بتعليق السيمافور `Mutex` هذا يعني أن هناك إجرائية تقوم بعملية معينة داخل المقطع الحرج وبالتالي لا يمكن لأي عملية أخرى أن تقوم بأي عملية ما لم يتم فك السيمافور `Mutex` عن طريق `sem_post` وهنا يتحقق لدينا مفهوم الاستبعاد المتبادل أيضاً.
- أيضاً يقوم بتعليق السيمافور `Items` وهنا نضمن أن الخزان يحوي عناصر جاهزة للاستهلاك، وفي حال كان الخزان فارغ ستنتظر الإجرائية حتى يقوم منتج ما بإضافة عنصر للخزان.

- يقوم السطر التالي [1 - item = Buffer[count] بعملية الاستهلاك من المخزن إذ يقوم بأخذ العنصر الأخير ووضعه في المتغير item وإخبار المستخدم عن طريق امر طباعة أن العنصر تم استهلاكه وإزالته من الخزان.
- ثم يقوم بخفض قيمة العداد بمقدار واحد كإشارة إلى المكان الفارغ في الخزان المؤقت.
- بعدها يقوم التابع بتحرير السيمافور Mutex أي انه يمكن لأي إجرائية الآن الدخول إلى المنطقة المشتركة وإجراء عمليات عليها.
- ثم يقوم بتحرير السيمافور Empty إشارة إلى أنه تم استهلاك عنصر من الخزان وبالتالي هناك مكان فارغ ضمن الخزان.
- .انتهى التابع Consumer.
- يعود البرنامج إلى التابع الرئيسي ويسأل المستخدم إذا كان يريد الإنتاج أو الاستهلاك أو الخروج.
- إذا قام المستخدم بإدخال القيمة 3 أي يريد الخروج من البرنامج يقوم البرنامج بتنفيذ تعليمة break تساعدة من الخروج من الحلقة النهاية وبالتالي تنتهي الحلقة تماماً ويقوم البرنامج بتنفيذ عدة تعليمات مهمة.
- ببدايةً (munmap(pointer, MemorySize) والتي بدورها تقوم بتحرير الذاكرة المشتركة وانتهائها.
- بعدها يقوم بإغلاق جميع السيمافورات التي قد سبق وتم إنشائها،
- Empty
- ثم يقوم بحذف جميع السيمافورات الثلاثة من النظام عن طريق .sem_unlink