

علی اسدی 403203919

تمرین سوم سیستم های نهفته بی درنگ

بخش عملی قسمت دوم

پاسخ به سوالات در فایل README داده شده است و جواب ها طبق فرمات گفته شده در مخزن گیت هاب پوش شده است
آدرس گیت هاب :

1 اول به سوالات پاسخ داده شده است بعد 2 روند کار و نتایج بدست امده را بررسی میکنیم و 3 قطعه کد ها را توضیح میدهیم.

1- پاسخ به سوالات در تمرین

کاوش کنید :

FIFO (Named Pipes)

تعریف :

نوعی pipe است که برخلاف pipe های معمولی، در فایل سیستم با یک نام خاص ثبت می شود و پردازه های مستقل (نه فقط پدر/فرزند) می توانند به آن دسترسی داشته باشند.

ویژگی ها :

- انتقال داده ها به صورت صف(First-In-First-Out) •
- فایل ویژه با دستور mkfifo ساخته می شود •
- داده ها بین پردازه ها با توابع استاندارد فایل (read, write) (رد و بدل می شوند) •
- مثال ساده : •

mkfifo /tmp/myfifo

یک پردازه :

echo "hello" > /tmp/myfifo

پردازه دیگر :

cat /tmp/myfifo

یک کانال ارتباطی است که به صورت صف عمل می کند.

داده ها به ترتیب ارسال و دریافت می شوند. (First In First Out).

برای ارتباط یک طرفه بین دو فرآیند استفاده می شود.

از فایل‌های خاص در سیستم فایل به عنوان واسطه استفاده می‌کند.

ساده و مناسب برای ارتباطات کوتاه و ساده.

Linux Signals

تعریف:

سیگنال‌ها برای اطلاع‌رسانی به پردازه‌ها درباره رویدادهایی مانند وقفه‌ها، خطاهای، یا رویدادهای کاربر استفاده می‌شوند.

ویژگی‌ها:

- سیگنال‌ها اعداد صحیحی هستند مثلًا SIGINT, SIGTERM, SIGKILL
- قابل استفاده برای توقف، از سرگیری، یا پایان دادن به پردازه‌ها
- می‌توان هندرلهای خاص برای دریافت و مدیریت آن‌ها نوشت با signal() یا sigaction()

مثال ساده:

```
signal(SIGINT, handler_function);
```

پیام‌های کوتاه و فوری برای اطلاع‌رسانی به فرآیندها.

معمولًا برای هشدار یا کنترل فرآیند (مثل توقف، ادامه، پایان) کاربرد دارند.

سیگنال‌ها دارای شماره و نام هستند مثلًا SIGKILL, SIGINT

قابل مدیریت و هندل توسط برنامه‌ها هستند (به جز چند سیگنال خاص).

سرعت بالایی دارند ولی حجم داده محدود است.

Shared Memory

تعریف:

در این روش، چند پردازه می‌توانند به یک بخش از حافظه دسترسی مستقیم داشته باشند؛ سریع‌ترین روش IPC است چون نیازی به کپی‌برداری از داده‌ها نیست.

ویژگی‌ها:

- نیازمند همزمان‌سازی) مثلاً با mutex یا semaphore برای جلوگیری از race condition استفاده از API‌هایی مثل (shmget(), shmat(), shmdt(), shmctl()

مثال کاربردی:

```
int shmid = shmget(IPC_PRIVATE, 1024, IPC_CREAT | 0666);  
char *shmaddr = (char *) shmat(shmid, NULL, 0);
```

فضایی در حافظه که توسط چند فرآیند به اشتراک گذاشته می‌شود.

سریع‌ترین روش IPC به دلیل دسترسی مستقیم به داده‌ها در RAM.

نیازمند هماهنگی و همزمان‌سازی برای جلوگیری از تداخل داده‌ها مثلاً با Semaphore.

امکان خواندن و نوشتن مستقیم داده‌های پیچیده را فراهم می‌کند.

مناسب برنامه‌های پیچیده با حجم بالای داده.

ویژگی‌ها	FIFO	Signals	Shared Memory
سرعت انتقال	متوفط	بسیار سریع (اما محدود به اطلاع‌رسانی)	بسیار بالا
پیچیدگی	کم	کم تا متوفط	زیاد (نیاز به همزمان‌سازی)
نوع ارتباط	داده متوالی	اطلاع‌رسانی لحظه‌ای	داده حجیم مستقیم
همزمان‌سازی	نه	نه	بله (ضروری)

IPC Mechanism	Data Transfer	Direction	Persistence	Speed	Complexity	Typical Use Cases
FIFO	Byte stream	Unidirectional	Transient	Moderate	Low	Simple producer-consumer tasks
Signals	Signal number	Asynchronous	Transient	Fast	Moderate	Notifications, control signals
Shared Memory	Shared memory	Bidirectional	Persistent (till removed)	Very fast	High	Large data, high performance

۱-تفاوت بین سیگنالهای SIGKILL و SIGINT چیست؟

تفاوت بین SIGKILL و SIGINT چیست؟

سیگنال

توضیح

SIGINT(2) وقتی در ترمینال کلیدهای Ctrl+C را فشار می‌دهید، این سیگنال به پردازه ارسال می‌شود.

به پردازه فرصت داده می‌شود تا خودش را به درستی خاتمه دهد (قابل مدیریت است).

SIGKILL(9) این سیگنال پردازه را بلاخلاصه و بدون امکان مدیریت یا پاکسازی منابع متوقف می‌کند

(غیرقابل کنترل و برگشت‌ناپذیر).

SIGKILL قابل کنترل یا مسدودسازی نیست

SIGINT قابل کنترل است

SIGINT(2) (Interrupt Signal)

این سیگنال زمانی ارسال می‌شود که کاربر کلید Ctrl+C را در ترمینال فشار دهد. سیگنال SIGINT به فرآیند این امکان را می‌دهد که قطع شود اما می‌تواند آن را مدیریت کند. یعنی برنامه می‌تواند با استفاده از هندر (signal handler) تصمیم بگیرد چه واکنشی به آن نشان دهد (مثلًاً ذخیره وضعیت، چاپ پیغام، یا حتی نادیده گرفتن آن).

SIGKILL(9) (Kill Signal)

این سیگنال برای اجبار به خاتمه فوری فرآیند است. برخلاف SIGINT، سیگنال SIGKILL قابل مدیریت یا نادیده گرفتن نیست. سیستم عامل بلافصله فرآیند را متوقف می کند، بدون اینکه فرصتی برای تمیز کردن منابع یا ذخیره وضعیت باقی بگذارد.

Feature	SIGINT	SIGKILL
Signal Number	2	9
Purpose	Interrupt signal (typically sent when user presses Ctrl+C)	Kill signal (forces immediate termination)
Can be caught or handled?	Yes , processes can catch, ignore, or handle SIGINT	No , cannot be caught, blocked, or ignored
Effect on process	Process can perform cleanup before terminating or continue execution if handled	Immediate termination without cleanup
Use case	Graceful termination initiated by user	Forceful termination of unresponsive or stuck processes

2 کدام سیگنالها میتوانند توسط یک فرآیند دریافت یا مدیریت شوند؟

اغلب سیگنال‌ها را می‌توان توسط توابعی مثل `signal()` یا `sigaction()` دریافت و مدیریت کرد.

اما دو سیگنال زیر غیرقابل کنترل هستند:

• عدد 9 SIGKILL

• عدد 19 SIGSTOP

بقیه سیگنال‌ها مثل SIGUSR1، SIGINT، SIGTERM و غیره قابل مدیریت یا حتی نادیده‌گیری هستند.

تقریباً تمام سیگنال‌ها بجز SIGSTOP و SIGKILL قابل مدیریت یا گرفتن (catch) توسط فرآیندها هستند.

• سیگنال‌هایی مثل:

◦ SIGINT توسط کاربر قطع

◦ SIGTERM پایان نرم فرآیند

SIGHUP بستن ترمینال ○

SIGUSR2، SIGUSR1 برای استفاده‌های سفارشی ○

SIGCHLD، SIGPIPE، SIGALRM و بسیاری دیگر ○

می‌توانند با signal() یا sigaction() مدیریت شوند.

3 چند سیگنال در لینوکس تعریف شد اند؟

در لینوکس معمولاً دو نوع سیگنال داریم:

سیگنال‌های استاندارد: حدود 31 عدد (از 1 تا 31)

سیگنال‌های Real-Time: از عدد 34 به بالا تا حدود 64

در مجموع، معمولاً 64 تا سیگنال در سیستم‌های لینوکس تعریف شده است.

برای مشاهده همه سیگنال‌ها در سیستم خودتان می‌توانید دستور زیر را اجرا کنید:

kill -l

در لینوکس معمولاً بین 31 تا 64 سیگنال استاندارد تعریف شده‌اند (بسته به معماری و نسخه کرنل). رایج‌ترین‌ها در فایل هدر

signal.h تعریف می‌شوند.

```

root@orangepizeroplus:~/myprojects/signals# kill -l
 1) SIGHUP      2) SIGINT      3) SIGQUIT      4) SIGILL      5) SIGTRAP
 6) SIGABRT     7) SIGBUS      8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX

```

تفاوت های کلیدی بین FIFO و حافظه اشتراکی برای IPC چیست؟

حافظه اشتراکی(Shared Memory)	FIFO (Named Pipe)	ویژگی
بسیار سریع (دسترسی مستقیم به حافظه)	کنترل (انتقال از طریق کرنل)	سرعت
دو طرفه (با همگام سازی)	یک طرفه یا نیمه دو طرفه	جهت ارتباط
دارد) مثلاً با mutex یا semaphore	ندارد (خودکار)	نیاز به همگام سازی
پیچیده تر استفاده از ...	ساده ایجاد با mkfifo	میزان پیچیدگی
اشتراک داده های زیاد و با سرعت بالا	پیام های ساده یا متنی بین دو پردازه	کاربرد مناسب

FIFO:

توضیح

ویژگی

ساختار	FIFO یک صف خطی است که داده‌ها را به همان ترتیبی که ارسال شده‌اند، دریافت می‌کند.
سرعت	معمولًاً کندر از حافظه اشتراکی است زیرا سیستم عامل باید داده‌ها را بین فرآیندها کپی کند.
ساده‌تر در استفاده	پیاده‌سازی و استفاده نسبتاً ساده‌تری دارد (مثل فایل معمولی رفتار می‌کند).
فرآیندها باشد.	فرآیندهای خواندن در صورت نبودن نویسنده مسدود می‌شود و بالعکس مگر نویسنده اینکه non-blocking باشد.
نحوه انتقال داده	داده‌ها از یک فرآیند به فرآیند دیگر کپی می‌شود.
دسترسی	فقط دو طرف در یک زمان خاص می‌توانند از آن استفاده کنند (خواننده و نویسنده).
موقعیت	در سیستم فایل ایجاد می‌شود مانند <code>/tmp/myfifo</code> .

Shared Memory:

توضیح

ویژگی

ساختار	یک فضای حافظه‌ای مشترک بین چندین فرآیند که می‌توانند مستقیم به آن دسترسی داشته باشند.
سرعت	بسیار سریع‌تر از FIFO است چون داده‌ها نیاز به کپی شدن ندارند، بلکه مستقیماً خوانده/نوشته می‌شوند.
پیچیدگی	نیاز به هماهنگی بیشتر بین فرآیندها دارد مثلاً با semaphore برای جلوگیری از تداخل دسترسی همزمان. بیشتر
فرآیندها	فرآیندها می‌توانند مستقل از زمان اجرا به حافظه دسترسی داشته باشند تا زمانی که بخش حافظه حذف نشده باشد.
نحوه انتقال داده	داده‌ها در یک محل قرار می‌گیرند و توسط هر دو فرآیند به‌طور مستقیم دیده می‌شوند.
دسترسی	چندین فرآیند می‌توانند به‌طور همزمان به حافظه دسترسی داشته باشند.
موقعیت	در فضای حافظه سیستم ذخیره می‌شود، نه در فایل سیستم.

خلاصه تفاوت‌ها:

ویژگی	FIFO	حافظه اشتراکی (Shared Memory)
سرعت	کندتر	سریع‌تر
انتقال داده	کمی بین فرآیندها	دسترسی مستقیم
نیاز به همزمانی (Synchronization)	بله	نه لزوماً (تا زمانی که حافظه باقی است)
هماهنگی	ساده‌تر	نیاز به ابزار کمکی مثل semaphore
استفاده	راحت‌تر	پیچیده‌تر ولی مؤثرتر

نتیجه: اگر سرعت و حجم داده مهم است، حافظه اشتراکی گزینه بهتری است. برای ارتباط ساده بین دو پردازه، FIFO کافی است.

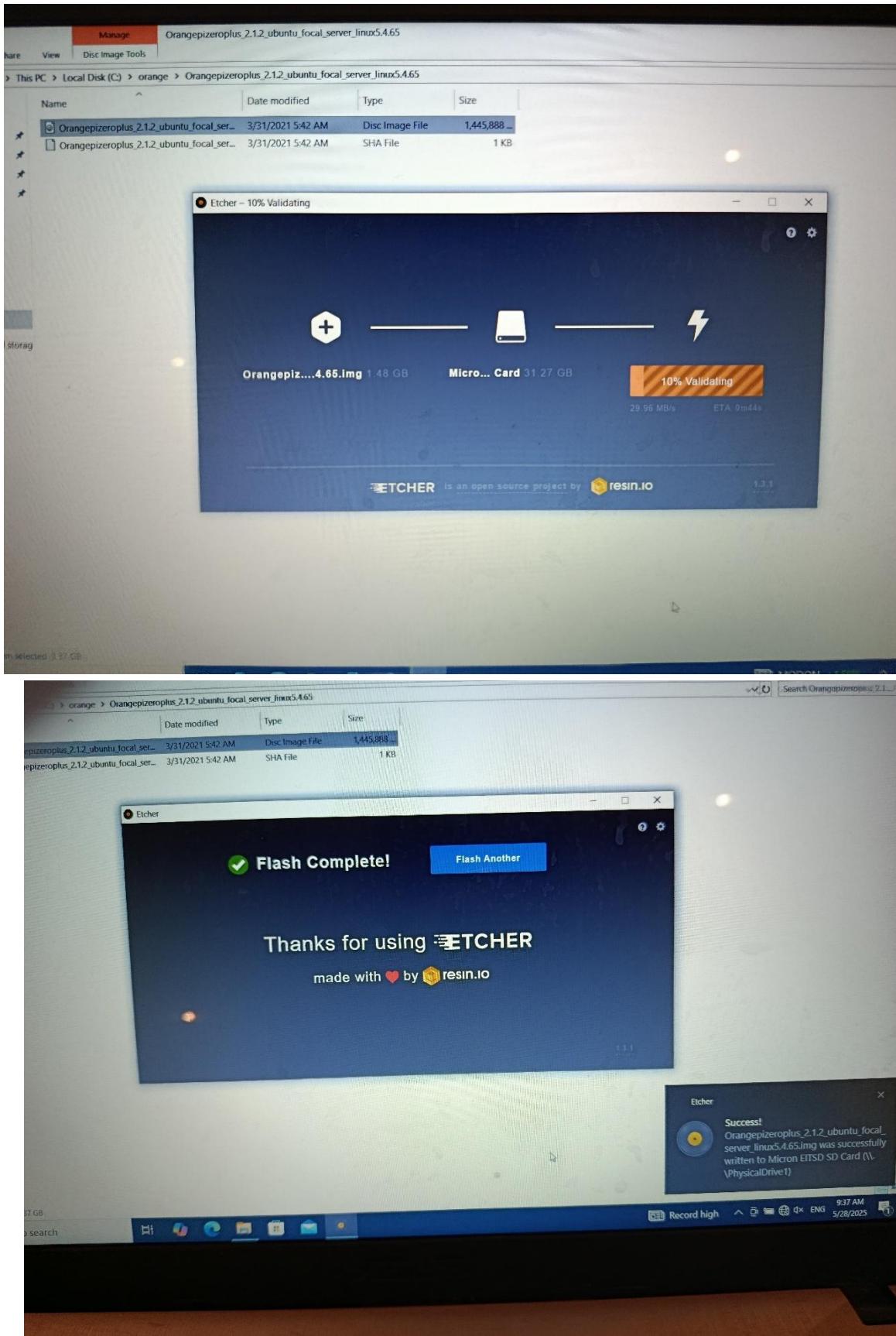
2 روند کار و نتایج بدست آمده:

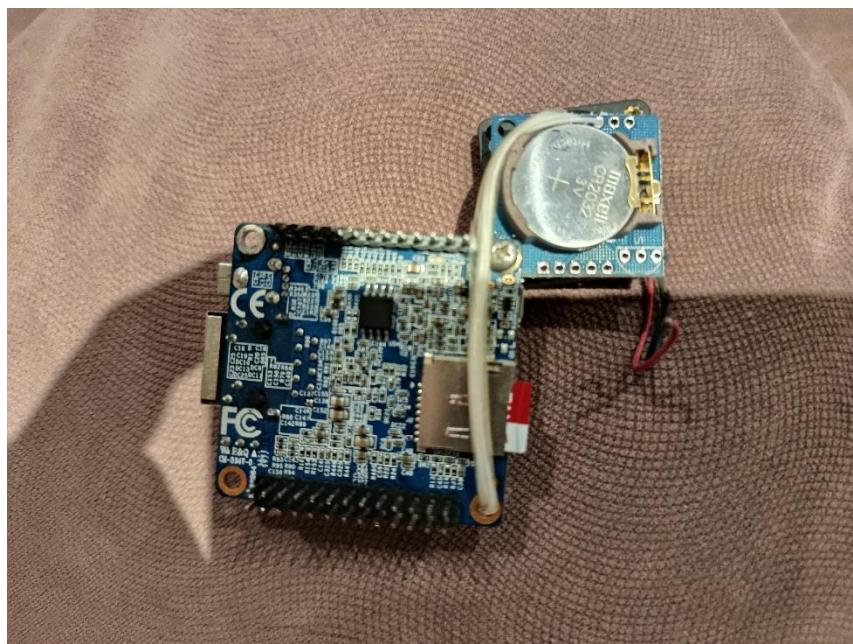
ابتدا یک حافظه 32 گیگ MicroSD تهیه شد



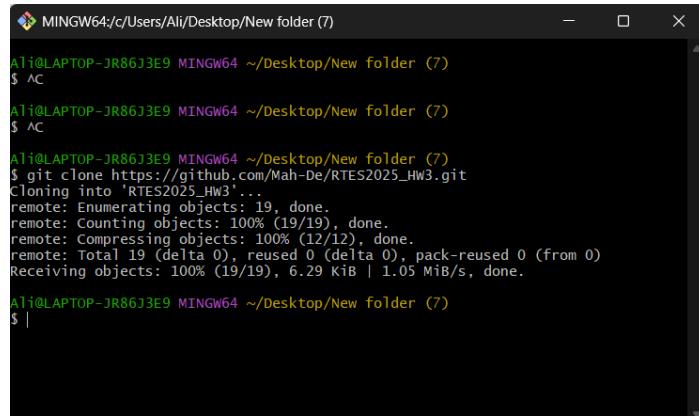
سپس با یک لپ تاپ دیگه که رم ریدر داشت سیستم عامل
OrangePizeroplus_2.1.2_ubuntu_focal_server_linux5.4.65
را دانلود شد و با برنامه Etcher-Setup-1.3.1-x64 روی MicroSd ریخته شد





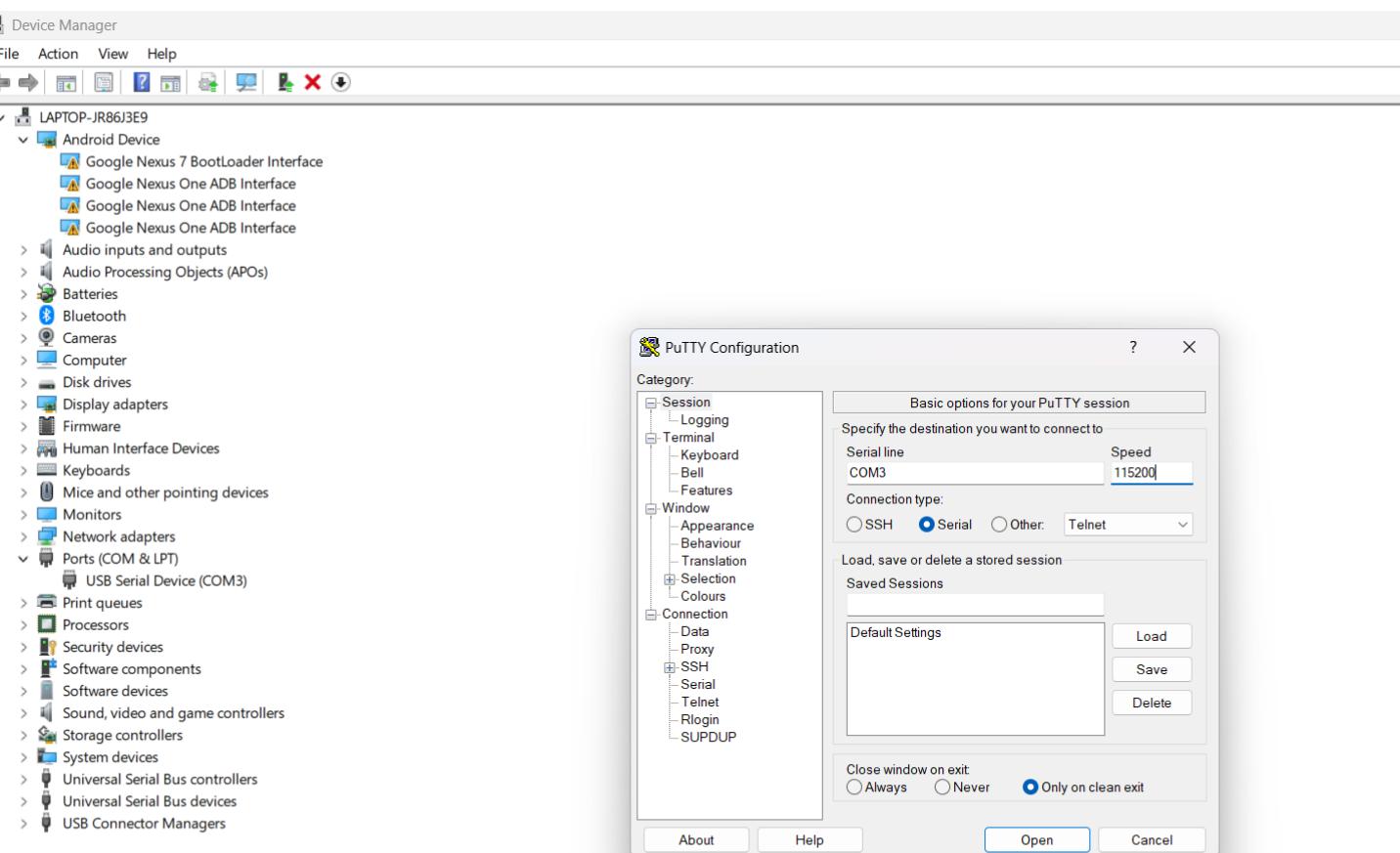


گیت هم نصب میکنیم و آدرس مخزن تمرين را clone میکنیم



```
Ali@LAPTOP-JR86J3E9 MINGW64 ~/Desktop/New folder (7)
$ ^C
Ali@LAPTOP-JR86J3E9 MINGW64 ~/Desktop/New folder (7)
$ ^C
Ali@LAPTOP-JR86J3E9 MINGW64 ~/Desktop/New folder (7)
$ git clone https://github.com/Mah-De/RTES2025_HW3.git
Cloning into 'RTES2025_HW3'...
remote: Enumerating objects: 19, done.
remote: Counting objects: 100% (19/19), done.
remote: Compressing objects: 100% (12/12), done.
remote: Total 19 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (19/19), 6.29 KiB | 1.05 MiB/s, done.
Ali@LAPTOP-JR86J3E9 MINGW64 ~/Desktop/New folder (7)
$ |
```

حال پس از نصب برنامه Putty و اطلاع از شماره USB متصل شده به لپ تاپ تنظیمات زیر را در Putty دنبال میکنیم



root

orangeipi

COM3 - PuTTY

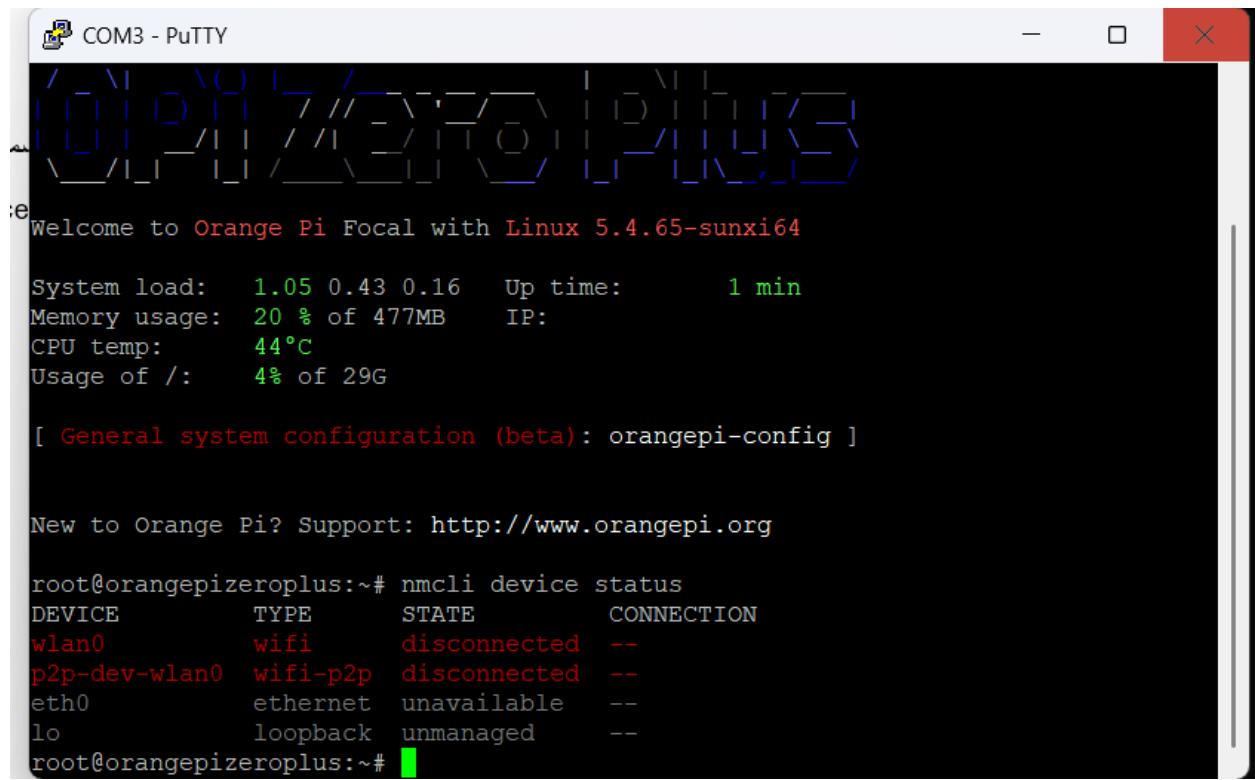
```
Orange Pi 2.1.2 Focal ttyGS0
orangepizeroplus login: root
Password:
Welcome to Orange Pi Focal with Linux 5.4.65-sunxi64
System load: 1.05 0.43 0.16 Up time: 1 min
Memory usage: 20 % of 477MB IP:
CPU temp: 44°C
Usage of /: 4% of 29G
[ General system configuration (beta): orangepi-config ]

New to Orange Pi? Support: http://www.orangepi.org
root@orangepizeroplus:~#
```

اتصال به SSH و Fi-Wi

با دستور زیر میتوانیم وضعیت اتصال به اینترنت را مطلع شویم

```
nmcli device status
```



COM3 - PuTTY

```
:e Welcome to Orange Pi Focal with Linux 5.4.65-sunxi64

System load: 1.05 0.43 0.16 Up time: 1 min
Memory usage: 20 % of 477MB IP:
CPU temp: 44°C
Usage of /: 4% of 29G

[ General system configuration (beta): orangepi-config ]

New to Orange Pi? Support: http://www.orangepi.org

root@orangepizeroplus:~# nmcli device status
DEVICE      TYPE      STATE      CONNECTION
wlan0       wifi      disconnected  --
p2p-dev-wlan0 wifi-p2p  disconnected  --
eth0        ethernet  unavailable  --
lo          loopback  unmanaged   --
root@orangepizeroplus:~#
```

حال لیست وای فای ها اطراف را میبینیم

nmcli device wifi list



COM3 - PuTTY

```

Orange Pi 2.1.2 Focal ttyGS0

orangepizeroplus login: root
Password:

Welcome to Orange Pi Focal with Linux 5.4.65-sunxi64

System load: 1.05 0.43 0.16 Up time: 1 min
Memory usage: 20 % of 477MB IP:
CPU temp: 44°C
Usage of /: 4% of 29G

[ General system configuration (beta): orangepi-config ]

New to Orange Pi? Support: http://www.orangepi.org

root@orangepizeroplus:~# nmcli device status
DEVICE      TYPE      STATE      CONNECTION
wlan0        wifi      disconnected  --
p2p-dev-wlan0 wifi-p2p  disconnected  --
eth0         ethernet  unavailable  --
lo           loopback  unmanaged   --
root@orangepizeroplus:~# nmcli device wifi list
IN-USE  BSSID          SSID      MODE     CHAN  RATE      SIGNAL  BARS  SECURITY
        48:FD:8E:84:95:F8  Ali       Infra    1      130 Mbit/s  44      **    WPA2  >
...skipping...
IN-USE  BSSID          SSID      MODE     CHAN  RATE      SIGNAL  BARS  SECURITY
        48:FD:8E:84:95:F8  Ali       Infra    1      130 Mbit/s  44      **    WPA2  >
~
~
```

با دستور زیر با **ssid** و رمز مودم به اینترنت متصل میشویم

`nmcli device wifi connect "Ali" password 'ali1379!#(aSadI'`

```

nmcli device wifi connect "Ali" password 'aIi1379nmcli device wifi connect "Ali" password 'aIi1379(aSadI'
-bash: syntax error near unexpected token `('
root@orangepizeroplus:~# ^C
root@orangepizeroplus:~# nmcli device wifi connect "Ali" password 'aIi1379!#(aSadI'

Device 'wlan0' successfully activated with 'da3e8a24-1ffb-47a8-9032-369d5463b68b'.
root@orangepizeroplus:~#
root@orangepizeroplus:~#
```

حال به اینترنت متصل شدیم با دستور زیر میتوانیم اطمینان پیدا کنیم

```
root@orangepizeroplus:~# nmcli -t -f active,ssid dev wifi | grep '^yes' | cut -d ':' -f2
Ali
root@orangepizeroplus:~#
```

اتصال خودکار به Wi-Fi

کد auto-connect.sh را مینویسیم در آخر گزارش همه‌ی کدها را توضیح میدهیم.

دستور زیر یعنی با ویرایشگر nano در مسیر root فایل شل را ویرایش کنیم

```
nano /root/auto-connect.sh
```

```
GNU nano 4.8                               /root/auto-connect.sh
#!/bin/bash

nano /root/auto-connect.sh
SSID="Ali"
PASSWORD="aIi1379!#(aSadI"

echo "----- $(date) -----" >> $LOG_FILE

CONNECTED_SSID=$(nmcli -t -f active,ssid dev wifi | grep '^yes' | cut -d ':' -f2)

if [ "$CONNECTED_SSID" = "$SSID" ]; then
    echo "Already connected to $SSID" >> $LOG_FILE
else
    echo "Not connected to $SSID. Trying to connect..." >> $LOG_FILE
    nmcli device wifi list >> $LOG_FILE
    nmcli device wifi connect "$SSID" password "$PASSWORD" >> $LOG_FILE 2>&1

    NEW_CONNECTED_SSID=$(nmcli -t -f active,ssid dev wifi | grep '^yes' | cut -d ':' -f2)
    if [ "$NEW_CONNECTED_SSID" = "$SSID" ]; then
        echo "Successfully connected to $SSID" >> $LOG_FILE
        [ Read 26 lines ]
[G] Get Help  [O] Write Out [W] Where Is  [K] Cut Text  [J] Justify  [C] Cur Pos
[X] Exit      [R] Read File  [\] Replace   [U] Paste Text[T] To Spell  [L] Go To Line
```

کد را در این محیط کپی میکنیم

بعد با **ctrl+o** کد را ذخیره و با **ctrl+x** از محیط خارج میشویم (تا اخر با ویرایشگر نانو هر فایلی را که باز کردیم این کار را انجام میدهیم)

با دستور زیر به فایل ایجاد شده دسترسی میدهیم

```
chmod +x /root/auto-connect.sh
```

حال با دنبال کردن دستور زیر اسکریپت را مینویسیم

```
crontab -e
```

```
/root/auto-connect.sh: line 7: $LOG_FILE: ambiguous redirect
/root/auto-connect.sh: line 12: $LOG_FILE: ambiguous redirect
/root/auto-connect.sh: line 26: $LOG_FILE: ambiguous redirect
root@orangepizeroplus:~# crontab -e
no crontab for root - using an empty one

Select an editor. To change later, run 'select-editor'.
 1. /bin/nano      <---- easiest
 2. /usr/bin/vim.basic

Choose 1-2 [1]: 1
```

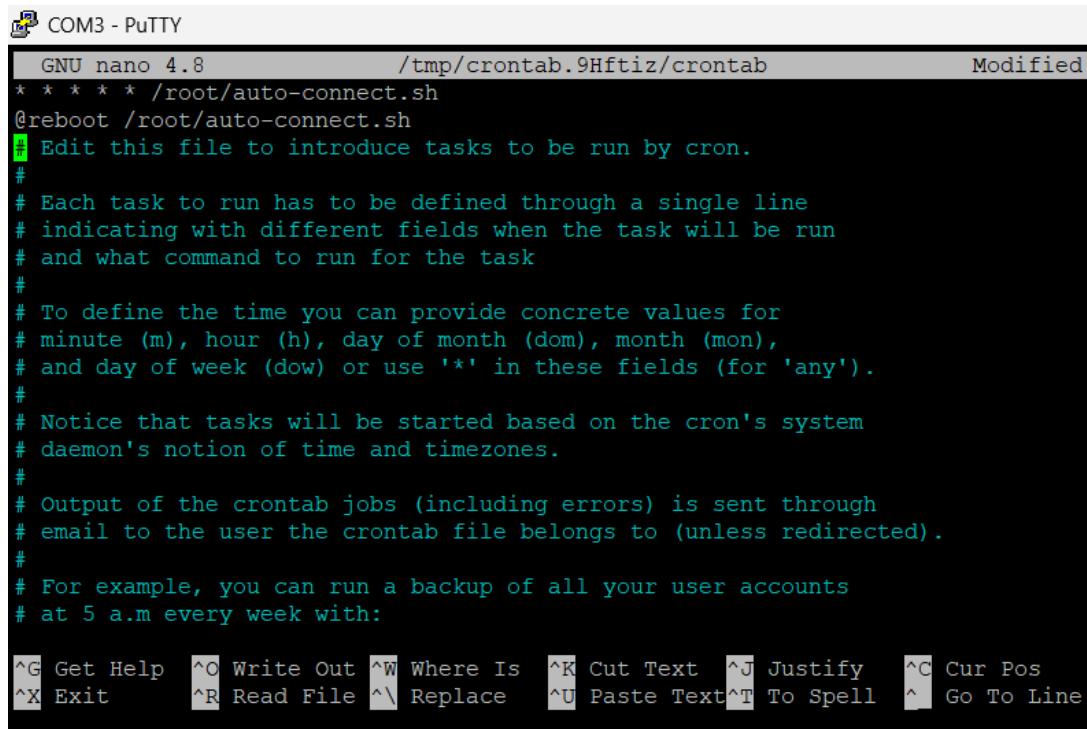
1 را انتخاب میکنیم که با ویرایشگر nano اسکریپت را بنویسیم.

```
* * * * * /root/auto-connect.sh
```

```
@reboot /root/auto-connect.sh
```

- خط اول: هر دقیقه اسکریپت اجرا می شود.

- خط دوم: یکبار هنگام بوت سیستم اسکریپت اجرا می شود.



```

GNU nano 4.8          /tmp/crontab.9Hftiz/crontab      Modified
* * * * * /root/auto-connect.sh
@reboot /root/auto-connect.sh
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:

```

Keyboard Shortcuts:

- ^G Get Help
- ^O Write Out
- ^W Where Is
- ^K Cut Text
- ^J Justify
- ^C Cur Pos
- ^X Exit
- ^R Read File
- ^L Replace
- ^U Paste Text
- ^T To Spell
- ^L Go To Line

با دستور `-l` crontab لیست دستورات را چک میکنیم که اطمینان پیدا کنیم دستوراتمان ذخیره شده است.

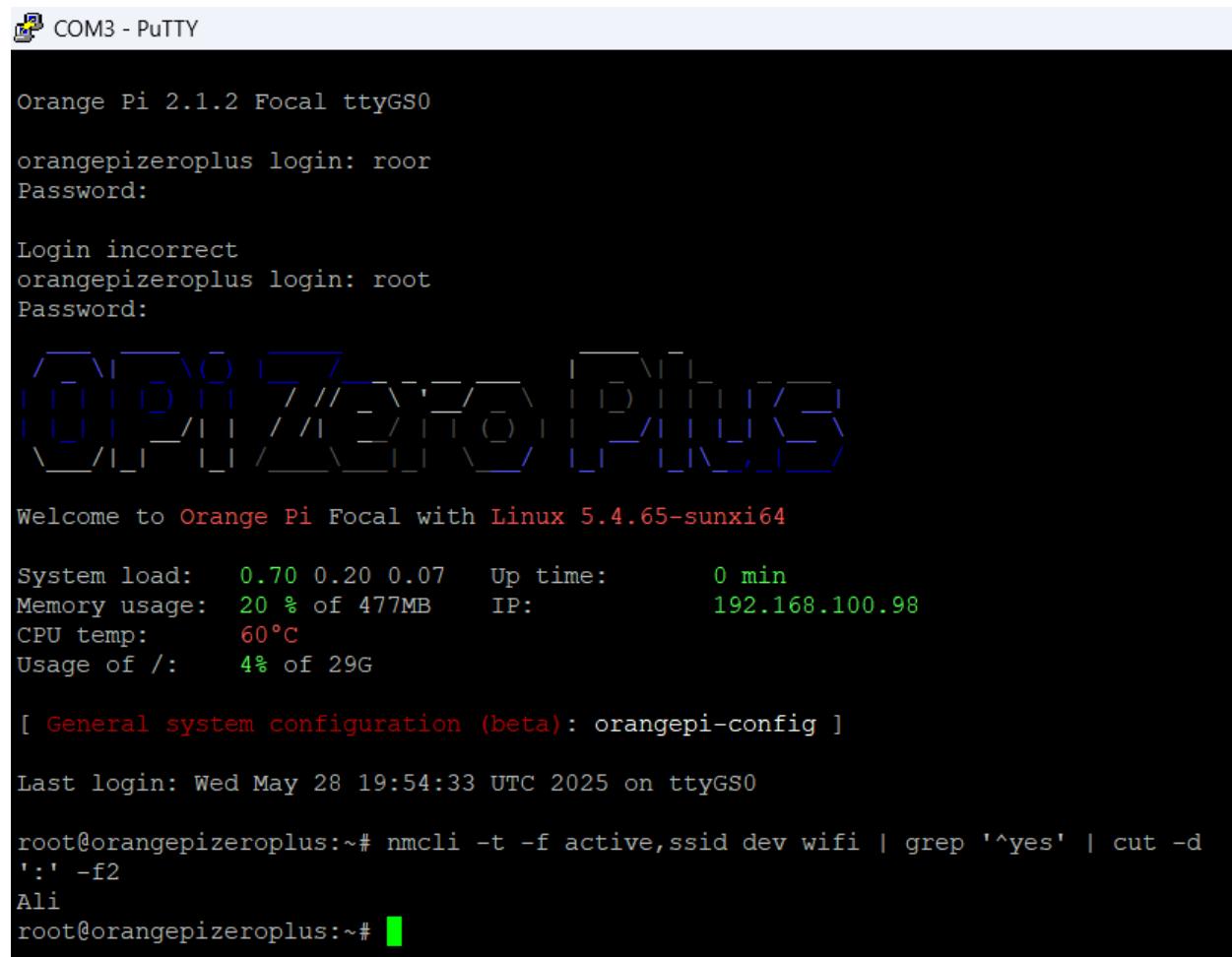
```

crontab: installing new crontab
root@orangezieroplus:~# crontab -l
* * * * * /root/auto-connect.sh
@reboot /root/auto-connect.sh
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h dom mon dow   command
root@orangezieroplus:~# 
```

وضعیت برد بعد از ریست کردن : ای پی اختصاص داده شده است که مشخص است خودکار به وای فای متصل شده است
با دستور زیر هم از وضعیت اتصال اطمینان پیدا میکنیم.

```
nmcli -t -f active,ssid dev wifi | grep '^yes' | cut -d ':' -f2
```

که پس از اجرا کردن دستور میگوید من به وای فای Ali متعلق هستم



COM3 - PuTTY

```
Orange Pi 2.1.2 Focal ttyGS0
orangepizeroplus login: roor
Password:

Login incorrect
orangepizeroplus login: root
Password:

Welcome to Orange Pi Focal with Linux 5.4.65-sunxi64

System load: 0.70 0.20 0.07 Up time: 0 min
Memory usage: 20 % of 477MB IP: 192.168.100.98
CPU temp: 60°C
Usage of /: 4% of 29G

[ General system configuration (beta): orangepi-config ]

Last login: Wed May 28 19:54:33 UTC 2025 on ttyGS0
root@orangepizeroplus:~# nmcli -t -f active,ssid dev wifi | grep '^yes' | cut -d ':' -f2
Ali
root@orangepizeroplus:~#
```

فایل log ذخیره شده در orangepi باز شده با برد

```
----- Wed 28 May 2025 08:16:31 PM UTC -----
Scanning available Wi-Fi networks...
IN-USE BSSID SSID MODE CHAN RATE SIGNAL BARS
SECURITY
* 48:FD:8E:84:95:F8 Ali Infra 1 130 Mbit/s 53 □_
WPA2
52:BE:E2:9F:38:38 Redmi Note 14 Infra 1 117 Mbit/s 44 □_
--
Already connected to Ali
-----
-----
----- Wed 28 May 2025 08:17:01 PM UTC -----
Scanning available Wi-Fi networks...
IN-USE BSSID SSID MODE CHAN RATE SIGNAL BARS
SECURITY
* 48:FD:8E:84:95:F8 Ali Infra 1 130 Mbit/s 54 □_
WPA2
52:BE:E2:9F:38:38 Redmi Note 14 Infra 1 117 Mbit/s 49 □_
--
Already connected to Ali
-----
-----
----- Wed 28 May 2025 08:17:07 PM UTC -----
Scanning available Wi-Fi networks...
IN-USE BSSID SSID MODE CHAN RATE SIGNAL BARS
SECURITY
* 48:FD:8E:84:95:F8 Ali Infra 1 130 Mbit/s 52 **
WPA2
52:BE:E2:9F:38:38 Redmi Note 14 Infra 1 117 Mbit/s 49 **
--
Already connected to Ali
-----
root@orangepireoplus:~#
```

حال فایل log هایی که از اتصال ا در orangepi ذخیره شده است را با دستور زیر از طریق cmd ویندوز در لپ تاپ کپی میکنیم تا فایل را در دسکتاپ داشته باشیم

```
scp root@192.168.100.98:/root/auto-connect.log C:\Users\Ali\Desktop\
```

```
C:\Users\Ali>scp root@192.168.100.98:/root/auto-connect.log C:\Users\Ali\Desktop\
root@192.168.100.98's password:
auto-connect.log
100% 173KB 1.8MB/s 00:00
C:\Users\Ali>
```

فایل auto-connect.log را در پوشه auto-wifi-connect تمرین میگذاریم و نمونه آن را با ویندوز هم باز میکنیم

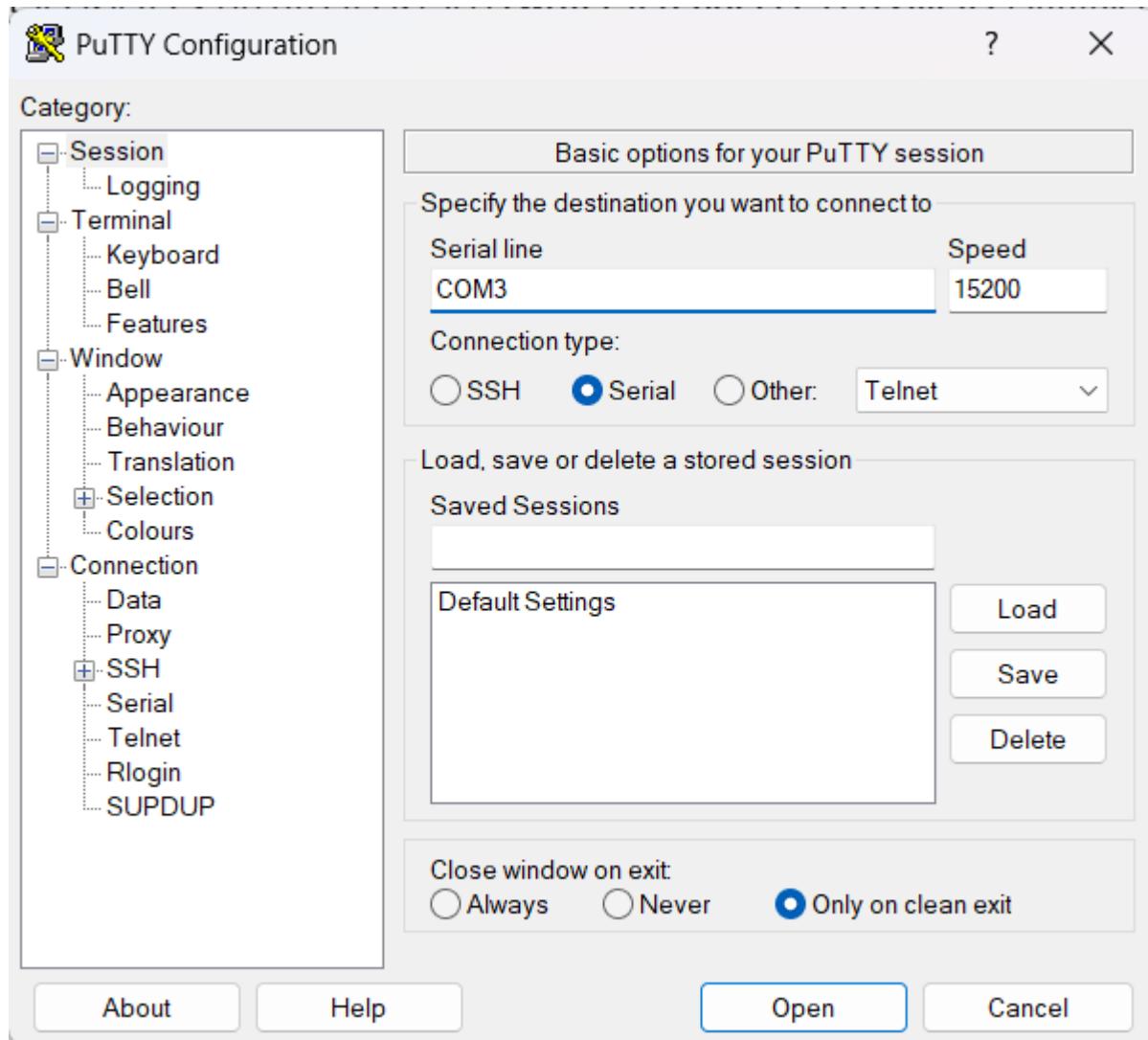
```
----- Wed 28 May 2025 08:17:01 PM UTC -----
Scanning available Wi-Fi networks...
IN-USE BSSID SSID MODE CHAN RATE SIGNAL BARS SECURITY
* 48:FD:8E:84:95:F8 Ali Infra 1 130 Mbit/s 54 ████ WPA2
  52:BE:E2:9F:38:38 Redmi Note 14 Infra 1 117 Mbit/s 49 ████ --
Already connected to Ali
-----

----- Wed 28 May 2025 08:17:07 PM UTC -----
Scanning available Wi-Fi networks...
IN-USE BSSID SSID MODE CHAN RATE SIGNAL BARS SECURITY
* 48:FD:8E:84:95:F8 Ali Infra 1 130 Mbit/s 52 ** WPA2
  52:BE:E2:9F:38:38 Redmi Note 14 Infra 1 117 Mbit/s 49 ** --
Already connected to Ali
-----

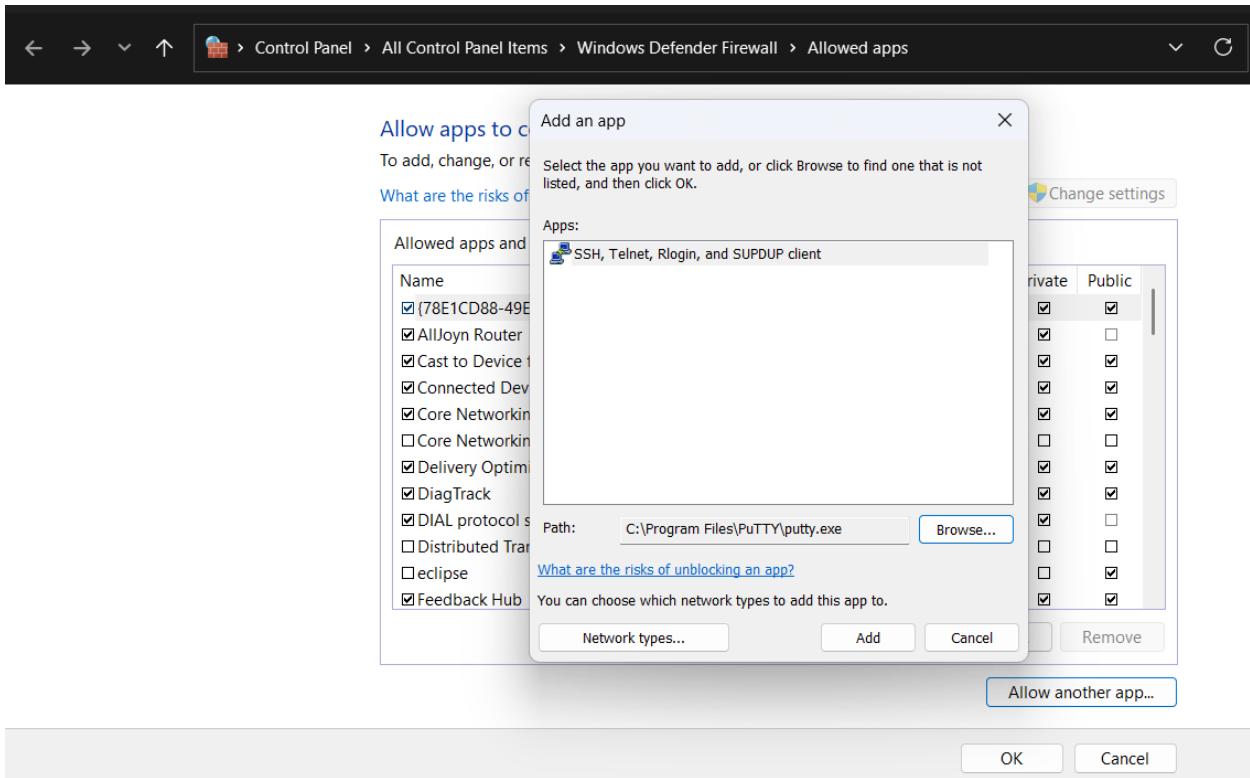
----- Wed 28 May 2025 08:18:01 PM UTC -----
Scanning available Wi-Fi networks...
IN-USE BSSID SSID MODE CHAN RATE SIGNAL BARS SECURITY
* 48:FD:8E:84:95:F8 Ali Infra 1 130 Mbit/s 52 ████ WPA2
  52:BE:E2:9F:38:38 Redmi Note 14 Infra 1 117 Mbit/s 49 ████ --
Already connected to Ali
```

مکانیزم های ارتباط بین فرآیندی

چند راه برای باز کردن دو ترمینال بود که ترجیح دادم ترمینال دوم رو با اتصال وای فای دستگاه باز کنم برای این کار باید ip دستگاه رو بدست بیاریم یعنی ترمینال اول :



برای اینکه فایروال ویندوز اتصال putty رو بلاک نکنه باید به ssh و putty دسترسی به اینترنت بدیم



حال برای اینکه بفهمیم با ip برد میتوانیم دیتا رد و بدل کنیم از دستور cmd ping در ویندوز استفاده میکنیم و میفهمیم که فایروال اتصال را بلاک نکرده و برد هم به اینترنت متصل هست. و در نتیجه لپ تاپ و برد با یکدیگر ارتباطی جز ارتباط usb دارند و میتوانیم ترمینال دوم را باز کنیم.

```
C:\Users\Ali>ping 192.168.100.98

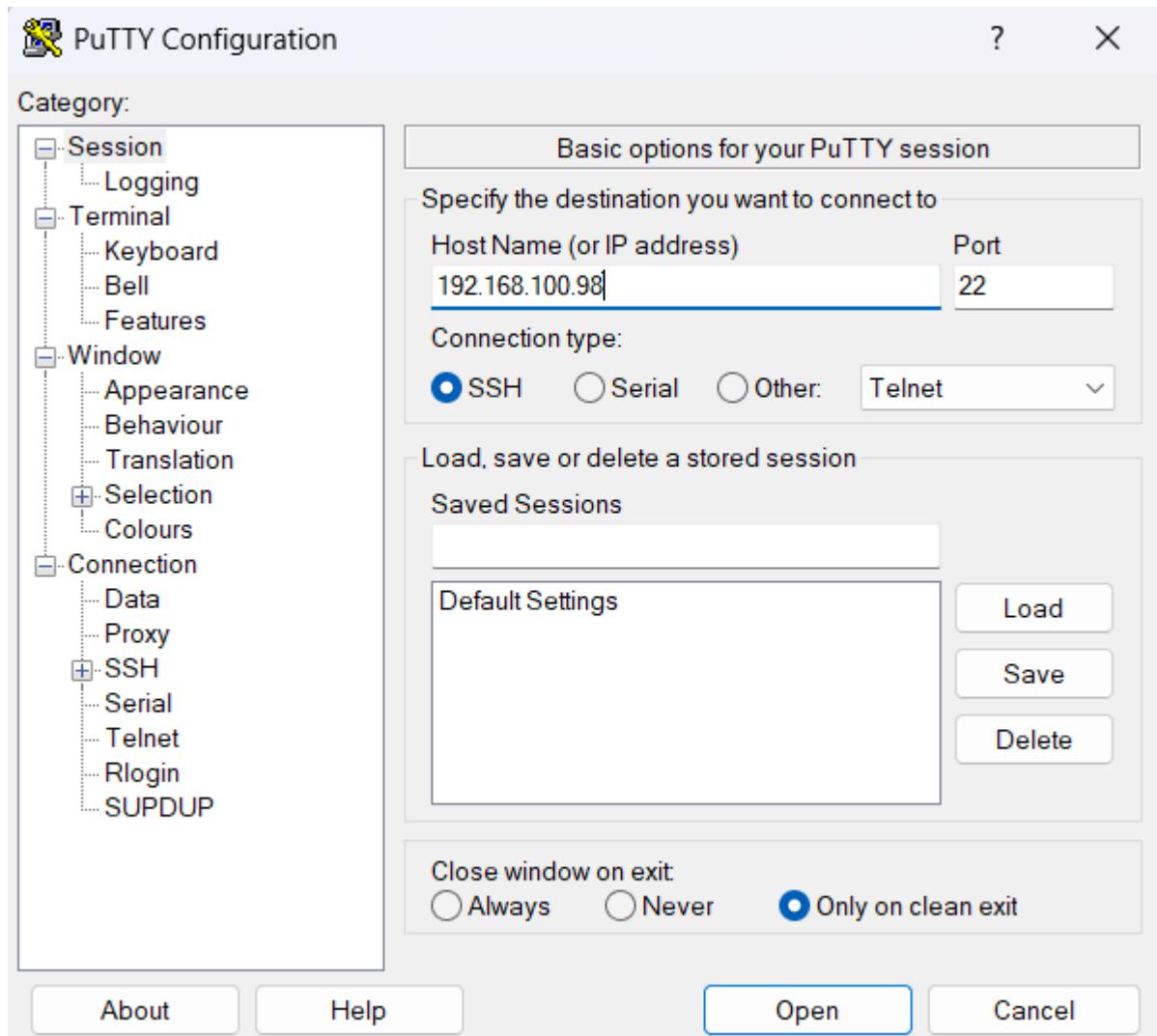
Pinging 192.168.100.98 with 32 bytes of data:
Reply from 192.168.100.98: bytes=32 time=1934ms TTL=64
Reply from 192.168.100.98: bytes=32 time=202ms TTL=64
Reply from 192.168.100.98: bytes=32 time=915ms TTL=64
Reply from 192.168.100.98: bytes=32 time=3ms TTL=64

Ping statistics for 192.168.100.98:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 3ms, Maximum = 1934ms, Average = 763ms

C:\Users\Ali>
```

حال در putty دوم

و در ترمینال دوم با تنظیمات زیر وارد میشویم پس یعنی با ترمینال دوم از طریق wifi متصل شده ایم.



root

orangepi



و تر مینیال دوم طبق تصویر زیر با موافقیت راه اندازی گردید.

```
root@orangepizeroplus: ~
root@orangepizeroplus: ~# login as: root
root@192.168.100.98's password:
Welcome to Orange Pi Focal with Linux 5.4.65-sunxi64

System load:  0.15 0.18 0.13   Up time:      11 min
Memory usage: 22 % of 477MB    IP:          192.168.100.99 192.168.100.98
CPU temp:     55°C
Usage of /:    4% of 29G

Last login: Thu May 29 07:55:30 2025
root@orangepizeroplus:~#
```

FIFO .1

حال کد هارا در ترمینال اول مینویسیم(البته فرقی ندارد)

کد fifo-reader.sh را مینویسیم.

```
nano /root/fifo-reader.sh
```

The screenshot shows a PuTTY terminal window titled "COM3 - PuTTY". The file being edited is "/root/fifo-reader.sh". The script content is as follows:

```
GNU nano 4.8          /root/fifo-reader.sh          Modified
#!/bin/bash

# مسیر FIFO
FIFO_PATH="/tmp/myfifo"

# اگر FIFO وجود نداشته، بساز
if [[ ! -p $FIFO_PATH ]]; then
    mkfifo $FIFO_PATH
fi

echo "Reader started. Waiting for messages..." # خواندن از FIFO و نمایش روی ترمینال

while true; do
    if read line < $FIFO_PATH; then
        echo "Received: $line"
    fi
done
```

At the bottom of the terminal window, there is a menu bar with the following options:

File Name to Write: /root/fifo-reader.sh	^G Get Help	M-D DOS Format	M-A Append	M-B Backup File
	^C Cancel	M-M Mac Format	M-P Prepend	^T To Files

و همینطور کد fifo-writer.sh را مینویسیم.

```
nano /root/fifo-writer.sh
```

GNU nano 4.8 /root/fifo-writer.sh

```
#!/bin/bash

# مسیر FIFO
FIFO_PATH="/tmp/myfifo"

# اگر FIFO وجود نداشت، بساز
if [[ ! -p $FIFO_PATH ]]; then
    mkfifo $FIFO_PATH
fi

echo "Writer started. Type messages below:"

# خواندن از ترمینال و نوشتن در FIFO
while true; do
    read input
    echo "$input" > $FIFO_PATH
done
```

root@orangepizeroplus:~#

و با دستور زیر به هردو کد نوشته شده اجازه دسترسی و اجرا میدهیم.

```
chmod +x /root/fifo-reader.sh /root/fifo-writer.sh
```

```
root@orangepizeroplus:~# chmod +x /root/fifo-reader.sh /root/fifo-writer.sh
root@orangepizeroplus:~#
```

کد زیر برای اطمینان از اتصال برد به اینترنت است

```
ip addr show wlan0
```

```
root@orangepizeroplus:~# ip addr show wlan0
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 12:01:1f:12:86:62 brd ff:ff:ff:ff:ff:ff
        inet 192.168.100.98/24 brd 192.168.100.255 scope global dynamic noprefixroute
          valid_lft 85483sec preferred_lft 85483sec
        inet6 fe80::e4d3:8f7d:e497:8a97/64 scope link noprefixroute
          valid_lft forever preferred_lft forever
root@orangepizeroplus:~#
```

و کد زیر برای اطمینان از دسترسی و اجرای فایل های شل

```
root@orangepizeroplus:~# systemctl status ssh
● ssh.service - OpenBSD Secure Shell server
  Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)
  Active: active (running) since Wed 2025-05-28 20:18:30 UTC; 10h ago
    Docs: man:sshd(8)
           man:sshd_config(5)
   Process: 1102 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)
    Main PID: 1149 (sshd)
      Tasks: 1 (limit: 403)
     Memory: 2.6M
      CGroup: /system.slice/ssh.service
              └─1149 sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups

May 28 20:18:29 orangepizeroplus systemd[1]: Starting OpenBSD Secure Shell serv>
May 28 20:18:30 orangepizeroplus sshd[1149]: Server listening on 0.0.0.0 port 22.>
May 28 20:18:30 orangepizeroplus sshd[1149]: Server listening on :: port 22.
May 28 20:18:30 orangepizeroplus systemd[1]: Started OpenBSD Secure Shell serve>
lines 1-16/16 (END)
```

حال برای دیدن پیام های دریافتی کد زیر را در ترمینال دوم وارد میکنیم

```
./fifo-reader.sh
```

و برای نوشتن پیام ها کد زیر را در ترمینال اول وارد میکنیم.

```
./fifo-writer.sh
```

ترمینال 1 فرستنده :

```
root@orange-pizero-plus:~# ./fifo-writer.sh
Writer started. Type messages below:
salam
mokhlesam Ali Asadi 403203919    :)
balakhare kar kard :)
salam
ali
salam
alihasatam
ali 1379 asadi
manam ali
403203919
ALI
SALAM
SALAM
ALI HASTAM
403203919
54
55
23
```

ترمیمال 2 گیرنده :

```
root@orangepizeroplus:~# ./fifo-reader.sh
Reader started. Waiting for messages...
Received: salam
Received: mokhlesam Ali Asadi 403203919    :)
Received: balakhare kar kard :)
Received: salam
Received: ali
Received: salam
Received: alihasatam
salamReceived: ali 1379 asadi

Received: manam ali
Received: 403203919
Received: ALI
Received: SALAM
Received: SALAM
Received: ALI HASTAM
Received: 403203919
Received: 54
Received: 55
Received: 23
```

دو ترمیمال کنار یکدیگر :

The image shows two PuTTY terminal windows side-by-side. The left window, titled 'COM3 - PuTTY', displays system status and network configuration for an Orange Pi Focal system. The right window, titled 'root@orangepizeroplus: ~', shows a FIFO reader/writer application exchange. The FIFO reader (left) sends messages like 'salam', 'mokhlesam Ali Asadi 403203919 :)', and 'balakhare kar kard :)' to the FIFO writer (right). The FIFO writer (right) responds with 'Received: salam', 'Received: mokhlesam Ali Asadi 403203919 :)', 'Received: balakhare kar kard :)', and other variations of 'Received: [message]'. Both windows show a small decorative graphic at the top.

```

System load: 0.41 0.25 0.10 Up time: 2 min
Memory usage: 20 % of 477MB IP: 192.168.100.98
CPU temp: 56°C
Usage of /: 4% of 29G

Last login: Wed May 28 20:17:22 UTC 2025 on ttyGS0

root@orangepizeroplus:~# ip addr show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 02:01:1f:12:86:62 brd ff:ff:ff:ff:ff:ff
        inet 192.168.100.99/24 brd 192.168.100.255 scope global dynamic noprefixroute
            valid_lft 86364sec preferred_lft 86364sec
        inet6 fe80::1978:f7ca:4a44:6738/64 scope link noprefixroute
            valid_lft forever preferred_lft forever
root@orangepizeroplus:~# ip addr show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 02:01:1f:12:86:62 brd ff:ff:ff:ff:ff:ff
        inet 192.168.100.99/24 brd 192.168.100.255 scope global dynamic noprefixroute
            valid_lft 86148sec preferred_lft 86148sec
        inet6 fe80::1978:f7ca:4a44:6738/64 scope link noprefixroute
            valid_lft forever preferred_lft forever
root@orangepizeroplus:~# ip addr show wlan0
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 12:01:1f:12:86:62 brd ff:ff:ff:ff:ff:ff
        inet 192.168.100.98/24 brd 192.168.100.255 scope global dynamic noprefixroute
            valid_lft 85862sec preferred_lft 85862sec
        inet6 fe80::e4d3:8f7d:e497:8a97/64 scope link noprefixroute
            valid_lft forever preferred_lft forever
root@orangepizeroplus:~# ./fifo-reader.sh
Writer started. Type messages below:
salam
mokhlesam Ali Asadi 403203919 :)
balakhare kar kard :)
salam
ali
salam
alihasatam
ali 1379 asadi
manam ali
403203919
ALI
SALAM
SALAM
ALI HASTAM
403203919
54
55
23
[green square icon]

root@orangepizeroplus:~# ./fifo-writer.sh
login as: root
root@192.168.100.98's password:
Welcome to Orange Pi Focal with Linux 5.4.65-sunxi64
System load: 0.15 0.18 0.13 Up time: 11 min
Memory usage: 22 % of 477MB IP: 192.168.100.99 192.168.100.98
CPU temp: 55°C
Usage of /: 4% of 29G

Last login: Thu May 29 07:55:30 2025

root@orangepizeroplus:~# ./fifo-reader.sh
Reader started. Waiting for messages...
Received: salam
Received: mokhlesam Ali Asadi 403203919 :)
Received: balakhare kar kard :)
Received: salam
Received: ali
Received: salam
Received: alihasatam
Received: ali 1379 asadi
Received: manam ali
Received: 403203919
Received: ALI
Received: SALAM
Received: SALAM
Received: ALI HASTAM
Received: 403203919
Received: 54
Received: 55
Received: 23
[green square icon]

```

مشاهده میکنیم که هر پیامی که فرستنده میفرستد بلافاصله گیرنده پیام را دریافت میکند.

2. سیگنال های لینوکس

برای مرتب شدن مکان کد ها دو پوشه به نام های myprojects/signals میسازیم و واردشان میشویم.

Mkdir -p /root/myprojects/signals

cd /root/myprojects/signals

با دستور زیر وقتی در پوشه ای ایجاد کردیم هستیم کد signal-receiver.cpp را مینویسیم.

nano signal-receiver.cpp

GNU nano 4.8 signal-receiver.cpp Modified

```
int main() {
    std::cout << "PID: " << getpid() << std::endl;
    for (int i = 1; i < 32; ++i) {
        signal(i, signal_handler);
    }
    while (true) {
        pause(); // wait for signal
    }
    return 0;
}
```

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit ^R Read File ^\ Replace ^U Paste Text ^T To Spell ^ Go To Line

پس از ذخیره کد نوشته شده مان
حال کد signal-sender.cpp را هم مینویسیم.

nano signal-sender.cpp

```
GNU nano 4.8          signal-sender.cpp
int sig = std::stoi(argv[2]);

if (kill(pid, sig) == -1) {
    std::cerr << "Failed to send signal: " << strerror(errno) << std::endl;
    return 1;
}

std::cout << "Signal " << sig << " sent to PID " << pid << std::endl;
return 0;
}

root@orangepizeroplus:~/myprojects/signals#
```

حال در همان پوشه یک makefile هم میسازیم

یک فایل متند ساده است که توسط ابزار make خونده می شه.

کارش اینه که فرایند کامپایل و ساخت (build) برنامه رو خودکار کنه.

مثالاً بجای اینکه هر بار دستی این دستور رو بنویسم:

```
g++ -Wall -std=c++11 signal-sender.cpp -o signal-sender
```

فقط کافیه بزنیم:

Make

و خودش همه چیز رو طبق Makefile کامپایل می کنه.

: makefile کد

CXX = g++

CXXFLAGS = -Wall -std=c++11

all: signal-sender signal-receiver

signal-sender: signal-sender.cpp

`$(CXX) $(CXXFLAGS) signal-sender.cpp -o signal-sender`

signal-receiver: signal-receiver.cpp

`$(CXX) $(CXXFLAGS) signal-receiver.cpp -o signal-receiver`

clean:

`rm -f signal-sender signal-receiver received-signals.log`

خط

توضیح

CXX = g++

مشخص می کنند که کامپایلر C++ مورد استفاده همون g++ هست

CXXFLAGS = -Wall -std=c++11

تعیین می کنند که هنگام کامپایل، هشدارها (-Wall) فعال باشند و استاندارد C++11 استفاده شود

بشه

all: signal-sender signal-receiver

هدف پیشفرض: یعنی وقتی فقط make را بزنیم، هر دوی signal- و signal-sender را ساخته می شنیم

signal-sender: signal-sender.cpp

اگر فایل signal-sender.cpp تغییر کرده، برنامه اجرایی signal-sender را دوباره بساز

`$(CXX) $(CXXFLAGS) signal-sender.cpp -o signal-sender`

همون دستور g++ برای کامپایل signal-sender.cpp را خروجی signal-sender می کند

توضیح

خط

signal-receiver: signal-receiver.cpp

مشابه بالا برای گیرنده

clean:

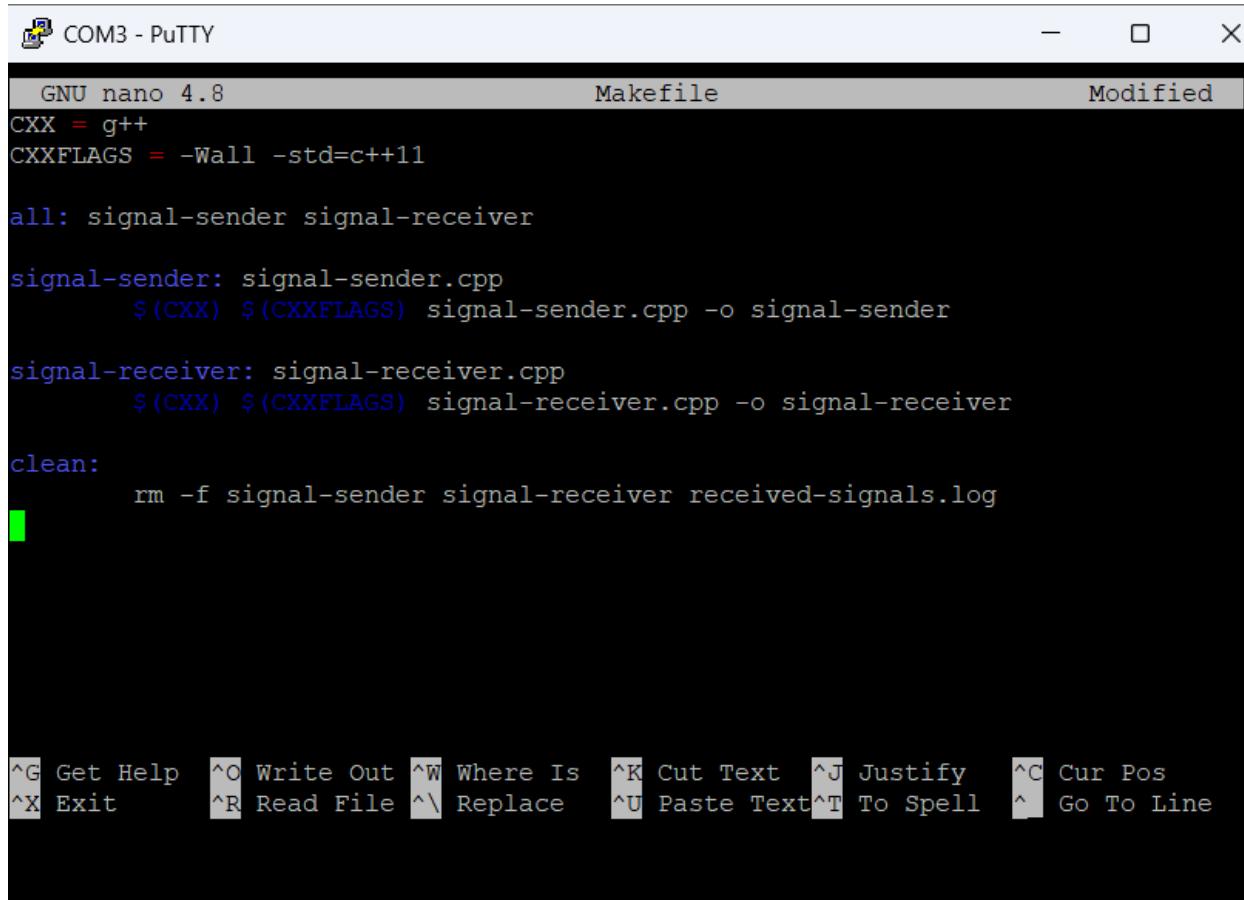
یک هدف اختیاری برای حذف فایل‌های اجرایی و لاغها

rm -f signal-sender signal-receiver received-signals.log

دستور پاک کردن فایل‌ها

حال فایل را میسازیم.

nano Makefile



```
GNU nano 4.8                                     Makefile                                         Modified
CXX = g++
CXXFLAGS = -Wall -std=c++11

all: signal-sender signal-receiver

signal-sender: signal-sender.cpp
    $(CXX) $(CXXFLAGS) signal-sender.cpp -o signal-sender

signal-receiver: signal-receiver.cpp
    $(CXX) $(CXXFLAGS) signal-receiver.cpp -o signal-receiver

clean:
    rm -f signal-sender signal-receiver received-signals.log

^G Get Help  ^O Write Out  ^W Where Is  ^K Cut Text  ^J Justify  ^C Cur Pos
^X Exit      ^R Read File  ^\ Replace   ^U Paste Text ^T To Spell  ^_ Go To Line
```

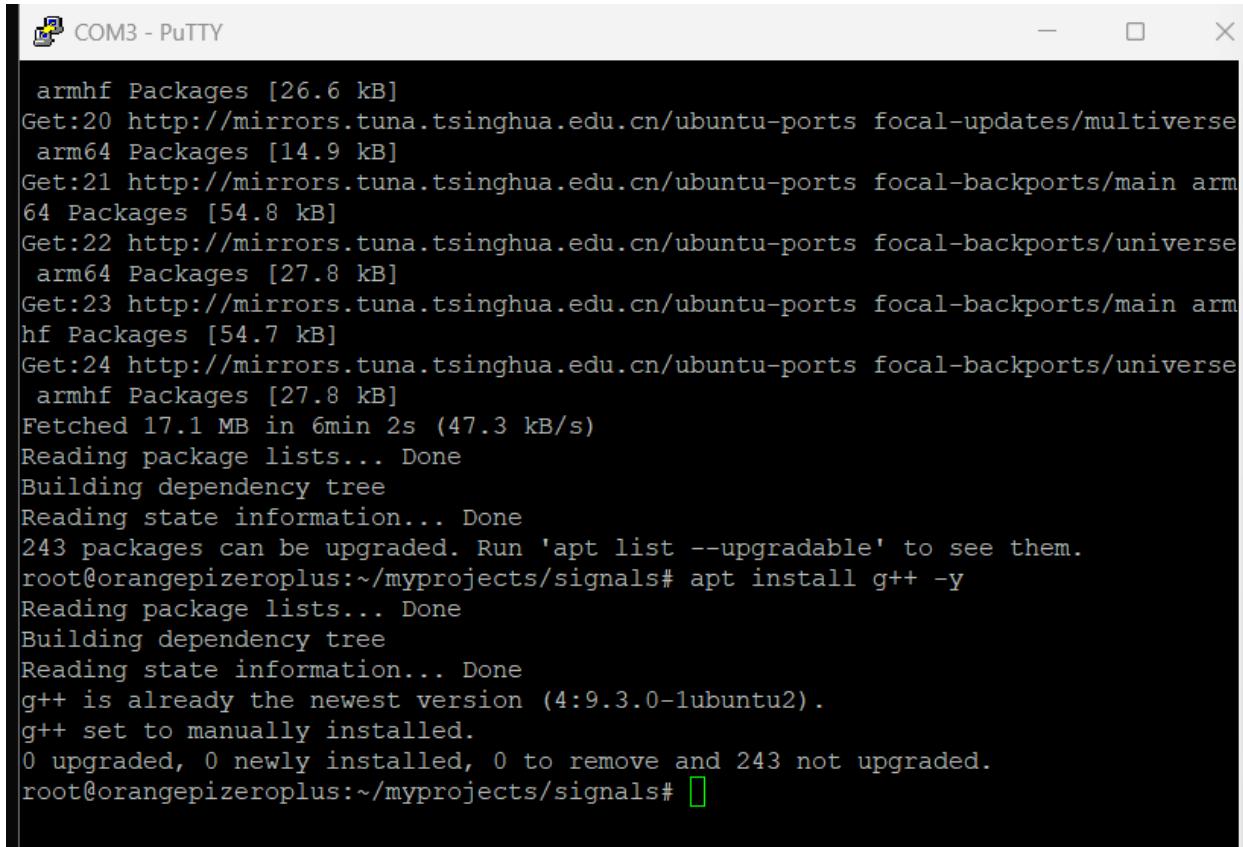
با دستورات زیر update میکنیم و کامپایلر g++ را نصب میکنیم.

```
apt update
```

```
apt install g++ -y
```

```
root@orangepileroplus:~/myprojects/signals# apt update
Hit:1 http://mirrors.tuna.tsinghua.edu.cn/ubuntu-ports focal InRelease
Get:2 http://mirrors.tuna.tsinghua.edu.cn/ubuntu-ports focal-security InRelease
[128 kB]
0% [2 InRelease 26.3 kB/128 kB 21%]
```

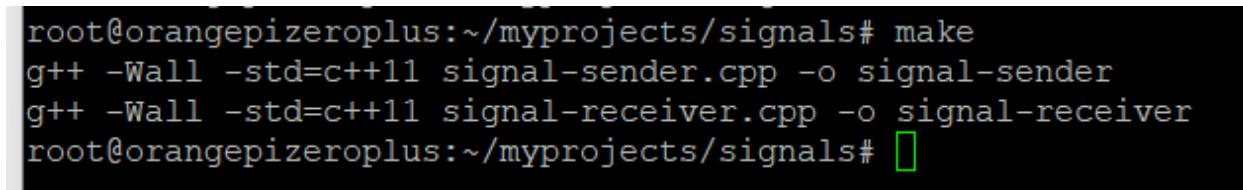
```
Hit:1 http://mirrors.tuna.tsinghua.edu.cn/ubuntu-ports focal InRelease
Get:2 http://mirrors.tuna.tsinghua.edu.cn/ubuntu-ports focal-security InRelease
[128 kB]
Get:3 http://mirrors.tuna.tsinghua.edu.cn/ubuntu-ports focal-updates InRelease [
128 kB]
Get:4 http://mirrors.tuna.tsinghua.edu.cn/ubuntu-ports focal-backports InRelease
[128 kB]
Get:5 http://mirrors.tuna.tsinghua.edu.cn/ubuntu-ports focal-security/universe a
rm64 Packages [1,215 kB]
Get:6 http://mirrors.tuna.tsinghua.edu.cn/ubuntu-ports focal-security/restricted
armhf Packages [26.1 kB]
Get:7 http://mirrors.tuna.tsinghua.edu.cn/ubuntu-ports focal-security/multiverse
arm64 Packages [8,083 B]
Get:8 http://mirrors.tuna.tsinghua.edu.cn/ubuntu-ports focal-security/main armhf
Packages [1,857 kB]
Get:9 http://mirrors.tuna.tsinghua.edu.cn/ubuntu-ports focal-security/multiverse
armhf Packages [6,243 B]
Get:10 http://mirrors.tuna.tsinghua.edu.cn/ubuntu-ports focal-security/universe
armhf Packages [996 kB]
Get:11 http://mirrors.tuna.tsinghua.edu.cn/ubuntu-ports focal-security/restricte
d arm64 Packages [77.0 kB]
Get:12 http://mirrors.tuna.tsinghua.edu.cn/ubuntu-ports focal-security/main arm6
4 Packages [3,419 kB]
37% [12 Packages 204 kB/3,419 kB 6%] 89.8 kB/s 2min 17s
```



```
armhf Packages [26.6 kB]
Get:20 http://mirrors.tuna.tsinghua.edu.cn/ubuntu-ports focal-updates/multiverse arm64 Packages [14.9 kB]
Get:21 http://mirrors.tuna.tsinghua.edu.cn/ubuntu-ports focal-backports/main arm64 Packages [54.8 kB]
Get:22 http://mirrors.tuna.tsinghua.edu.cn/ubuntu-ports focal-backports/universe arm64 Packages [27.8 kB]
Get:23 http://mirrors.tuna.tsinghua.edu.cn/ubuntu-ports focal-backports/main armhf Packages [54.7 kB]
Get:24 http://mirrors.tuna.tsinghua.edu.cn/ubuntu-ports focal-backports/universe armhf Packages [27.8 kB]
Fetched 17.1 MB in 6min 2s (47.3 kB/s)
Reading package lists... Done
Building dependency tree
Reading state information... Done
243 packages can be upgraded. Run 'apt list --upgradable' to see them.
root@orangepizeroplus:~/myprojects/signals# apt install g++ -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
g++ is already the newest version (4:9.3.0-1ubuntu2).
g++ set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 243 not upgraded.
root@orangepizeroplus:~/myprojects/signals# 
```

حال با دستور make دو فایل c++ را میسازیم و build میکنیم.

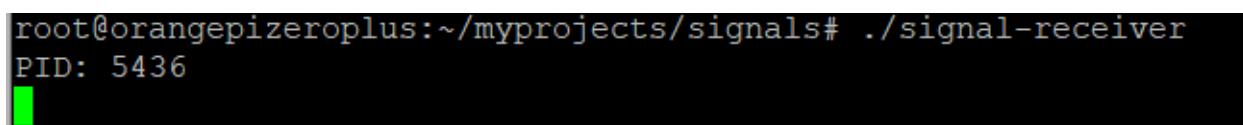
Make



```
root@orangepizeroplus:~/myprojects/signals# make
g++ -Wall -std=c++11 signal-sender.cpp -o signal-sender
g++ -Wall -std=c++11 signal-receiver.cpp -o signal-receiver
root@orangepizeroplus:~/myprojects/signals# 
```

حال در ترمینال دوم فایل گیرنده را اجرا میکنیم تا کد PID گیرنده رو بهمون بده

./signal-receiver



```
root@orangepizeroplus:~/myprojects/signals# ./signal-receiver
PID: 5436
root@orangepizeroplus:~/myprojects/signals# 
```

و با دستور زیر در ترمینال اول دستورات لینوکسی که میخواهیم را با استفاده از کد PID گیرنده میفرستیم.

```
./signal-sender 5436 2
```

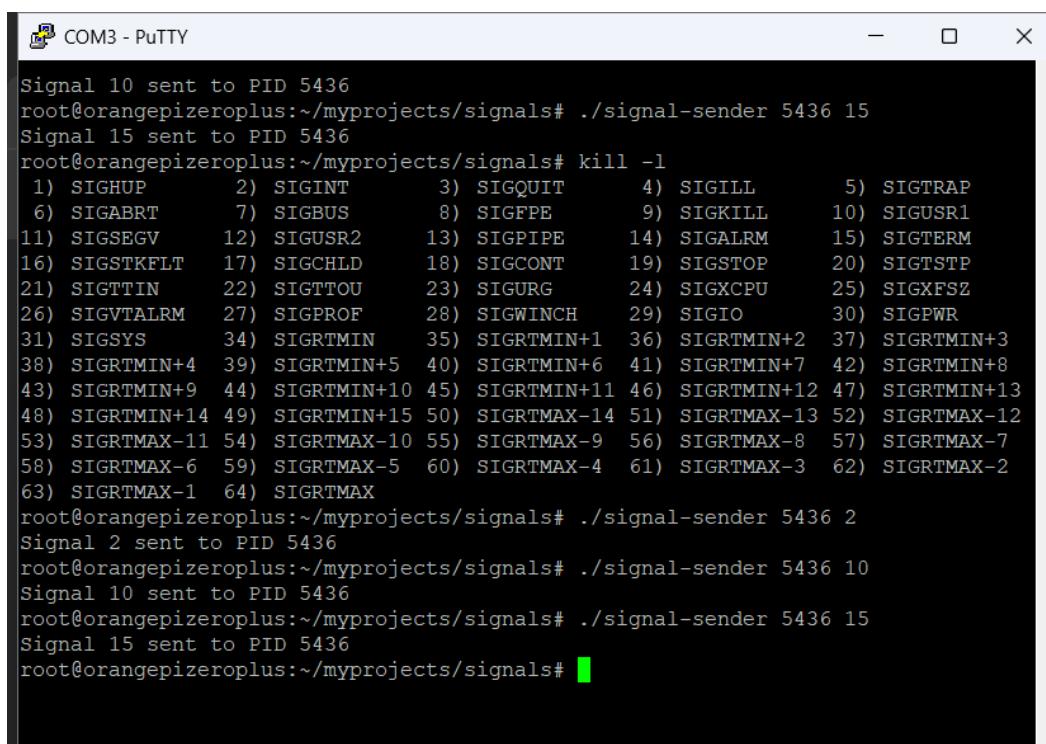
ترمینال 1 فرستنده :

```
root@orangepizeroplus:~/myprojects/signals# ./signal-sender 5436 2
Signal 2 sent to PID 5436
root@orangepizeroplus:~/myprojects/signals# ./signal-sender 5436 10
Signal 10 sent to PID 5436
root@orangepizeroplus:~/myprojects/signals# ./signal-sender 5436 15
Signal 15 sent to PID 5436
```

ترمینال 2 گیرنده:

```
root@orangepizeroplus:~/myprojects/signals# ./signal-receiver
PID: 5436
Received signal: 2 (Interrupt)
Received signal: 10 (User defined signal 1)
Received signal: 15 (Terminated)
```

برای دیدن کل دستورات لینوکسی : kill -l دستور



```
Signal 10 sent to PID 5436
root@orangepizeroplus:~/myprojects/signals# ./signal-sender 5436 15
Signal 15 sent to PID 5436
root@orangepizeroplus:~/myprojects/signals# kill -l
 1) SIGHUP      2) SIGINT      3) SIGQUIT      4) SIGILL      5) SIGTRAP
 6) SIGABRT     7) SIGBUS      8) SIGFPE       9) SIGKILL     10) SIGUSR1
 11) SIGSEGV    12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
 16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
 21) SIGTTIN    22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
 26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
 31) SIGSYS     34) SIGRTMIN    35) SIGRTMIN+1   36) SIGRTMIN+2   37) SIGRTMIN+3
 38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6   41) SIGRTMIN+7   42) SIGRTMIN+8
 43) SIGRTMIN+9  44) SIGRTMIN+10  45) SIGRTMIN+11  46) SIGRTMIN+12  47) SIGRTMIN+13
 48) SIGRTMIN+14 49) SIGRTMIN+15  50) SIGRTMAX-14  51) SIGRTMAX-13  52) SIGRTMAX-12
 53) SIGRTMAX-11 54) SIGRTMAX-10  55) SIGRTMAX-9   56) SIGRTMAX-8   57) SIGRTMAX-7
 58) SIGRTMAX-6  59) SIGRTMAX-5   60) SIGRTMAX-4   61) SIGRTMAX-3   62) SIGRTMAX-2
 63) SIGRTMAX-1  64) SIGRTMAX
root@orangepizeroplus:~/myprojects/signals# ./signal-sender 5436 2
Signal 2 sent to PID 5436
root@orangepizeroplus:~/myprojects/signals# ./signal-sender 5436 10
Signal 10 sent to PID 5436
root@orangepizeroplus:~/myprojects/signals# ./signal-sender 5436 15
Signal 15 sent to PID 5436
root@orangepizeroplus:~/myprojects/signals#
```

ترمینال 1 فرستنده و ترمینال 2 گیرنده کنار یکدیگر :

The image shows two terminal windows side-by-side. The left window, titled 'COM3 - PuTTY', is running on an Orange Pi Zero Plus. It displays the output of a script that sends signals to a process with PID 5436. The right window, titled 'root@orangepizeroplus: ~ /myprojects/signals', is also running on an Orange Pi Zero Plus. It shows a script that receives signals and prints them to the console. The two terminals are connected via a pipe, demonstrating signal transmission between different hosts.

```
Signal 10 sent to PID 5436
root@orangepizeroplus:~/myprojects/signals# ./signal-sender 5436 15
Signal 15 sent to PID 5436
root@orangepizeroplus:~/myprojects/signals# kill -1
1) SIGHUP      2) SIGINT      3) SIGQUIT     4) SIGILL      5) SIGTRAP
6) SIGABRT     7) SIGBUS      8) SIGFPE      9) SIGKILL     10) SIGUSR1
11) SIGSEGV    12) SIGUSR2    13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD    18) SIGCONT    19) SIGSTOP     20) SIGTSTP
21) SIGTTIN    22) SIGTTOUT   23) SIGURG     24) SIGXCPU    25) SIGXFSZ
26) SIGVTALRM  27) SIGPROF    28) SIGWINCH   29) SIGIO      30) SIGPWR
31) SIGSYS     34) SIGRTMIN   35) SIGRTMIN+1 36) SIGRTMIN+2 37) SIGRTMIN+3
38) SIGRTMIN+4 39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 42) SIGRTMIN+8
43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9 56) SIGRTMAX-8 57) SIGRTMAX-7
58) SIGRTMAX-6 59) SIGRTMAX-5 60) SIGRTMAX-4 61) SIGRTMAX-3 62) SIGRTMAX-2
63) SIGRTMAX-1 64) SIGRTMAX

root@orangepizeroplus:~/myprojects/signals# ./signal-sender 5436 2
Signal 2 sent to PID 5436
root@orangepizeroplus:~/myprojects/signals# ./signal-sender 5436 10
Signal 10 sent to PID 5436
root@orangepizeroplus:~/myprojects/signals# ./signal-sender 5436 15
Signal 15 sent to PID 5436
root@orangepizeroplus:~/myprojects/signals#
```

```
Welcome to Orange Pi Focal with Linux 5.4.65-sunxi64

System load: 0.15 0.18 0.13 Up time: 11 min
Memory usage: 22 % of 477MB IP: 192.168.100.99 192.168.100.98
CPU temp: 55°C
Usage of /: 4% of 29G

Last login: Thu May 29 07:55:30 2025

root@orangepizeroplus:~# ./fifo-reader.sh
Reader started. Waiting for messages...
Received: salam
Received: mokhlesam Ali Asadi 403203919 :)
Received: balakhare kar kard :)
Received: salam
Received: ali
Received: salam
Received: alihasatam
salamReceived: ali 1379 asadi

Received: manam ali
Received: 403203919
Received: ALI
Received: SALAM
Received: SALAM
Received: ALI HASTAM
Received: 403203919
Received: 54
Received: 55
Received: 23
Received:
Received:
^C
root@orangepizeroplus:~# ./signal-receiver
-bash: ./signal-receiver: No such file or directory
root@orangepizeroplus:~# cd /root/myprojects/signals
root@orangepizeroplus:~/myprojects/signals# ^C
root@orangepizeroplus:~/myprojects/signals# ./signal-receiver
-bash: ./signal-receiver: No such file or directory
root@orangepizeroplus:~/myprojects/signals# ./signal-receiver
PID: 5436
Received signal: 2 (Interrupt)
Received signal: 10 (User defined signal 1)
Received signal: 15 (Terminated)
Received signal: 2 (Interrupt)
Received signal: 10 (User defined signal 1)
Received signal: 15 (Terminated)
```

ترمینال 1 فرستنده :

```
root@orangepizeroplus:~/myprojects/signals# ./signal-sender
Usage: ./signal-sender <PID> <SIGNAL NUMBER>
root@orangepizeroplus:~/myprojects/signals# ./signal-sender 6740 20
Signal 20 sent to PID 6740
root@orangepizeroplus:~/myprojects/signals#
```

تر مینال 2 گیرنده:

```
root@orangepizeroplus:~/myprojects/signals# ./signal-receiver
PID: 6740
./signal-receiver
Received signal: 20 (Stopped)
```

با ارسال دستور زیر به گیرنده و سپس **Ctrl+c** در فرستنده از این حالت خارج میشویم تا کار دیگری با برد

انجام بدهیم.

```
kill -9 <PID>
```

```
kill -9 6740
```

این **SIGKILL** میفرسته که غیرقابل مدیریته و همیشه برنامه رو میبنده.

3. حافظه اشتراکی

دو پوشه **myprojects/shm** میسازیم و به آنها وارد میشویم:

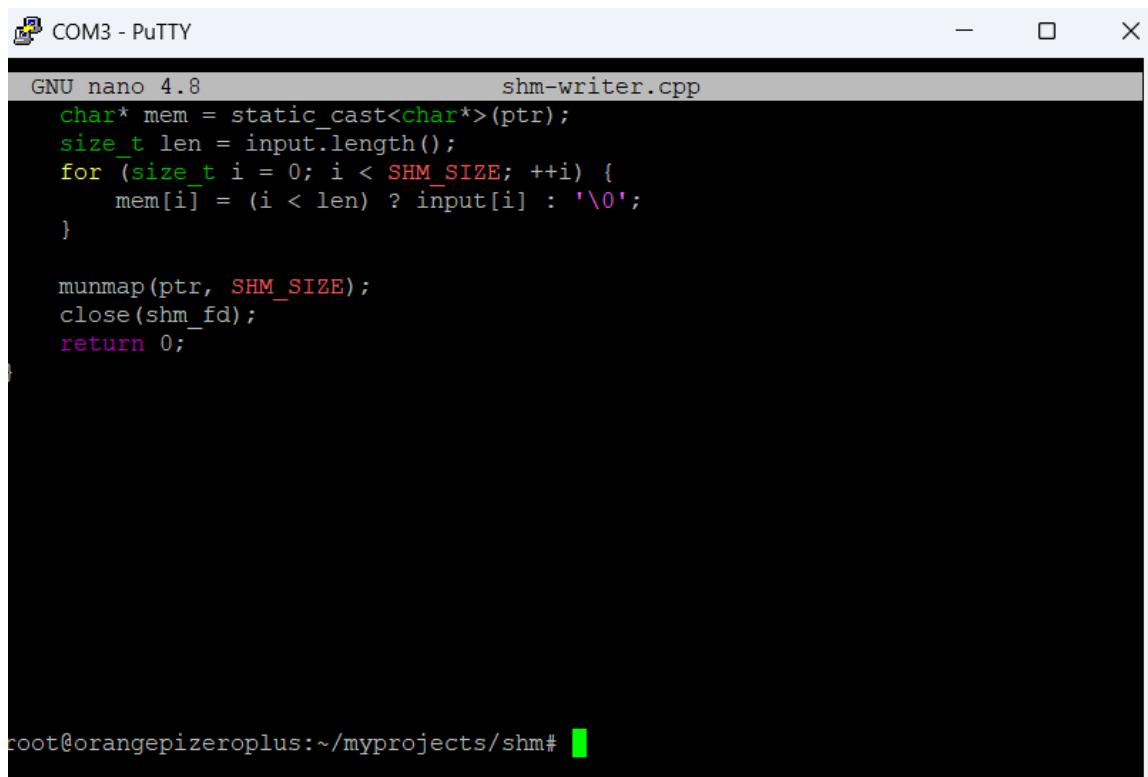
```
mkdir -p /root/myprojects/shm
```

```
cd /root/myprojects/shm
```

```
root@orangeplus:~/myprojects/signals# mkdir -p /root/myprojects/shm
root@orangeplus:~/myprojects/signals# cd /root/myprojects/shm
root@orangeplus:~/myprojects/shm#
```

کد **shm-writer.cpp** را مینویسیم.

```
nano shm-writer.cpp
```



The screenshot shows a PuTTY terminal window titled "COM3 - PuTTY". The window displays the source code for "shm-writer.cpp" using the nano editor. The code is as follows:

```
GNU nano 4.8          shm-writer.cpp
char* mem = static_cast<char*>(ptr);
size_t len = input.length();
for (size_t i = 0; i < SHM_SIZE; ++i) {
    mem[i] = (i < len) ? input[i] : '\0';
}

munmap(ptr, SHM_SIZE);
close(shm_fd);
return 0;
}
```

At the bottom of the terminal window, the prompt "root@orangeplus:~/myprojects/shm#" is visible.

کد `shm-reader.cpp` را مینویسیم:

nano shm-reader.cpp



```
GNU nano 4.8          shm-reader.cpp
}

char buffer[SHM_SIZE + 1] = {0};
memcpy(buffer, ptr, SHM_SIZE);
std::cout << "SHM Content: \\" << buffer << "\"" << std::endl;

munmap(ptr, SHM_SIZE);
close(shm_fd);
return 0;
}

root@orangepizeroplus:~/myprojects/shm# 
```

فایل `makefile` را که در قسمت قبل کارکرد آن را توضیح دادیم مینویسیم.

کد نوشته شده برای `:makefile`

CXX = g++

CXXFLAGS = -Wall -std=c++11

all: shm-writer shm-reader

shm-writer: shm-writer.cpp

`$(CXX) $(CXXFLAGS) shm-writer.cpp -o shm-writer -lrt`

shm-reader: shm-reader.cpp

```
$(CXX) $(CXXFLAGS) shm-reader.cpp -o shm-reader -lrt
```

clean:

```
rm -f shm-writer shm-reader
```

nano Makefile

توضیح کد :

CXX = g++

- این متغیر مشخص می‌کنے که از کامپایلر g++ کامپایلر زبان C++ استفاده کنیم.
- در جاهای دیگه از \$(CXX) استفاده می‌شے به جای نوشتن مستقیم g++.

CXXFLAGS = -Wall -std=c++11

- -Wall- همهی هشدارهای رایج کامپایلر رو فعال می‌کنے (برای کمک به پیدا کردن باگ).
- -std=c++11- تعیین می‌کنے که کد باید با استاندارد C++11 کامپایل بشه.

all: shm-writer shm-reader

هدف پیشفرض هست.

وقتی در ترمینال فقط می‌زنی make، این خط باعث می‌شے که هم shm-writer و هم shm-reader ساخته بشن.

shm-writer: shm-writer.cpp

```
$(CXX) $(CXXFLAGS) shm-writer.cpp -o shm-writer -lrt
```

- اگر فایل shm-writer.cpp تغییر کنے، این خط اجرا می‌شے.

- -lrt به معنی لینک کردن با کتابخانه **real-time** لینوکس است، که برای mmap و shm_open در حافظه اشتراکی نیاز است.

shm-reader: shm-reader.cpp

```
$(CXX) $(CXXFLAGS) shm-reader.cpp -o shm-reader -lrt
```

دقیقاً مشابه بالا ولی برای گیرنده.

فایل اجرایی shm-reader را از روی سورس shm-reader.cpp سازه.

clean:

```
rm -f shm-writer shm-reader
```

make clean

- این دستور، فایل‌های اجرایی shm-writer و shm-reader را حذف می‌کنه.

```

GNU nano 4.8
Makefile
Modified

CXX = g++
CXXFLAGS = -Wall -std=c++11

all: shm-writer shm-reader

shm-writer: shm-writer.cpp
    $(CXX) $(CXXFLAGS) shm-writer.cpp -o shm-writer -lrt

shm-reader: shm-reader.cpp
    $(CXX) $(CXXFLAGS) shm-reader.cpp -o shm-reader -lrt

clean:
    rm -f shm-writer shm-reader

```

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
 ^X Exit ^R Read File ^\ Replace ^U Paste Text ^T To Spell ^_ Go To Line

حال مانند قسمت قبل دو فایل C++ را میسازیم

Make

```
root@orangepizeroplus:~/myprojects/shm# make
g++ -Wall -std=c++11 shm-writer.cpp -o shm-writer -lrt
g++ -Wall -std=c++11 shm-reader.cpp -o shm-reader -lrt
root@orangepizeroplus:~/myprojects/shm# █
```

ترمینال 1 فرستنده:

```
root@orangepizeroplus:~/myprojects/shm# g++ shm-writer.cpp -o shm-writer -lrt
root@orangepizeroplus:~/myprojects/shm# make
make: Nothing to be done for 'all'.
root@orangepizeroplus:~/myprojects/shm# ./shm-writer
Enter text lines (type 'exit' to quit):
12345
ABCDE
XYZ
DZBDZBSDBHSDBSHB
█
```

ترمینال 2 گیرنده:

```
root@orangepizeroplus:~/myprojects/shm# ./shm-reader
SHM Content: "12345"
root@orangepizeroplus:~/myprojects/shm# ./shm-reader
SHM Content: "12345ABCDE"
root@orangepizeroplus:~/myprojects/shm# ./shm-reader
SHM Content: "XYZ45ABCDE"
root@orangepizeroplus:~/myprojects/shm# ./shm-reader
SHM Content: "DBHSDBSHBS"
root@orangepizeroplus:~/myprojects/shm# █
```

توضیح:

(shm-writer) نویسنده

• یک بافر دایره‌ای با اندازه ۱۰ کاراکتر در حافظه اشتراکی ایجاد کرده است.

• ورودی کاربر را خطبه خط دریافت می‌کند.

• به صورت چرخشی (دایره‌ای) در این بافر می‌نویسد.

خواننده (shm-reader)

• محتوای فعلی بافر را از حافظه اشتراکی می‌خواند و نمایش می‌دهد.

فرستنده :

Enter text lines (type 'exit' to quit):

12345

گیرنده :

SHM Content: "12345 "

فرستنده :

ABCDE

گیرنده :

SHM Content: "12345ABCDE"

فرستنده :

XYZ

گیرنده :

SHM Content: "XYZ45ABCDE"

فرستنده :

DZBDZBSDBHSDBSHB

گیرنده :

SHM Content: "DBHSDBSHBS"

چون در خط قبل DZBDZBSDBHSDBSHB وارد شد، فقط آخرین ۱۰ کاراکتر از این رشته در بافر ذخیره شده.

"DBHSDBSHBS" یعنی

- فرض کنیم حافظه فقط ۱۰ خانه دارد. وقتی پر شد، اگر داده جدید بیاد، از ابتدا شروع به جایگزینی می‌کنه.
- شبیه یک دایره که هر ورودی جدید، جای ورودی قدیمی تر را می‌گیره.
- برنامه درست عمل کرد.
- هر بار که بیش از ۱۰ کاراکتر وارد می‌کنید، داده‌ها به صورت چرخشی (دایره‌ای) جایگزین می‌شن.
- خواننده همیشه آخرین ۱۰ کاراکتر ذخیره شده در حافظه اشتراکی را نمایش می‌ده.

مثال بیشتر با در در نظر گرفتن فاصله (space) :

ترمینال ۱ فرستنده :

```
root@orangepizeroplus:~/myprojects/shm# g++ shm-writer.cpp -o shm-writer -lrt
root@orangepizeroplus:~/myprojects/shm# make
make: Nothing to be done for 'all'.
root@orangepizeroplus:~/myprojects/shm# ./shm-writer
Enter text lines (type 'exit' to quit):
12345
ABCDE
XYZ
DZBDZBSDBHSDBSHB
12342

123

123

123
ALI
QWER

123
ALI
QWERT
GHJGJHFJGCGFJGGJ

123

123

123

123
ASADI
█
```

ترمینال 2 گیرنده کل ترمینال :

```
root@orangepizeroplus: ~/myprojects/shm  
root@orangepizeroplus:~/myprojects/shm# ./shm-reader  
SHM Content: "2342DBSHB1"  
root@orangepizeroplus:~/myprojects/shm# ./shm-reader  
SHM Content: "2342DBSHB1"  
root@orangepizeroplus:~/myprojects/shm# ./shm-reader  
SHM Content: ""  
root@orangepizeroplus:~/myprojects/shm# ./shm-reader  
SHM Content: " 123 "  
root@orangepizeroplus:~/myprojects/shm# ./shm-reader  
SHM Content: " 123 "  
root@orangepizeroplus:~/myprojects/shm# ./shm-reader  
SHM Content: " 123 "  
root@orangepizeroplus:~/myprojects/shm# ./shm-reader  
SHM Content: "23 1"  
root@orangepizeroplus:~/myprojects/shm# ./shm-reader  
SHM Content: "23ALI 1"  
root@orangepizeroplus:~/myprojects/shm# ./shm-reader  
SHM Content: "23ALIWER1"  
root@orangepizeroplus:~/myprojects/shm#  
root@orangepizeroplus:~/myprojects/shm# ./shm-reader  
SHM Content: "23ALIWER1"  
root@orangepizeroplus:~/myprojects/shm# ./shm-reader  
SHM Content: ""  
root@orangepizeroplus:~/myprojects/shm# ./shm-reader  
SHM Content: "23 1"  
root@orangepizeroplus:~/myprojects/shm# ./shm-reader  
SHM Content: "23ALI 1"  
root@orangepizeroplus:~/myprojects/shm# ./shm-reader  
SHM Content: "23ALIQWERT"  
root@orangepizeroplus:~/myprojects/shm# ./shm-reader  
SHM Content: "GFJGGJFJGC"  
root@orangepizeroplus:~/myprojects/shm# ./shm-reader  
SHM Content: ""  
root@orangepizeroplus:~/myprojects/shm# ./shm-reader  
SHM Content: " 123 "  
root@orangepizeroplus:~/myprojects/shm#  
root@orangepizeroplus:~/myprojects/shm#  
root@orangepizeroplus:~/myprojects/shm#  
root@orangepizeroplus:~/myprojects/shm# ^C  
root@orangepizeroplus:~/myprojects/shm# ./shm-reader  
SHM Content: "123 "  
root@orangepizeroplus:~/myprojects/shm# ./shm-reader  
SHM Content: ""  
root@orangepizeroplus:~/myprojects/shm# ./shm-reader  
SHM Content: " 123 "  
root@orangepizeroplus:~/myprojects/shm# ./shm-reader  
SHM Content: ""  
root@orangepizeroplus:~/myprojects/shm# ./shm-reader  
SHM Content: " 123 "  
root@orangepizeroplus:~/myprojects/shm# ./shm-reader  
SHM Content: "ADI 123AS"  
root@orangepizeroplus:~/myprojects/shm#
```

پوشه myprojects میسازیم و به آن وارد میشویم

(اگه پوشه موجود نباشه ساخته میشه)(که از قبل موجوده)

```
mkdir -p /root/myprojects
```

```
cd /root/myprojects
```

```
Orange Pi 2.1.2 Focal ttyGS0
orangepizeroplus login:
orangepizeroplus login: root
Password:
Welcome to Orange Pi Focal with Linux 5.4.65-sunxi64
System load:  0.06 0.18 0.09   Up time:      4 min
Memory usage: 20 % of 477MB   IP:          192.168.100.99 192.168.100.98
CPU temp:    39°C
Usage of /:   5% of 29G

Last login: Wed May 28 20:17:22 UTC 2025 on ttyGS0
root@orangepizeroplus:~# mkdir -p /root/myprojects
root@orangepizeroplus:~# cd /root/myprojects
root@orangepizeroplus:~/myprojects#
```

کد simple-web-server.sh را مینویسیم:

```
nano simple-web-server.sh
```

```
GNU nano 4.8           simple-web-server.sh
#!/bin/bash

PORT=80

while true; do
    echo "Waiting for connection on port $PORT..."

    {
        read request_line
        echo "Received request: $request_line"
        response="HTTP/1.1 200 OK\r
Content-Type: text/plain\r
Connection: close\r
\r
Current date and time: $(date)\r
"
        echo -e "$response"
    } | nc -l -p $PORT -q 1
done
```

root@orangepizeroplus:~/myprojects#

اجازه دسترسی بهش میدیم:

chmod +x simple-web-server.sh

```
root@orangepizeroplus:~/myprojects# chmod +x simple-web-server.sh
root@orangepizeroplus:~/myprojects#
```

کد webserver.service هم مینویسیم :

nano webserver.service

```
GNU nano 4.8                               webserver.service
[Unit]
Description=Simple Web Server Service
After=network-online.target
Wants=network-online.target

[Service]
ExecStart=/root/myprojects/simple-web-server.sh
Restart=always
RestartSec=5
StandardOutput=journal
StandardError=journal

[Install]
WantedBy=multi-user.target

root@orangepireplus:~/myprojects#
```

```
cp webserver.service /etc/systemd/system/
```

این دستور فایل `webserver.service` را که ساختیم، کپی می کنیم به مسیر اصلی `systemd` در سیستم:
`/etc/systemd/system/` فقط فایل های `.service` رو از این مسیر یا مسیرهای مشابه می خونه.

```
systemctl daemon-reload
```

این دستور به `systemd` می گه:
"**فایل های سرویس جدید رو دوباره بارگذاری کن.**"
چون ما الان یه سرویس جدید (`webserver.service`) ساختیم، باید به `systemd` بگیم تغییرات جدید رو بشناسه.

```
systemctl enable webserver.service
```

این دستور باعث می شه که سرویس در زمان بوت سیستم به صورت خودکار اجرا بشه.
•
• یه جور ثبت دائم سرویسه.

```
systemctl start webserver.service
```

• سرویس Webserver.service را همین حالا اجرا می‌کنه.

• اگه اسکریپت داخل فایل درست نوشته شده باشه، حالا وبسرور بالا میاد

```
systemctl status webserver.service
```

وضعیت فعلی سرویس رو نشون می‌ده:

آیا فعال هست؟

با موفقیت اجرا شده؟

اگر خطای رخ داده، چی بوده؟

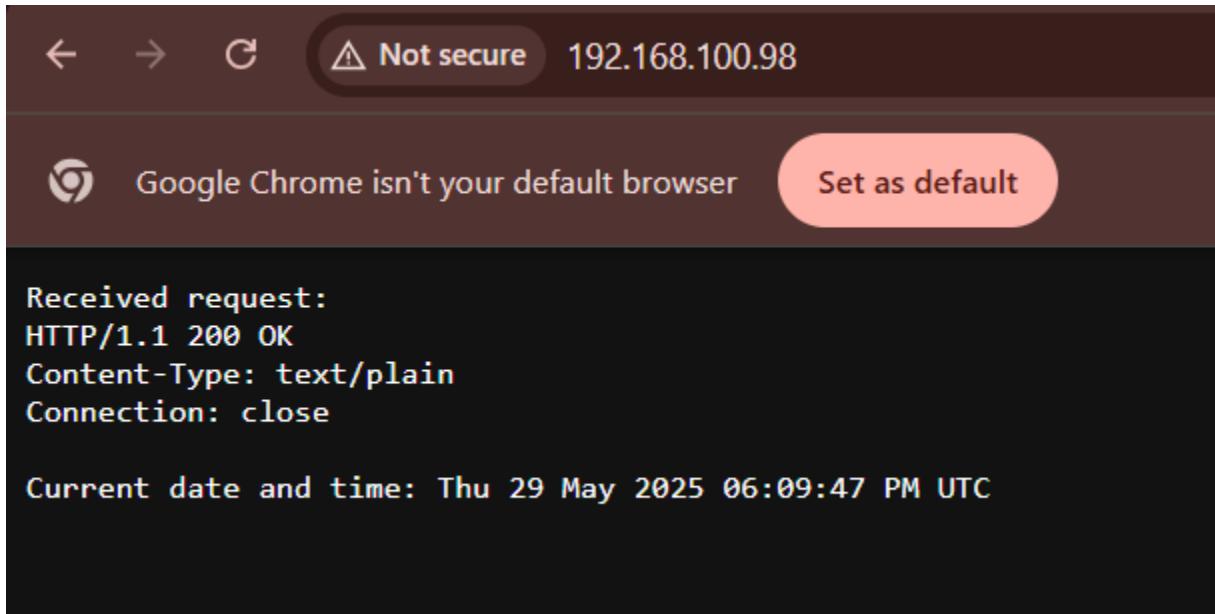
PID و زمان اجرا رو هم نشون می‌ده.

```
root@orangepizeroplus:~/myprojects# cp webserver.service /etc/systemd/system/
root@orangepizeroplus:~/myprojects# systemctl daemon-reload
root@orangepizeroplus:~/myprojects# systemctl enable webserver.service
Created symlink /etc/systemd/system/multi-user.target.wants/webserver.service →
/etc/systemd/system/webserver.service.
root@orangepizeroplus:~/myprojects# systemctl start webserver.service
root@orangepizeroplus:~/myprojects# systemctl status webserver.service
● webserver.service - Simple Web Server Service
   Loaded: loaded (/etc/systemd/system/webserver.service; enabled; vendor pre>
   Active: active (running) since Thu 2025-05-29 18:03:43 UTC; 4s ago
     Main PID: 2027 (simple-web-serv)
        Tasks: 2 (limit: 403)
      Memory: 1.0M
        CGroup: /system.slice/webserver.service
                  └─2027 /bin/bash /root/myprojects/simple-web-server.sh
                      ├─2029 nc -l -p 80 -q 1

May 29 18:03:43 orangepizeroplus systemd[1]: Started Simple Web Server Service.
May 29 18:03:43 orangepizeroplus simple-web-server.sh[2027]: Waiting for connec>
lines 1-12/12 (END)
```

حال که وب سرور اجرا شد با ip خود برد با لپ تاپ بهش متصل میشیم

192.168.100.98



حال با هربار رفرش کردن صفحه‌ی مرورگر نتیجه رو با زدن کد زیر در ترمینال برد میتوانیم ببینیم:

```
journalctl -u webserver.service -f
```

• خروجی لاغ مربوط به سرویس webserver.service را نمایش بده (-u)

• و همزمان با اجرای زنده سرویس، لاغ‌ها را دنبال کن (-f = follow)

خروجی ترمینال برد :

```

root@orangeplus:~/myprojects# journalctl -u webserver.service -f
-- Logs begin at Thu 2025-05-29 17:56:03 UTC. --
May 29 18:08:30 orangeplus simple-web-server.sh[2027]: Waiting for connection on port 80...
May 29 18:09:21 orangeplus simple-web-server.sh[2353]: GET / HTTP/1.1
May 29 18:09:21 orangeplus simple-web-server.sh[2353]: Host: 192.168.100.98
May 29 18:09:21 orangeplus simple-web-server.sh[2353]: Connection: keep-alive
May 29 18:09:21 orangeplus simple-web-server.sh[2353]: Upgrade-Insecure-Requests: 1
May 29 18:09:21 orangeplus simple-web-server.sh[2353]: User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/136.0.0.0 Safari/537.36
May 29 18:09:21 orangeplus simple-web-server.sh[2353]: Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8,application/signed-exch
ange;v=b3;q=0.7
May 29 18:09:21 orangeplus simple-web-server.sh[2353]: Accept-Encoding: gzip, deflate
May 29 18:09:21 orangeplus simple-web-server.sh[2353]: Accept-Language: en-US,en;q=0.9
May 29 18:09:21 orangeplus simple-web-server.sh[2353]:
May 29 18:09:22 orangeplus simple-web-server.sh[2027]: Waiting for connection on port 80...
May 29 18:09:44 orangeplus simple-web-server.sh[2411]: GET / HTTP/1.1
May 29 18:09:44 orangeplus simple-web-server.sh[2411]: Host: 192.168.100.9
8
May 29 18:09:44 orangeplus simple-web-server.sh[2411]: Connection: keep-alive
May 29 18:09:44 orangeplus simple-web-server.sh[2411]: Cache-Control: max-
age=0
May 29 18:09:44 orangeplus simple-web-server.sh[2411]: Upgrade-Insecure-Re-
quests: 1
May 29 18:09:44 orangeplus simple-web-server.sh[2411]: User-Agent: Mozilla
/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome
/136.0.0.0 Safari/537.36
May 29 18:09:44 orangeplus simple-web-server.sh[2411]: Accept: text/html,a-
pplication/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*
;q=0.8,application/signed-exchange;v=b3;q=0.7
May 29 18:09:44 orangeplus simple-web-server.sh[2411]: Accept-Encoding: gz-
ip, deflate
May 29 18:09:44 orangeplus simple-web-server.sh[2411]: Accept-Language: en-
-US,en;q=0.9
May 29 18:09:44 orangeplus simple-web-server.sh[2411]:
May 29 18:09:46 orangeplus simple-web-server.sh[2027]: Waiting for connect-
ion on port 80...
May 29 18:09:46 orangeplus simple-web-server.sh[2425]: GET /favicon.ico HT

```

Waiting for connection on port 80...

این پیام یعنی **netcat** (یا اسکریپت ما) آمده است تا هر درخواست **HTTP** که به پورت 80 بیاد رو دریافت کنه.

مرورگر به سرور وصل شده:

GET / HTTP/1.1

Host: 192.168.100.98

User-Agent: Mozilla/5.0 (Windows NT 10.0; ...)

این قسمت زمانی اتفاق افتاده که ما در مرورگر لپتاپ زدیم 192.169.100.98

و مرورگر درخواست **GET** فرستاده تا صفحه وب رو دریافت کنه.

Accept: text/html,application/xhtml+xml,...

Connection: keep-alive

Upgrade-Insecure-Requests: 1

Accept-Encoding: gzip, deflate

این‌ها هدرهای **HTTP** هستند که مرورگر برای اطلاعات بیشتر درباره‌ی خودش و قابلیت‌هایی مثل فشرده‌سازی و زبان می‌فرسته.

هربار صفحه مرورگر رو رفرش میکنیم :

```
May 30 11:51:26 orangepizeroplus simple-web-server.sh[1121]: Waiting for connection on port 80...
May 30 11:52:13 orangepizeroplus simple-web-server.sh[1614]: GET / HTTP/1.1
May 30 11:52:13 orangepizeroplus simple-web-server.sh[1614]: Host: 192.168.100.99
May 30 11:52:13 orangepizeroplus simple-web-server.sh[1614]: Connection: keep-alive
May 30 11:52:13 orangepizeroplus simple-web-server.sh[1614]: Cache-Control: max-age=0
May 30 11:52:13 orangepizeroplus simple-web-server.sh[1614]: Upgrade-Insecure-Requests: 1
May 30 11:52:13 orangepizeroplus simple-web-server.sh[1614]: User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/136.0.0.0 Safari/537.36
May 30 11:52:13 orangepizeroplus simple-web-server.sh[1614]: Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
May 30 11:52:13 orangepizeroplus simple-web-server.sh[1614]: Accept-Encoding: gzip, deflate
May 30 11:52:13 orangepizeroplus simple-web-server.sh[1614]: Accept-Language: en-US,en;q=0.9
May 30 11:52:13 orangepizeroplus simple-web-server.sh[1614]:
May 30 11:52:14 orangepizeroplus simple-web-server.sh[1121]: Waiting for connection on port 80...
```

May 30 11:51:26 orangepizeroplus simple-web-server.sh[1121]: Waiting for connection on port 80...

May 30 11:52:13 orangepizeroplus simple-web-server.sh[1614]: GET / HTTP/1.1

May 30 11:52:13 orangepizeroplus simple-web-server.sh[1614]: Host: 192.168.100.99

May 30 11:52:13 orangepizeroplus simple-web-server.sh[1614]: Connection: keep-alive

May 30 11:52:13 orangepizeroplus simple-web-server.sh[1614]: Cache-Control: max-age=0

May 30 11:52:13 orangepizeroplus simple-web-server.sh[1614]: Upgrade-Insecure-Requests: 1

May 30 11:52:13 orangepizeroplus simple-web-server.sh[1614]: User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/136.0.0.0 Safari/537.36

May 30 11:52:13 orangepizeroplus simple-web-server.sh[1614]: Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7

May 30 11:52:13 orangepizeroplus simple-web-server.sh[1614]: Accept-Encoding: gzip, deflate

May 30 11:52:13 orangepizeroplus simple-web-server.sh[1614]: Accept-Language: en-US,en;q=0.9

May 30 11:52:13 orangepizeroplus simple-web-server.sh[1614]:

May 30 11:52:14 orangepizeroplus simple-web-server.sh[1121]: Waiting for connection on port 80...

اسکریپت simple-web-server.sh داره با nc -l -p 80 (یا مشابهش) منتظر اتصال جدید میمونه .

این یعنی: آماده‌ام که کلاینت جدیدی به پورت 80 وصل شه.

- مسیر درخواست شده / صفحه اصلی
- از کجا 98.168.100.192 :
- مرورگر Chrome روی ویندوز 10
- می‌توانه فایل‌های XML، HTML، تصویر، و غیره دریافت کنه.

حال با زدن کد زیر در ترمینال:

```
./simple-web-server.sh
```

```
root@orangepizeroplus:~/myprojects# ./simple-web-server.sh
Waiting for connection on port 80...
```

و با هربار رفرش کردن صفحه مرورگر:

```
root@orangepizeroplus:~/myprojects# ./simple-web-server.sh
Waiting for connection on port 80...
GET / HTTP/1.1
Host: 192.168.100.98
Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/136.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9

./simple-web-server.sh
Waiting for connection on port 80...
GET / HTTP/1.1
Host: 192.168.100.98
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/136.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
```

```
root@orangepizeroplus:~/myprojects# ./simple-web-server.sh
```

Waiting for connection on port 80...

GET /favicon.ico HTTP/1.1

Host: 192.168.100.98

Connection: keep-alive

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/136.0.0.0 Safari/537.36

Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*/*;q=0.8

Referer: http://192.168.100.98/

Accept-Encoding: gzip, deflate

Accept-Language: en-US,en;q=0.9

در اینجا، وقتی اسکریپت simple-web-server.sh روی پورت 80 منتظر اتصال بود، مرورگر ما مثلاً Chrome ویندوز به IP 192.168.100.98 برد Orange Pi یعنی درخواست بالا را فرستاده

GET /favicon.ico HTTP/1.1

مرورگر می خواهد فایل کوچکی به نام favicon.ico را از مسیر ریشه‌ی سایت دریافت کنه. این فایل آیکون کوچکیه که معمولاً کنار عنوان تب مرورگر نمایش داده می‌شه.

Host: 192.168.100.98

این آدرس IP سروریه که مرورگر به اون درخواست فرستاده برد ما

Connection: keep-alive

یعنی مرورگر ترجیح می‌ده اتصال TCP بعد از این درخواست باز بمنه، برای اینکه اگه درخواست دیگه‌ای باشه، سریع‌تر انجام شه در HTTP/1.1 عادیه

User-Agent: Mozilla/5.0 ...

اطلاعات مرورگر کلاینت هست در اینجا 136 Chrome

Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8

مرورگر اعلام می‌کنه که چه نوع فایل‌هایی رو می‌تونه قبول کنه. در اینجا چون دنبال favicon.ico هست، لیستی از انواع فرمتهای تصویری آورده.

Referer: <http://192.168.100.98/>

مرورگر می‌گه این درخواست از چه صفحه‌ای ارسال شده. یعنی وقتی ما آدرس http://192.168.100.98/ رو باز کردیم، مرورگر سعی کرده favicon.ico /هم بخونه.

Accept-Encoding: gzip, deflate

Accept-Language: en-US,en;q=0.9

مرورگر اعلام می‌کنه می‌تونه پاسخ رو فشرده‌شده (gzip) دریافت کنه و زبان ترجیحی اش انگلیسیه.

مرورگر به طور خودکار هنگام باز کردن یک سایت، دنبال فایل favicon.ico می‌گردد تا در تپ مرورگر نشون بده. اگر سرور شما جوابی نده یا این فایل وجود نداشته باشد، مشکلی پیش نمی‌یاد، فقط در ترمینال لاجش چاپ می‌شود. این رفتار طبیعی و استاندارد مرورگرهای است.

فایل webserver.service که در برد ساختیم کپی می‌کنیم روی ویندوز لپ تاپمون:

(در cmd ویندوز)

```
scp root@192.168.100.98:/root/myprojects/webserver.service C:\Users\Ali\Desktop\
```

```
C:\Windows\System32>scp root@192.168.100.98:/root/myprojects/webserver.service C:\Users\Ali\Desktop\  
root@192.168.100.98's password:  
webserver.service  
100% 271 15.6KB/s 00:00  
C:\Windows\System32>
```

کد ها و توضیح آن ها:

auto-connect.sh:

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

```
LOG_FILE="/root/auto-connect.log"
```

```
SSID="Ali"
```

```
PASSWORD="ali1379!#(aSadI"
```

```
echo "-----" >> $LOG_FILE
```

```
echo "---- $(date) ----" >> $LOG_FILE
```

در حال اجرا هست یا نه NetworkManager بررسی اینکه

```
if ! pgrep -x "NetworkManager" > /dev/null; then
```

```
    echo "Error: NetworkManager is not running." >> $LOG_FILE
```

```
    echo "-----" >> $LOG_FILE
```

```
    exit 1
```

```
fi
```

فعالسازی کارت وای‌فای و آماده‌سازی برای اسکن

```
nmcli radio wifi on
```

```
ip link set wlan0 up
```

```
sleep 2
```

اسکن شبکه‌های وای‌فای اطراف

```
echo "Scanning available Wi-Fi networks..." >> $LOG_FILE
```

```
nmcli device wifi rescan  
sleep 2  
nmcli device wifi list >> $LOG_FILE
```

بررسی اتصال فعال

```
CONNECTED_SSID=$(nmcli -t -f active,ssid dev wifi | grep '^yes' | cut -d':' -f2)
```

```
if [ "$CONNECTED_SSID" = "$SSID" ]; then  
    echo "Already connected to $SSID" >> $LOG_FILE  
else  
    echo "Not connected to $SSID. Trying to connect..." >> $LOG_FILE  
    nmcli device wifi connect "$SSID" password "$PASSWORD" >> $LOG_FILE 2>&1
```

بررسی نتیجه اتصال

```
NEW_CONNECTED_SSID=$(nmcli -t -f active,ssid dev wifi | grep '^yes' | cut -d':' -f2)  
if [ "$NEW_CONNECTED_SSID" = "$SSID" ]; then  
    echo "Successfully connected to $SSID" >> $LOG_FILE  
else  
    echo "Failed to connect to $SSID" >> $LOG_FILE  
fi
```

```
echo "-----" >> $LOG_FILE
```

توضیح :

این اسکریپت هر بار اجرا می شه:

- بررسی می کنے که سیستم به یک شبکه وای فای خاص مثلاً **Ali** متصل هست یا نه.
- اگه نبود، تلاش می کنے وصل بشه.
- لاغ تمام اقدامات و خطاهای را در فایل **root/auto-connect.log** ثبت می کنے.
- برای اجرا شدن خودکار در بوت یا به صورت زمان بندی شده مثلاً با **cronjob** مناسبه.

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

خط اول: مشخص می کنے که این فایل یک اسکریپت بش هست و باید با **bash** اجرا بشه.

```
LOG_FILE="/root/auto-connect.log"
SSID="Ali"
PASSWORD="ali1379!#(aSadl"
```

◆ سه متغیر تعریف شده:

LOG_FILE: مسیر فایل لاغ برای ذخیره و وضعیت ها

SSID: نام وای فای هدف

PASSWORD: رمز وای فای

```
echo "-----" >> $LOG_FILE
echo "---- $(date) ----" >> $LOG_FILE
```

◆ ثبت تاریخ و جدادگری در فایل لاغ برای هر اجرای جدید اسکریپت

```
if ! pgrep -x "NetworkManager" > /dev/null; then
    echo "Error: NetworkManager is not running." >> $LOG_FILE
    echo "-----" >> $LOG_FILE
    exit 1
fi
```

◆ بررسی می کنے آیا سرویس **NetworkManager** فعال است یا نه.

اگر فعال نبود:

پیام خطا در لاغ ثبت می شه

اجرای اسکریپت متوقف می شه

```
nmcli radio wifi on
```

```
ip link set wlan0 up
```

```
sleep 2
```

◆ اطمینان از اینکه کارت وای فای (wlan0) روشن و فعال باشه.

◆ sleep 2 برای دادن فرصت به سیستم جهت فعال سازی کامل.

```
echo "Scanning available Wi-Fi networks..." >> $LOG_FILE
```

```
nmcli device wifi rescan
```

```
sleep 2
```

```
nmcli device wifi list >> $LOG_FILE
```

◆ اسکن شبکه های اطراف و ثبت لیست کامل آنها در لاغ.

(خیلی مفیده برای بررسی اینکه SSID موردنظر در دسترس هست یا نه)

```
CONNECTED_SSID=$(nmcli -t -f active,ssid dev wifi | grep '^yes' | cut -d':' -f2)
```

◆ این خط بررسی می کنه که آیا الان به وای فای متصل هستیم یا نه.

اگر متصل باشیم، نام SSID متصل شده را استخراج می کنه.

```
if [ "$CONNECTED_SSID" = "$SSID" ]; then
```

```
echo "Already connected to $SSID" >> $LOG_FILE
```

◆ اگر به همون وای فای موردنظر (Ali) وصل بودیم → ثبت در لاغ و تمام.

```
else
```

```
echo "Not connected to $SSID. Trying to connect..." >> $LOG_FILE
```

```
nmcli device wifi connect "$SSID" password "$PASSWORD" >> $LOG_FILE 2>&1
```

◆ در غیر این صورت:

تلash برای اتصال به وای فای

لگ تمام خروجی و خطاهای اتصال

```
NEW_CONNECTED_SSID=$(nmcli -t -f active,ssid dev wifi | grep '^yes' | cut -d':' -f2)
```

```
if [ "$NEW_CONNECTED_SSID" = "$SSID" ]; then  
    echo "Successfully connected to $SSID" >> $LOG_FILE
```

```
else
```

```
    echo "Failed to connect to $SSID" >> $LOG_FILE
```

```
fi
```

```
fi
```

◆ بعد از تلاش اتصال:

دوباره بررسی می کنند آیا اتصال موفق بوده یا نه

ثبت نتیجه در لگ

```
echo "-----" >> $LOG_FILE
```

◆ پایان بندی لگ برای مرتب بودن لگ فایل

fifo-reader.sh

```
#!/bin/bash

# مسیر FIFO
FIFO_PATH="/tmp/myfifo"

# اگر FIFO نداشت، بساز
if [[ ! -p $FIFO_PATH ]]; then
    mkfifo $FIFO_PATH
fi

echo "Reader started. Waiting for messages..."
```

```
# خواندن از FIFO ترمینال روی نمایش و
while true; do
    if read line < $FIFO_PATH; then
        echo "Received: $line"
    fi
done
```

توضیح :

ساخت یا استفاده از یک فایل FIFO در مسیر /tmp/myfifo و سپس خواندن پیغام‌هایی که توسط یک فرستنده (**writer**) به این FIFO ارسال می‌شود.

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

مشخص می کنے کہ اسکریپٹ باید با Bash اجرا بشے.

```
# مسیر FIFO
```

```
FIFO_PATH="/tmp/myfifo"
```

مسیر فایل FIFO رو در یک متغیر ذخیره کرده تا در ادامه استفاده کنے.

```
# وجود نداشت، بساز اگر FIFO
```

```
if [[ ! -p $FIFO_PATH ]]; then  
    mkfifo $FIFO_PATH  
fi
```

بررسی می کنے کہ آیا فایل FIFO قبلاً ایجاد شده یا نه با -p.
اگر وجود نداشت، با دستور mkfifo ایجادش می کنے.

```
echo "Reader started. Waiting for messages..."
```

پیام ساده‌ای چاپ می کنے تا به کاربر نشون بده که برنامه در حال اجراست و منتظر دریافت پیام از FIFO هست.

```
# و نمایش روی ترمینال FIFO خواندن از
```

```
while true; do  
    if read line < $FIFO_PATH; then  
        echo "Received: $line"  
    fi
```

```
done
```

یک حلقه بی‌نهایت:

- از فایل FIFO به صورت بلاکینگ (blocking) می خونه

- وقتی پیامی دریافت شد، در متغیر line اذخیره می کنے

- سپس پیام دریافتی را در ترمینال چاپ می کنے با:

- > Received: <

کاربرد عملی:

1. ترمینال اول: اجرای این اسکریپت (گیرنده)

```
./fifo-reader.sh
```

2. ترمینال دوم: نوشتن پیام در FIFO فرستنده

```
echo "Hello FIFO!" > /tmp/myfifo
```

هر پیام جدید که نوشته بشه، در ترمینال اول نمایش داده می‌شه.

نکته:

برای نوشتن به FIFO حتماً باید خواننده‌ای در حال اجرا باشد، چون FIFO‌تا وقتی کسی منتظر نباشد، write را کامل نمی‌کند (یعنی بلاک می‌موند).

```
fifo-writer.sh:
```

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

```
# مسیر FIFO
```

```
FIFO_PATH="/tmp/myfifo"
```

```
# وجود نداشت، بساز اگر FIFO
```

```
if [[ ! -p $FIFO_PATH ]]; then
```

```
    mkfifo $FIFO_PATH
```

```
fi
```

```
echo "Writer started. Type messages below:"
```

```
# خواندن از ترمینال و نوشتن در FIFO
```

```
while true; do
```

```
    read input
```

```
    echo "$input" > $FIFO_PATH
```

```
done
```

: توضیح

ایجاد یک فایل FIFO (در صورت نبود آن) و سپس خواندن ورودی کاربر از ترمینال و ارسال آن به FIFO برای اینکه توسط یک فرآیند گیرنده (reader) خوانده شود.

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

مشخص می‌کند که اسکریپت با Bash اجرا شود.

```
# مسیر FIFO
FIFO_PATH="/tmp/myfifo"
```

مسیر فایل FIFO در متغیر FIFO_PATH ذخیره شده است. این همان مسیری است که فرآیند reader نیز از آن استفاده می‌کند.

```
# اگر FIFO وجود نداشت، بساز
if [[ ! -p $FIFO_PATH ]]; then
    mkfifo $FIFO_PATH
fi
```

بررسی می‌کند که آیا FIFO وجود دارد یا نه -p برای check کردن named pipe. اگر وجود نداشته باشد، با دستور mkfifo آن را می‌سازد.

```
echo "Writer started. Type messages below:"
```

یک پیام چاپ می‌شود تا کاربر بداند برنامه آماده‌ی دریافت ورودی است.

```
# خواندن از ترمینال و نوشتن در FIFO
while true; do
    read input
    echo "$input" > $FIFO_PATH
done
```

حلقه بی‌نهایت:

- از ترمینال یک خط ورودی می‌گیرد (read input)
- آن را در FIFO می‌نویسد (echo "\$input" > \$FIFO_PATH)

این فرآیند ادامه دارد تا زمانی که اسکریپت متوقف شود.

اگر reader فعال نباشد و شما پیام بنویسید، echo روی FIFO متوقف می‌ماند تا یک گیرنده پیدا شود.

فایل FIFO مانند فایل عادی نیست؛ داده‌ها پس از خوانده شدن از بین می‌روند. (one-time read).

signal-receiver:

```
#include <iostream>
#include <fstream>
#include <signal.h>
#include <unistd.h>
#include <cstring>

std::ofstream logfile("received-signals.log", std::ios::app);

void signal_handler(int signal) {
    std::string name = strsignal(signal);
    std::cout << "Received signal: " << signal << " (" << name << ")" << std::endl;
    if (logfile.is_open()) {
        logfile << "Received signal: " << signal << " (" << name << ")" << std::endl;
    }
}

int main() {
    std::cout << "PID: " << getpid() << std::endl;
    for (int i = 1; i < 32; ++i) {
        signal(i, signal_handler);
    }
}
```

```

    }

while (true) {
    pause(); // wait for signal
}

return 0;
}

```

توضیح :

```

#include <iostream>

#include <fstream>

#include <signal.h>

#include <unistd.h>

#include <cstring>

```

این‌ها کتابخانه‌های مورد نیاز هستند:

برای چاپ به ترمینال **iostream**

برای نوشتن در فایل **fstream**

برای مدیریت سیگنال‌ها **signal.h**

برای **pause()** و **getpid()** **unistd.h**

برای تبدیل عدد سیگنال به نام آن با **strsignal()** **cstring**

```
std::ofstream logfile("received-signals.log", std::ios::app);
```

یک فایل به نام **received-signals.log** باز می‌کند و به انتهای آن (**append**) می‌نویسد. این فایل برای ذخیره لاغ سیگنال‌های دریافتی استفاده می‌شود.

```
void signal_handler(int signal){
```

```
    std::string name = strsignal(signal);
```

```

std::cout << "Received signal: " << signal << "(" << name << ")" << std::endl;
if (logfile.is_open())
{
    logfile << "Received signal: " << signal << "(" << name << ")" << std::endl;
}

```

وقتی سیگنالی دریافت شود:

نام آن را (مثل "Interrupt") پیدا می‌کند.

پیام در ترمینال و فایل لگ چاپ می‌شود.

خط 16: نمایش شناسه پردازش (PID)

```
std::cout << "PID: " << getpid() << std::endl;
```

شناسه این پردازش را می‌گیرد و نمایش می‌دهد. برای ارسال سیگنال لازم است این PID را داشته باشد.

◆ **خط 17–18:** ثبت تمام سیگنال‌ها

```

for (int i = 1; i < 32; ++i)
{
    signal(i, signal_handler);
}

```

برای سیگنال‌های شماره 1 تا 31، هندر سفارشی تعریف می‌کند. این یعنی اگر این سیگنال‌ها دریافت شوند، به جای توقف یا رفتار پیش‌فرض، تابع signal_handler اجرا می‌شود.

ولی دقت کنم: سیگنال‌های SIGKILL (9) و SIGSTOP (19) قابل کنترل (catch) نیستند و توسط signal مدیریت نمی‌شوند.

◆ **خط 19–22:** انتظار برای سیگنال

```
while (true)
```

```
pause(); // wait for signal  
{
```

پردازش با pause() متوقف می‌ماند تا وقتی که سیگنالی دریافت شود.

signal-sender:

```
#include <iostream>  
  
#include <csignal>  
  
#include <cstdlib>  
  
#include <cerrno>  
  
#include <cstring>  
  
  
int main(int argc, char* argv[]) {  
    if (argc != 3) {  
        std::cerr << "Usage: " << argv[0] << " <PID> <SIGNAL_NUMBER>" << std::endl;  
        return 1;  
    }  
  
  
    pid_t pid = std::stoi(argv[1]);  
    int sig = std::stoi(argv[2]);  
  
  
    if (kill(pid, sig) == -1) {  
        std::cerr << "Failed to send signal: " << strerror(errno) << std::endl;  
        return 1;
```

```

    }

std::cout << "Signal " << sig << " sent to PID " << pid << std::endl;

return 0;
}

```

توضیح :

فرستادن یک سیگنال مثلاً $\text{SIGINT} = 2$ به یک پردازش مشخص (با PID معلوم) از طریق دستور `kill()` در زبان C++

1 کتابخانه‌ها:

```

#include <iostream> // cout و cerr برای استفاده از
#include <csignal> // kill برای تابع
#include <cstdlib> // exit, stoi
#include <cerrno> // errno برای
#include <cstring> // strerror برای

```

2 بررسی تعداد آرگومان‌ها:

```

if (argc != 3) {

    std::cerr << "Usage: " << argv[0] << " <PID> <SIGNAL_NUMBER>" << std::endl;
    return 1;
}

```

بررسی می‌کند که آیا دقیقاً دو آرگومان PID و شماره سیگنال وارد شده‌اند یا نه. اگر نه، پیغام راهنمایی چاپ می‌شود.

3 تبدیل ورودی‌ها به عدد:

```

pid_t pid = std::stoi(argv[1]);
int sig = std::stoi(argv[2]);

```

از std::stoi برای تبدیل [1] و [2] (که به صورت رشته هستند) به عدد صحیح استفاده می کند:

- pid شناسه پردازش
- sig شماره سیگنال مثلاً 2 برای SIGINT

4 ارسال سیگنال:

```
if (kill(pid, sig) == -1) {  
    std::cerr << "Failed to send signal: " << strerror(errno) << std::endl;  
    return 1;  
}
```

با استفاده از kill()، سیگنال sig به pid داده شده ارسال می شود. اگر ارسال موفق نباشد، خطای چاپ می شود مثلاً اگر PID وجود نداشته باشد یا اجازه نداشته باشیم.

5 تأیید موفقیت:

```
std::cout << "Signal " << sig << " sent to PID " << pid << std::endl;
```

اگر سیگنال با موفقیت ارسال شود، تأیید در ترمینال چاپ می شود.

shm-writer.

```
#include <iostream>  
  
#include <fcntl.h>  
  
#include <sys/mman.h>  
  
#include <unistd.h>  
  
#include <cstring>  
  
#include <cerrno>  
  
#include <string>
```

```

#define SHM_SIZE 10

const char* SHM_NAME = "/my_shm";


int main() {

    int shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666);

    if (shm_fd == -1) {

        std::cerr << "Failed to create shared memory: " << strerror(errno) << std::endl;

        return 1;
    }

    if (ftruncate(shm_fd, SHM_SIZE + 1) == -1) { // +1 for write_pos

        std::cerr << "Failed to resize shared memory: " << strerror(errno) << std::endl;

        return 1;
    }

    void* ptr = mmap(0, SHM_SIZE + 1, PROT_READ | PROT_WRITE, MAP_SHARED,
    shm_fd, 0);

    if (ptr == MAP_FAILED) {

        std::cerr << "Failed to map shared memory: " << strerror(errno) << std::endl;

        return 1;
    }

    char* mem = static_cast<char*>(ptr);

    char& write_pos = mem[SHM_SIZE]; // آخرین بایت برای نگهداری موقعیت نوشتن
}

```

```

// مقدار اولیه موقعیت نوشتن رو اگر نامشخصه صفر کن
if (write_pos < 0 || write_pos >= SHM_SIZE) {
    write_pos = 0;
}

std::string input;
std::cout << "Enter text lines (type 'exit' to quit):" << std::endl;

while (true) {
    std::getline(std::cin, input);
    if (input == "exit") {
        break;
    }

    for (char c : input) {
        mem[write_pos] = c;
        write_pos = (write_pos + 1) % SHM_SIZE;
    }
}

munmap(ptr, SHM_SIZE + 1);
close(shm_fd);

return 0;
}

```

توضیح :

نوشتن داده‌ی متنی از کاربر در یک حافظه اشتراکی (**Shared Memory**) به صورت دایره‌ای (**Circular Buffer**) به اندازه ۱۰ بایت.

اجزای اصلی برنامه:

```
#define SHM_SIZE 10  
const char* SHM_NAME = "/my_shm";
```

اندازه‌ی بافر اشتراکی (۱۰ بایت). SHM_SIZE:

نام حافظه اشتراکی با پیشوند / طبق استاندارد POSIX SHM_NAME:

گام ۱: ایجاد و آماده‌سازی حافظه اشتراکی

```
int shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666);
```

حافظه اشتراکی را با دسترسی خواندن و نوشتن ایجاد می‌کند. shm_open:

0666: سطح دسترسی برای فایل.

```
ftruncate(shm_fd, SHM_SIZE + 1);
```

حافظه را به اندازه ۱۰ بایت + ۱ بایت اضافی گسترش می‌دهد.

آن بایت اضافی برای ذخیره موقعیت نوشتن (write_pos) استفاده می‌شود.

```
void* ptr = mmap(...);
```

حافظه اشتراکی را به فضای آدرس برنامه نگاشت می‌کند.

اگر موفق نشود، پیام خطای چاپ می‌شود.

اساره‌گرها:

```
char* mem = static_cast<char*>(ptr);  
char& write_pos = mem[SHM_SIZE];
```

- اشارهگر به بافر داده mem.

- به آخرین بایت (بایت شماره ۱۰) اشاره دارد که برای نگهداری موقعیت نوشتن استفاده می‌شود.

```
if (write_pos < 0 || write_pos >= SHM_SIZE) {
    write_pos = 0;
}
```

- اگر مقدار موقعیت نوشتن معتبر نیست، مقداردهی اولیه به صفر انجام می‌شود.

حلقه ورود داده:

```
while (true) {
    std::getline(std::cin, input);
    if (input == "exit") break;

    for (char c : input) {
        mem[write_pos] = c;
        write_pos = (write_pos + 1) % SHM_SIZE;
    }
}
```

- تا زمانی که کاربر "exit" وارد نکند، از ورودی می‌خواند.

- هر کarakتر به بافر نوشته می‌شود و موقعیت نوشتن با چرخش (Modulo) به روزرسانی می‌شود.

این رفتار، بافر را دایره‌ای (**circular**) می‌کند: وقتی به انتهای بافر می‌رسد، از اول شروع می‌کند.

پاکسازی:

```
munmap(ptr, SHM_SIZE + 1);
close(shm_fd);
```

- حافظه را از فضای برنامه آزاد می‌کند.
- فایل حافظه اشتراکی را می‌بندد.

خروجی و کاربرد

این برنامه به طور پیوسته پیام‌های متنی را در بافر اشتراکی می‌نویسد. برنامه‌ی گیرنده (shm-reader) می‌تواند این حافظه را بخواند و محتوا را نمایش دهد.

shm-reader:

```
#include <iostream>
#include <fcntl.h>
#include <sys/mman.h>
#include <unistd.h>
#include <cerrno>
#include <cstring>

#define SHM_SIZE 10
const char* SHM_NAME = "/my_shm";

int main() {
    int shm_fd = shm_open(SHM_NAME, O_RDONLY, 0666);
    if (shm_fd == -1) {
        std::cerr << "Failed to open shared memory: " << strerror(errno) << std::endl;
        return 1;
}
```

```

void* ptr = mmap(0, SHM_SIZE, PROT_READ, MAP_SHARED, shm_fd, 0);

if (ptr == MAP_FAILED) {

    std::cerr << "Failed to map shared memory: " << strerror(errno) << std::endl;

    return 1;
}

char buffer[SHM_SIZE + 1] = {0};

memcpy(buffer, ptr, SHM_SIZE);

std::cout << "SHM Content: \\" << buffer << "\"" << std::endl;

munmap(ptr, SHM_SIZE);

close(shm_fd);

return 0;
}

```

توضیح :

```

#define SHM_SIZE 10

const char* SHM_NAME = "/my_shm";

```

- تعداد بایت‌هایی که قرار است از حافظه خوانده شود (۱۰ بایت).
- نام حافظه‌ی اشتراکی که باید با برنامه‌ی writer یکسان باشد.

گام ۱: باز کردن حافظه اشتراکی

```
int shm_fd = shm_open(SHM_NAME, O_RDONLY, 0666);
```

- فایل حافظه اشتراکی با دسترسی فقط خواندنی (O_RDONLY) باز می‌شود.
- اگر حافظه وجود نداشته باشد مثلاًshm_open هنوز اجرا نشده خطای دهد.

گام ۲: نگاشت حافظه به فضای آدرس

```
void* ptr = mmap(0, SHM_SIZE, PROT_READ, MAP_SHARED, shm_fd, 0);
```

- حافظه اشتراکی به فضای آدرس فرایند نگاشت می‌شود.
- فقط دسترسی خواندن (PROT_READ) دارد.
- اگر mmap شکست بخورد، پیام خطای چاپ می‌شود و برنامه خارج می‌شود.

گام ۳: کپی داده از حافظه اشتراکی

```
char buffer[SHM_SIZE + 1] = {0};  
memcpy(buffer, ptr, SHM_SIZE);
```

- یک بافر محلی buffer از اندازه‌ی ۱۱ بايت (۱۰ داده + ۱ بايت برای پایان رشته) ساخته شده.
- داده‌ها را از حافظه اشتراکی به buffer کپی می‌کند.
- به علت صفر کردن اولیه {0}، بافر همیشه به درستی به پایان می‌رسد. (null-terminated).

گام ۴: چاپ داده‌ها

```
std::cout << "SHM Content: \\" << buffer << "\\\" << std::endl;
```

- محتوای خوانده شده از حافظه اشتراکی را نمایش می‌دهد.

گام ۵: پاکسازی

```
munmap(ptr, SHM_SIZE);  
close(shm_fd);
```

- حافظه را از فضای آدرس برنامه آزاد می‌کند.
- فایل حافظه را می‌بندد.

```
simple-web-server.sh:
```

```
#!/bin/bash

PORT=80

while true; do
    echo "Waiting for connection on port $PORT..."

    {
        read request_line
        echo "Received request: $request_line"
        response="HTTP/1.1 200 OK\r
Content-Type: text/plain\r
Connection: close\r
\r
Current date and time: $(date)\r
"
        echo -e "$response"
    } | nc -l -p $PORT -q 1
}

Done
```

: توضیح

کدی که نوشته‌یم یک وب‌서ور بسیار ساده با استفاده از bash و ابزار netcat (nc) هست که روی پورت ۸۰ گوش می‌دهد و به درخواست HTTP کلاینت‌ها (مثل مرورگر) پاسخ می‌دهد.

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

- مشخص می‌کنه که این اسکریپت با مفسر Bash اجرا بشه.

PORT=80

- متغیر PORT را برابر ۸۰ پورت استاندارد HTTP قرار می‌ده.

while true; do ... done

- یک حلقه بی‌نهایت که هر بار یک اتصال ورودی (مثلاً مرورگر) رو دریافت و پردازش می‌کنه.

echo "Waiting for connection on port \$PORT..."

- هر بار که آماده دریافت اتصال شد، این پیام رو در ترمینال چاپ می‌کنه.

{ ... } | nc -l -p \$PORT -q 1

این قسمت مهم‌ترین بخش:

سمت راست:

nc -l -p \$PORT -q 1

یعنی:

ا-حالت لیسن (یعنی منتظر اتصال باشه).

ب-پورت مشخص شده گوش بد.

ج-بعد از ۱ ثانیه بی‌فعالیتی، اتصال رو ببند.

سمت چپ: ({ ... })

کدی که به اتصال داده می‌شه (و به مرورگر ارسال می‌شه):

داخل بلوک:

read request_line

- اولین خط از درخواست HTTP دریافتی (مثلاً GET / HTTP/1.1) را می‌خونه.

echo "Received request: \$request_line"

- این خط درخواست را در ترمینال چاپ می‌کند (برای مشاهده عملکرد در لاگ).

تعریف پاسخ:

```
response="HTTP/1.1 200 OK\r
```

```
Content-Type: text/plain\r
```

```
Connection: close\r
```

```
\r
```

```
Current date and time: $(date)\r
```

```
"
```

- این ساختار پاسخ استاندارد HTTP هست:

- HTTP/1.1 200 OK

- Content-Type: text/plain

- خط خالی (دو بار \r\n) برای جدا کردن header و body.

- Current date and time: \$(date)

echo -e "\$response"

- پاسخ بالا را به خروجی می‌فرسته (به مرورگر).

نتیجه کلی:

این اسکریپت هر بار که مرورگر صفحه رو رفرش می کنه:

1. اتصال رو قبول می کنه،
2. یک پاسخ ساده با زمان فعلی می دهد.
3. در ترمینال هم اطلاعات درخواست رو چاپ می کنه.

webserver.service:

[Unit]

Description=Simple Web Server Service

After=network-online.target

Wants=network-online.target

[Service]

ExecStart=/root/myprojects/simple-web-server.sh

Restart=always

RestartSec=5

StandardOutput=journal

StandardError=journal

[Install]

WantedBy=multi-user.target

توضیح :

کدی که نوشتیم مربوط به یک فایل سرویس webserver.service به نام Systemd هاست که برای اجرای خودکار simple-web-server.sh طراحی شده است.

بخش تعریف کلی سرویس

[Unit]

Description=Simple Web Server Service

After=network-online.target

Wants=network-online.target

• **Description** توضیحی برای سرویس، که در خروجی‌های systemctl دیده می‌شود.

• **After=network-online.target** سرویس فقط زمانی اجرا شود که اتصال شبکه به طور کامل برقرار شده باشد

(نه فقط کارت شبکه بالا آمده باشد).

• **Wants=network-online.target** systemd می‌گوید که network-online.target را هم فعال کند،

اما الزام آور نیست.

[Service]

ExecStart=/root/myprojects/simple-web-server.sh

Restart=always

RestartSec=5

StandardOutput=journal

StandardError=journal

• **ExecStart** مسیر اسکریپتی که باید اجرا شود و بسرور ساده ما

• **Restart=always** اگر اسکریپت به هر دلیلی متوقف شود، سرویس دوباره راهاندازی شود.

• **RestartSec=5** پنج ثانیه بعد از توقف، دوباره راهاندازی شود.

• **StandardError=journal:** خروجی استاندارد (stdout) و خطای (stderr) به journalctl -u webserver.service منتقل شود قابل مشاهده با

journalctl -u webserver.service به journalctl -u webserver.service

بخش فعالسازی در بوت [Install]

[Install]

WantedBy=multi-user.target

- **multi-user.target** یعنی این سرویس باید در زمان بوت، وقتی سیستم در حالت چندکاربره هست (حالت استاندارد لینوکس بدون محیط گرافیکی)، فعال شود.

با این فایل، سرویس:

- پس از بالا آمدن کامل شبکه اجرا می‌شود،
- هر بار اسکریپت متوقف شود، بعد از ۵ ثانیه دوباره راهاندازی می‌شود،
- در زمان بوت به صورت خودکار اجرا می‌شود،
- لاغ‌های آن در journalctl ثبت می‌شوند.

باتشکر