Subject: آزمایشگاه سیستم عامل

۱_ linux (system programming با زبان C) نیاز داریم که linux بریزیم. خودمان نصب کنیم (اگه VMware استفاده کنیم) — share scripting

2_ لپ‌تاپ داریم حتما

3_ حضور نمره داره (اجلسه غیبت مجاز) غیبت نداشته باشم اگر نمره می‌گیریم

4_ بارم بندی ۴ نمره حضور و فعالیت سر کلاس، 8 نمره تمرین‌ها، 8 نمره امتحان عملی پایان ترم

امتحان عملی: share or Batch سوال 3 تا 2 (نمره) مثال باید از قسمت دوم

(روی Linux نصب کنیم.)

نمره‌ سوم خوب نمره می‌دن ← C رو دوره کنیم. آدرس و pointer و ..... کامپایلر gcc

"جلسه دوم" openSuse 15 v

+ power ← مرحله (power and self test) power on ⇐ BIOS (تست قطعات و امتحان می‌کنه)

stage1: حافظه [ ] نگاه می‌کنه و مشخص می‌شه چه سیستمی Boot می‌شه، مثلاً ویندوز یا لینوکس می‌خواهیم

Boot کنه، ۲ مدل Boot داریم. Master Boot Record = نرم افزار کوچک ۴۴۴ بایتی (B) اطلاعاتی نوشته شده

Boot:
- MBR (سیستم قدیمی‌تر): وظیفش Boot Loader پیدا کنه بیاره توی Ram. [پیدا کنه یعنی چی؟]
- UEFI (سیستم جدیدتر) — Meta data, Flag

MBR 444 Byte, sector 512 Byte, Partition Table 64 Byte (از ۴ بخش ۱۶ بایتی) → primary partition, refer extended partition, 2 Byte

OS روی یکی از primary part قرار داره.

partition: مدل پارتیشن‌ها هست. primary extended (داشته باشیم ۳ تا primary) قسمت ۴ اُم هم می‌تونیم extended داشته باشیم

primary تعریف کم. توی یکی از رکوردهای partition Table قرار می‌گیرد. مثلاً ۴ تا، سیستم عامل... logical ها

در Linux یه بخش به نام LVM (logical volume management) استفاده کردن و به صورت Logical انجام میده.

Magic Code: از یه Code (0xAA55) مثلاً برای قرار گرفتن OS داریم.

EFI: partition EFI (باید ایجاد کنیم.) "Boot manager" دنبال Boot Loader می‌گرده.

Boot manager یه بخشی داره به نام Secure Boot:

Grub2: BootLoader برای Linux است.

gnome, KDE

init: initializ process

IDEA

Subject:

App
shell
system call
kernel
HW

لایه بندی OS ←

جایی که قرار کار انجام بدیم از shell

تمامی کدها در Terminal انجام می‌شه.

معروف‌ترین Bash (Terminal), Zsh, CShell

درایوهایی که روی دیسک هستند با خاموش روشن شدن از بین نمی‌رود.

basic Command Linux:

فایل‌هایی که روی دیسک قرار ندارد، با خاموش روشن شدن از بین می‌رود.

(VFS: virtual file system)

① pwd: کجای سیستم قرار داره این خط دستور

/ → etc (conf file) → ادامه → Lib → library, package, Kernal Modules (همون درایور) و غیره
→ boot (Boot loader, Kernel) → Lib64
→ home → sys → system files
→ proc (vfs, process file) → tmp → temporary file
→ opt → optional
→ bin → binary, فایل‌های اصلی
→ lib, library

در یک قالب یکسان نمایش داده می‌شود روی Ram قرار دارد.

همه چیز از / شروع می‌شه.

در دایرکتوری home دایرکتوری‌های user قرار دارد.

host name
@Linux-tag1: اسم هاست

(تیلدا) ~ آدرس جایی که هستیم. ← یعنی (home/oslab) ← آدرس جایی که هستیم.

② ls: دستوری که فایل‌ها و دایرکتوری‌هایی وجود دارد.

③ cd ← دایرکتوری را عوض می‌کنه. ، cd.. ← برمی‌گرده به قبل
"change directory"

* در Linux همه چیز فایل است.

Bios نرم افزار است که تک تک قطعات رو امتحان می‌کنه.

Subject: "جلسه سوم" Alt + F2 → gnome - Terminal

آدرس ها: ① نسبی ② مطلق

① نسبی: از جایی که هستیم شروع می کنیم آدرس دهی. مثال: Download

② مطلق: از ریشه آدرس دهی کردی شه (/) شروع به آدرس دهی کنیم. /home/OSlab/Videos/

آرگومان ورودی بدیم، option محسوب می شود. چیزی که با - (خط تیره) شروع بشه option

-l (جزئیات) , -a → show hidden files/directories می ده. چیزایی که با . شروع می شه.

. → ارجاع می ده به همین دایرکتوری که هست. همین جا که است → cd . → ./
.. → یک دایرکتوری عقب تر

④ man → manuall (Linux صفحه) مثال → man ls

* manuall: سطح بندی استسطح ① Command

⑤ Clear → خط زمان را دوباره اول (پاک می کنه)

⑥ برای ساخت دایرکتوری از دستور mk dir استفاده می کنیم مخفف "make file"، و باید آدرس بدیم.

لـ بعدش باید option بدیم → -p (دایرکتوری تو در تو بسازیم) 1/2/3 mkdir -p 1/2/3

⑦ ساخت فایل: touch → نام دلخواه touch myfile

لـ از جایی که هستیم شروع می کنیم آدرس دادن.

⑧ nano myfile → باز می کنیم و چیزی توش می نویسیم

⑨ vim myfile : ۲ حالت داره، یا حالت نوشتاری یا حالت Command.

وقتی وارد می شیم حالت Command هست. یکسری Basic command داره.

vim → escap → (shift + [illegible]) → :q (با بستن)

→ [illegible] (کن save)

→ :wq (save کن با بستن)

→ :xq ≃ wq

یکی Mood Command بخواهیم بریم ESC می زنیم و اگر DD بزنیم کل خط رو پاک می کنه، و p → past می کنه.

Subject:

U=undo و yy : کپی یک خط (copy) ⑨ امادت

- search → :/bash ← حالا اگر تعداد bash زیاد بود بین این‌ها که پیدا کرده می‌خواهیم جابه‌جا بشیم از N استفاده می‌کنیم.

- :%S/bash/shell/g → (Replace) جای Bash ، shell می‌زنه
:%S/<word$/<word>/g global

- Cat → text file مشاهده می‌کنیم.

- rm → remove → rm ۱ (recursive بازگشتی)

↳ rm -r ۱

- f (استفاده نکنیم.)

- mv <source آدرس> <de_f> جابه‌جایی فایل

برای تغییر اسم از mv استفاده می‌کنیم مثلاً mv myfile myfile2

- LL etc یا Ls (l: option)/etc/ ←

+ more + Less

Pipe

Command ۱ | (pipe) Command2

خروجی اولیه بشه ورودی دومی

↳ more استفاده کنیم فقط می‌ریم جلو } → more فایل‌های خیلی زیاد
↳ less بزنیم می‌توانیم بالا و پایین بریم. } → less فایل‌های کوچک

IDEA

Subject:

تعداد خط‌هایی که میخواهیم / شروع خط

+ head: مثال ← cat myFile2 | head -n 2 (از ابتدا ۲ خط می‌نویسد)

+ tail: cat myFile2 | tail -n 3 ← 18 19 20

12 13 14 ← cat myFile2 | head -n 14 | tail -n 3

>> → append

جریان‌ها
- std out —redirect→ ls /etc/ > output ← ls ← cat myfile output
- std in
- std error ← رو جریان ارور پیام بنویسیم

صورت پیش‌فرض با ترمینال ارتباط داده‌اند.

ls -45 > output ← هر چیز داخلش بود پاک کن
برای همین وقتی cat output میزنیم چیزی نشون نمیده چون توی std out داریم میریزیم

redirect هم append هم
برای std error بزنیم 2>> باید بذاریم.
file descriptor

- stderror , stdout ← &>
- append &>>
- stdin ← <

جلسه چهارم:

Bash Script : (یعنی از چه ابزاری استفاده می‌کنیم از Bash . این ترمینال یک Bash است.)

1: mkdir session3
cd session3
vim script1.sh ← برای نوشتن Insert بزنی

* 1-1 #!/bin/bash (she bang) ← این اسکریپت که می‌نویسیم توسط چه مفسری خوانده می‌شود. اگر python بنویسیم به جای این آدرس python می‌داریم. / Permission جاب می‌داریم

1.2 echo "Hello world" ← چاپ می‌شه ← command (:x) ← میاد بیرون

1.3 alias LL='ls -l' ، md='mkdir -p'

اسم مورد علاقه بدیم؟ alias salam='ls' -

IDEA

per = permission

Subject:

1_4 -rw-r--r-- (per per per: user owner / group / other) → r ⇒ read, w ⇒ write, x ⇒ execute

نوع فایل

ترکیب این‌ها permission می‌گویند.

صاحب فایل، گروه، بقیه (۳ سطح دسترسی)

توی این قسمت نقطه می‌تونه نخونه و بنویسید!

1_5 script رو مستقیم اجرا کنیم → نمی‌تونیم → باید "Edit permission" کنیم (راه ۱)

(راه ۱) EDIT Permission : chmod

LL (قبلاً) باید بنویسیم

```
Ch mod  U+X  script1.sh   → به user اجازه اجرا بدیم.
ch mod  u-xw  script1.sh  → اجازه اجرا، نوشتن رو بگیریم (readonly)
        ug+xw  script1.sh
        ugo+rwx  script1.sh
        o-xw  script1.sh
```

راه 2: rw- r-- r-- (110 100 100 → 6 4 4) حالت بیت انجام بدیم. → chmod 644 script1.sh

چاپ کنه → ./script1.sh

```
1_6  echo -e "Hello \n world"   ( رفتن خط پایین
     echo -n
```

کامنت: comment #(تک خطی)

1_7 ورودی بگیریم → read name family (متغیر) → { read name family / echo "your name : $name $family

1_8 -s (password پنهان می‌کنه)

```
1_9 read -n1 choose
    echo -e "\n $choose"
    echo -n1 -t3
```

دستور قبلی با موفقیت اجرا شد

Subject:

```
_ ./a.out && echo success
_ ./a.out || echo fail
```

→ اگر fail بود دستور بعدی رو

```
#include <stdio.h>
int main()
{
```

```
_ ./a.out && echo success || echo fail
```

```
_ a=2
_ b=3
_ echo $a → 2
_ echo $b → 3
```

test یا اینکه [ ] بنویسیم `[ $a -eq $b ]` (برای مقایسه اعداد استفاده میشود)

space space

echo $? → جواب 1

```
[ $a -ne $b ]
[ $a -ne $b ] && echo not equal || echo equal
```

Man test (1) → استفاده میکنیم که یکسری option ها رو پیدا میکنیم مثل -eq ، -ge ، -gt ، -le ، -lt و ...

```
[ -d /etc/ ]
  echo $?
```

→ نتیجه 0

```
[ -d /etc/hosts ]
  echo $?
1
```

sudo = دسترسی یوزر

env = "متغیرهای برنامه"

۱_۱۰ (echo راه دیگر) → printf "%8s,%d, %.2f\n" salam 23 44.6666

۱_۱۱ (متغیر تعریف کنیم) → title = (OSlab) ; (مقدار)

echo $title 123

name = value بین اینا space باشه متغیر عوض می کنه.

متغیر لوکال که بیشتر valid باشد → export name=value (env)

.bashrc → هر بار Terminal باز می کنیم این فایل یک بار اجرا می شود.
└ export myvar=opensuse

"جلسه پنجم"

محاسبات ریاضی در Bash :

```
#!/bin/bash

a=2
b=3
val=$[$a+$b]
val2=$(($a*$b))
let val3=$b-$a  → ? let
echo $val1 $val2 $val3
```

- $?

- vim source.c
- gcc source.c -o
- ls
- ./a.out
- echo $?

IDEA

Subject:

```
a=2
b=3
if [ $a -eq $b ]
 then
    echo "equal"
 fi

if [                ]
the
    echo "equal"
elif [ $a -gt $b ]
then
  echo "greater than"
  then
    :
```

- Switch Case

```
#!/bin/bash

read num
case $num in
0)
    echo "shanbe";;
211)
    echo "1shanbe";;
default *)
  echo "wrong input"

esac
```

Subject:

$0 اسم اسکریپت و $1 آرگومان . $# تعداد آرگومان‌هایی که به اسکریپت ارسال می‌شود

دستور shift آرگومان‌های ورودی را shift می‌دهد

اگر حلقه shift ← کل آرگومان ورودی را بخوانیم

exit 0 ← مقداری که آدم بیرون را چاپ می‌کند.

«جلسه ششم» ۸٫۸ ادامه Bash

مثال shift:

```
#!/bin/bash
echo $0, $1 $2
shift
echo $0 $1 $2
```

(exe دستور ← chmod u+x script1.sh)

* زبان Bash یک زبان functional است یعنی یک function تعریف کنیم یعنی برنامه را به چند function تقسیم کنیم

تعریف Function:

تعریف تابع () اسم تابع ←

```
add () {

    echo "addi"

}
    echo "start"
    add
echo "end"
```

← دریافت

می‌خواهیم ۲ تا متغیر بگیریم داخل یک تابع کنیم بعد با هم جمع کنیم.

```
add () {
  sum =$[$1 + $2]
   echo $ sum }
  echo "start"
  add 7 3
  echo "end"
```

ÍDEA

Subject:

Subject:

حالا می‌خواهیم تابع که متغیری را Return کند

```
add (){
  sum=$[$1+$2]
}
echo "start"
add 7 3
echo $sum
echo "end"
```

* اگر پشت متغیر کلمه "local" استفاده کنیم متغیر scoop می‌شود اینی که تعریف می‌کنیم

یعنی بالا پشت local sum استفاده کنیم دیگه مقدار چاپ می‌کند.

* حلقه و تعریف List (for each جلو)

مثال 1:
```
*For i in 1 2 3 4 5 6 7 8
do
    echo $i"
done
```

مثال 2:
```
* for city in tehran Mashhad Tabriz 9 10
do
   echo $city
done
```

مثال 3:
```
* for j in {100...112}
do
   echo $j
done
```

مثال 4:
```
* for i in {a...m}
do
  echo $i
done
```

مثال 5:
```
* for path in /home/aslab/*
do
   echo $path
done
```

num=1
"num [][]$: um" ⟶ num[]:[]1
'num[]:[]$num' ⟶ num[]:[]$ num

IDEA

for C style

Subject

شکل ۱ *
```
for ((n=1;n<=10;n++))
do
   echo $n
done
```

کوچکتر مساوی رو نداریم یعنی اینجا می‌شه مثل Compare عددی (test)
و باید حتماً Le بذاریم

while و for ← یک حلقه شمارشی
← غیر شمارشی

```
k=2
while [ .k  -le  35 ]
do
   echo $k
   let k=$k+2
done
```

چه جوری یک while true بنویسیم؟

```
while:
do
read input
[ -z $input ] && break || echo "your input: $input"
```

Zero Length

- تعریف آرایه:

```
declare -a myArray (روش تعریفی هست)
```

روش راحت‌تر
```
myArray[0]=12
myArray[1]=13
```

روش معمول‌تر
```
myArray=( 1  2  3 )
# myArray+=(23)
echo ${myArray[1]}
```
← بهش 2 تا می‌ده

Subject:

myArray[122] = 1004

```
echo ${myArray[1]}
echo ${myArray[122]}
echo ${myArray[@]}
echo ${myArray[*]}
```

نتیجه ← 1 2 3 1004

```
for i in ${myArray[@]}
do
    echo $i
done
```

```
1
2
3
1004
2
```

```
echo ${myArray[@]:1:2}      ← 2 3 چاپ می‌کنه
echo ${myArray[@]/2/512}    ← 2 رو تبدیل کن به 512
```

(index کجا شروع کنه / چند تا)

* نکته فرق @ و * چی هست؟ (در بالا)

```
names=("Mohammad Karimian","ali karimi")
for name in "${names[@]}"
do
    echo ${names[@]}
done
```

* کل آرایه کانکت می‌کنه با spacel برمی‌گرده
@ نداریم ←

IDEA

Subject:

تمرین سری اول: اسکریپتی که directory ، غیر directory ها رو هم جدا کنه. (آدرس هم آرگومان ورودی آدرس است.)

* Mantest نیاز داریم.
* سری آرگومان نیاز داریم.
* حلقه نیاز داریم.

*** جلسه هفتم ۲۴،۸،۰ ***

ادامه " Bash script " :

strings :

```
* STR="hello world"
echo $STR
```
نتیجه → hello world

```
echo ${STR}
```
→ hello world

```
echo ${STR}oslab
```
→ hello worldoslab

```
echo ${STR#he}
```
→ llo world

```
echo ${STR/world/}
```
→ hello

```
echo ${STR/world/oslab}
```
→ hello oslab

```
* str2="1232527"
```
echo ${STR2} → 1232527

echo ${STR2/2/8} → 1832527 (اولین 2 که پیدا کرد تبدیل به 8 کرد.)

echo ${STR2//2/8} → 1838587 (تمامی 2 ها رو به 8 تبدیل کرد.)

echo ${#STR} → 11 → طول آرایه به دست

echo ${STR:3:5} → lo wo → Sub string (می‌گیره) از کاراکتر سوم، 5 تا بعدش رو جدا می‌کنه.

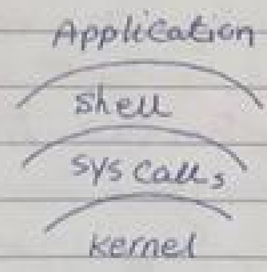
Subject:

system programing : (sys calls) : راه ارتباطی با سیستم عامل

وظیفه‌ی سیستم عامل: واسط بین user و سخت افزار. توی سطح بالا انجام می‌دهد. به اشتراک گذاشتن

preprocesser

یک واسط بین نرم افزار و سخت افزار می‌شود.

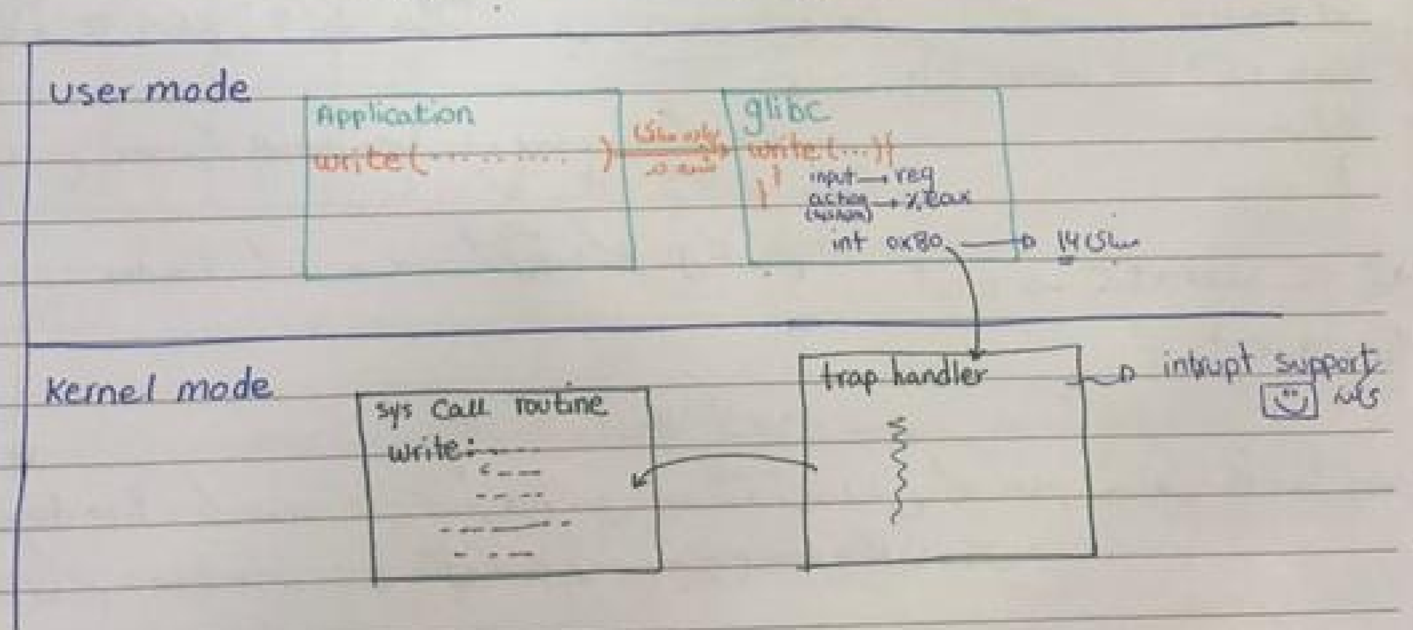


می‌خوام با syscall کار انجام بدم مثل اینکه disk نوشته می‌شود
Block می‌شود تا زمانی که جواب از kernel بیاد.

user mode   kernel mode

تفاوتشان یک flag هست.

در Kernel mode به همه چیز دسترسی داره
ولی در user mode آدرس‌های محدوده‌ی خودش رو دسترسی داشته باشه نمی‌تونه.



چه چیزهایی در write وجود داره؟ مثلاً "Hello world" حالا باید که kernel رو صدا کنه و write کنه.
یه جای سریع مثل Register (یک سطحی از حافظه) ← بین input که سریع حاضر شه توی Reg
دومین کار هم اینکه بگم چه action می‌خواهم انجام بدم action در %eax (حافظه) قرار می‌دهم.
حالا باید یک interupt بدم که بین Kernel و User، Switch بشه.
↲ int 0x80 باعث می‌شه که mode عوض شه و بره kernel، بعدش وارد trap Handler
از trap handler میاد سراغ sys call routine و write رو انجام می‌ده

Subject:

32

→ nasm -f elf64 hello.asm

(فایل‌ها رو میاره) یا ... دیس

linker را ... Ld -o hello hello.o

کسایی obj file → compile → exec.table

linker یعنی o هایی که ایجاد می‌شه رو هم link می‌کنه.

"جلسه هشتم" ۹۸،۸،۱۹

"system call files"

در linux هر کاری انجام بشه باید با فایل انجام بشود

* بخواهیم با فایل کار کنیم باید سیستم هر نوع فایلی است؟ شده انجام این کار ساده کرده است

← syscall های linux اینن تمام فایل‌ها استفاده کردن که این نوع syscall های universal syscall

← ۴ نوع هستند

مثال / یک برنامه ساده که یک syscall رو صدا کنیم

```
① mkdir session7
② vim script.sh
   #include <fcntl.h>
   int main(){
   write(1, "hello world!", 12);
   return 0;
   }
③ gcc source1.c -o a.out
④ ./a.out
```

file descriptor

12 character

مثلاً ۱۲ تا داریم تا ۱۲ کاراکتر چاپ می‌کنه ← این (hello world!) هستن

این تابع write که hello world! که چیو می‌نویسه ۱ کار رو انجام می‌ده

file des یک عدد است که می‌گه با کدوم فایل کار می‌کنیم

file descriptor: یک عدد است نشان‌دهنده فایلی است که برای عملیاتی کار انجام شه یعنی مثلاً فقط خواندنی باز می‌کنیم یا فقط نوشتنی ← فایلی که باز می‌شه یک file descriptor (شماره) می‌ده. (برای این برنامه)

این ۱ برای stdout است. ← این برنامه رو کجا اجرا می‌کنیم؟ داخل terminal اجرا کردیم

وقتی برنامه اجرا می‌شه از parent هر کدوم این فایل‌ها رو به ارث می‌بره. (این فایل‌ها file descriptor ها) ←

0 stdin
1 stdout
2 stderr

این file descriptor ها در جدولی نگهداری می‌شه؟ داخل ۲ تا جدول نگهداری می‌شوند.

در داخل هر process یک file des یک flag یک ptr دفتر preprocess:

process A

fd	flag
1	close on
23	exec

تنظیم flag دخیره می‌شه داریم "close on exec" است.

process رو کول می‌کنه یعنی از terminal ... دیگر تبدیل می‌شه!

process B

fd	flag
46	
50	

شماره فایل

file descriptor (system_wide)

offset	flag	ptr
[illegible]	read, write, ...	

در ptr به جدول فایل‌ها در ptr ها اشاره
می‌کند که هر کدوم به کدوم سطر این جدول map می‌شوند

ممکنه دو تا process به یک ردیف اشاره کنند که قبلاً توسط parent
باز شده باشند.

VarLog: در این جا یک log هر process حاصل را به انتها نوشته می‌شود یعنی
همه یک offset مشترک داشته باشند.

همچنین حالت ۲ process یک جدول را با چند تا offset داشته باشند.

این ptr به جدول i_node اشاره می‌کند

جدول i_node

Address	owner	lock	.....

این ۳ تا جدول از لحاظ ذخیره‌سازی چه تفاوتی با هم دارند؟

① i_node ثابته ولی این ۲ تا جدول تغییر می‌کند.

② i_node روی سیستم ذخیره می‌شه

③ i_node یک تعداد محدودی دارد

inode [illegible]

P

فایل ?

df -h : اطلاعات file descriptor

df -i : اطلاعات i_node

اگر کامنت "stat source1.c" بزنیم اطلاعات i_node برای فایل source1.c رو نمایش می‌دهد.

"open" syscall :

* syscall ها در Lvl 2 هستند , Lvl 1 Command ها هستند open(2) — man open

read(2), write(2), lseek(2)

از man باید استفاده کنیم.

Subject:

```
include <fcntL.h>
include<sys/types.h>
include<sys/stat.h>
include <stdio.h>
int main(){
  //write(1, "hello world!", 12);
  int fd = open("new file.txt", O_RDWR|O_CREAT, S_IRWXU);
  printf("%d\n", fd);
  return 0;
```

mode

"write" : syscall. جلسه کم

```
SSize_t write(int fd, const void * buf, size_t count);
```

* count باید ارسال کنیم.

مثال

```
* #include <fcntl.h>
 #include <sys/types.h>
 #include <sys/stat.h>
 #include <stdio.h>
 #include <unistd.h>

 int ( ){
 int Fd = open("newFile.txt, O_RDWR|O_Create, S_IRWXU);
 int retw = write(Fd, "hello world!", 5);
 printf("%d\n", retw);
 return 0;
```

→ .\a.out → 5 , cat source2.c → hello

12-5 فرق → .\a.out → 12 , cat " → hello world!

IDEA

Subject:

توضیحات در voice : Sys Call "Read"

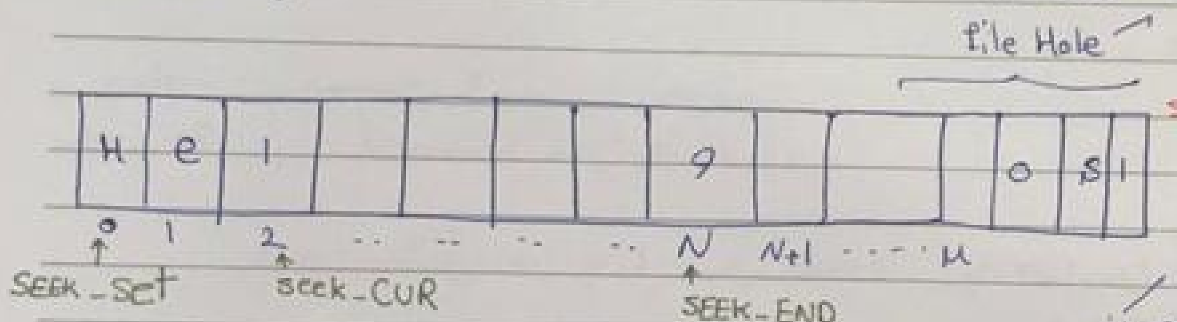
```c
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <stdio.h>

int main(){
int fd = open("newFile.txt", O_RDWR|O_Creat, S_IRWXU);
int retw= write(fd, "hello world!", 12);
printf("%d\n", retw);

Char buf[100];
int retr= read(fd, &buf/100);
printf("%d, %S\n", retr, buf);
return 0;
```

متنی که وسط کار داریش می‌نویسیم.

Lseek
sys call "modify"



با Lseek offset را reposition کنیم.
یعنی باید pointer رو ببریم جاهای دیگه بخوانیم. SEEKEnd-5 ← از آخرین offset، ۵تا به عقب

SEEK_CUR یعنی جایی که الان هستیم، ۳تا جلوتر یا
مقدار منفی هم می‌توانیم بدیم.

SEEK_set، 12 ← یعنی file offset 12

۴۰۹۶ تا
block device، character device : block اندازه در ذاتش ببینیم، که اینجا به صورت بلاکی می‌نویسه

قرار باشه بنویسه یک بلاک رو کامل می‌نویسه

IDEA

Subject:

مطالب تکمیلی:

```
int main (int argc, char ** argv)
{ int i=0;
  for( i=0; i<argc ; i++)
  { printf ("arg[%i]: %s\n", i, arg[i]); }
  return 0;
```

← آرگومان های ورودی رو برمی‌گردونه

⑧ در man قسمت Lv13 می‌شه library

* می‌تونیم <string.h> رو include کنیم

man 2 close →

echo "hello" | tee -a output.txt

cp هم مثل t command است.

جلسه دهم ۹،۱۳

+ pstree → commond ، linux ساختار درختی در

+ ps -ef → رابطه کلی تر با هر command

تعریف process: به برنامه در حال اجرا، process می‌گویند pid , ppid parent process id

انواع processها

Normal → که وی اجرا شدنش رو دارم می‌بینم

daemon → مثل network manager توی Background اجرا می‌شوند که یک Task رو اجرا کنند

orphan → هر process یک parent داره، اگر parent kill باشه، ۲ حالت < بسته می‌شود / یتیم می‌شود و باید به یک parent اجرا شه و init می‌شه (asign process)

Zombie → از بین رفته اند و منابعشان را آزاد می‌کنند.

Zombie چگونه ساخته می‌شود؟ وقتی parent، child رو درست می‌کنه و منتظر می‌شه که ببینه در چه حالتی آزاد می‌شود. وقتی parent بره توی حالت suspend، برنامه به این تعلق می‌گیره، child باید گزارش بده که kill شده یا مرحله اجرا (terminate شده) چون parent suspend هست نمی‌تونه گزارش بده بنابراین این child تبدیل به Zombie می‌شود.

Subject:

child process را بسازد درست بکنه.

یه child process درست کنیم ا (یعنی یک برنامه می‌نویسیم که بشه یک process).

* sys get pid ، sys get ppid

man pId ← Getpid → process id می‌ده (unistd.h)

```
mkdir session9
cd session9
vim source.c  →   #include <unistd.h>
                  #include <stdio.h>
                  int main() { printf("PID: %d\n PPID: %d", getpid(), getppid());
gcc source.c -o a.out   return 0; }
./a.out
```

pcounter

Code	Data	heap	gap	stack
کد ماشین برنامه	Constant variable Global	Dynamic memory allocation زمانی که داریم		جای variable های که در داخل تابع ها معرفی می‌کنید.

** از sys fork() برای ساخت child استفاده می‌شود. یعنی memory رو عیناً تو یک state درست کپی می‌کنه و تحویل process دوم می‌ده، pointer دومی همین جایی است که memory می‌ده.

Header همون اوامر برنامه بالا

```
int main() {
  int a = 10;        // جا تا داخلش قرار بگیره
  a++;               // یعنی مقدار a می‌شه
  printf("a: %d\n", a);   // مقدار a جایی می‌شه
  pid_t pid = fork();     // جایی که fork می‌زنیم یعنی process ساخته می‌شه و یکی از PA داخلش قرار می‌گیره
                          // a=11 => stack در برنامه
  if (pid == 0) { printf("Child process :\n PId : %d  PPID: %d  a: %d\n", getpid(),
  getppid(), a); } elseif (pid > 0) { printf("parent process :  PID: %d  ppid: %d
  a: %d\n", getpid(), getpid(), a); } else perror("error!");
```

Subject:

جلسه ۱۱ام ۹۸/۹/۲۰ Home38

برای این که بتوان process image را عوض کرد می‌توان از دستورهای exec استفاده کرد.

- exec
  - execl (list) ← لیست آرگومان می‌گیره — لیست می‌گیره
    - execle
    - execlp
  - execv (vector) — وکتور می‌گیره
    - execve → enviroment variable
    - execvp → full path search

حرفی از exec استفاده می‌کنیم p.

```
#include <unistd.h>
#include <stdio.h>
    pId_t pid = fork();
    if (pid == 0) {
      execlp("mkdir", "mkdir", "-p", "1/2/3", Null [illegible] );
      printf ("Child process: PID: %d PPID: %d/n  , getpid(), getppid());
    elese if (pid > 0) {
          prinf ("parent process : PID: %d PPID: %d\n", get pid(), get ppid();
    else {
        perror("error!"); }
        return 0;
```

آدرس می‌دهم — مسیر رو باید مشخص کنیم

لازم نیست دیگه آدرس کامل بدیم — [illegible] printf این جا بعد از نوشته شود.

gcc source.c -o a.out
./a.out

حالا حرفی از حرف‌های p استفاده کنیم p
* باید آرگومان‌ها رو در قالب یک لیستی از رشته‌ها بدیم
[illegible] متغیرهایی که با " " مشخص می‌شه const char* است.

```
pid_t pid = fork();
  if (pid == 0) {
    printf("child process : PID: %d/n PPID: %d /n, getpid(), getppid());
    Char  ** argv = { "uname", "-a", Null };
    execvp ("Uname"); }
  else if (pid > 0) { printf ("parent process : PID: %d PPID: %d\n", getpid(), getppid());
  else {
      perror ("error!");
  }
```

IDEA

GeTENV(3)

هر چیزی که بخواهیم مقدارش را بدانیم می‌توانیم از getEnv استفاده کنیم

از دستور export می‌توانیم استفاده کنیم که بشه متغیر env   OSLabENV=oslab981

چون توی برنامه‌ای که اجرا کردیم دیده نمیشه ← Null ، خروجی

export OSLABENV=oslab981 ← خروجی = oslab981

gets ← string را می‌گیرد

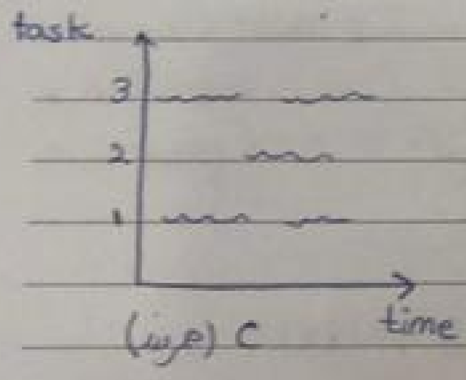
برنامه‌ای بنویسید که gets انجام بده و دستور مثلاً ls یا date خروجی تولید کند فقط.

جلسه دوازدهم

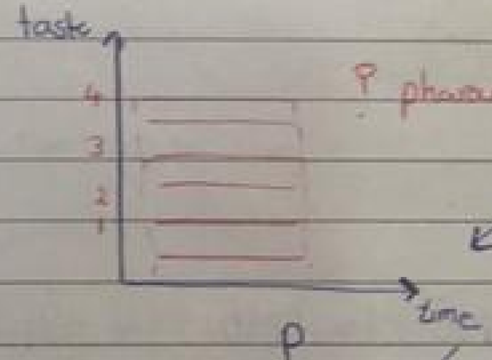
امکانات پردازنده:

ضرب ماتریس: $[\ ]_{2\times3}\ [\ ]_{3\times2} = [\ ]_{2\times2}$

کدوم کارها را می‌توان multithread؟ محاسباتی که به هم وابسته نباشند (قابلیت موازی) انجام داد.



(عربی) C



p

فرق concurrent , parallel ؟

* همزمان جدا جدا

با cpu نمی‌توان parallel انجام بده ـ در Gpu ها هستند. تکنولوژی CUDA

در miner ها ، Gpu استفاده می‌شد.

* چرا در سایر پردازنده‌ها از Gpu استفاده نمی‌شود؟ چون سرعت ـ می‌تواند context switch انجام بدهد.

*man pthread_create → درباره thread

Subject:

برای قابلیت Thread ها استفاده می‌شود

برنامه‌ای می‌نویسیم که چند تا پردازش داشته باشیم.

```
vim source.c
#include<stdio.h>
#include<unistd.h>
int add( inta, intb){
return a+b;}
int sub(inta, intb)
{return a-b;}
void execute (int(*func)(int, int), inta, intb)
{ printf("result: %d\n", func (a,b));}
int main ()
{
execute(add, 2, 3);
execute(sub, 8, 2);
return0;}
```

قابلیت Threads استفاده کردن از این استفاده می‌کنیم

interface [illegible] OS [illegible] posix

[illegible] windows [illegible]

از sys call fork استفاده کنیم.

[illegible] thread [illegible] pthread [illegible]

thread سازی

تابع callback: به عنوان آرگومان به یک تابع دیگر ارسال می‌شود تا در زمان دیگری به آن ارجاع شود.

```
#include <stdio.h>
#include<unistd.h>
#include <pthread.h> → ساخت Thread
void * func1 (void *input)
{ char *str= (char*) input;
int i=0;
for(i=0; i<20; i++)
{ printf("%s: %d\n", str, i); if(i==5)
sleep(1); }   pthread_exit((void*) "Good bye");
int main{
pthread_t p1, p2;
pthread_create(&p1, Null, func1, (void*) "thread1");
pthread_create (&p2, Null, func1, (void*) "thread2");
//sleep(22);
void* output;
pthread_join(p2,&output);
printf("ret: %s\n", (char*)output)
return 0;
}
```

برای اجرا از این استفاده می‌کنیم.

```
gcc -lpthread source.c -o a.out
    -pthread
./a.out
```

با sleep [illegible] با این [illegible]

از تابع pthread_join استفاده می‌کنیم.

[illegible] pthread [illegible] main [illegible] separate [illegible]

```
void * func2 (void * input){
Char* str= (char*) input;
int i=0;
for(i=0; i<10; i++){
printf("B\n");
sleep(1); }
```

func2 عکس

IDEA

(C کد می‌نویسن) "هاشه کلیشه ترین ها"

Bash , system program

Subject: اسامی 8:15 → اسامی

جلسه 13 (آخر) تغییر مسیر کاری ← آشنا می‌شیم . اگه آرگومان concat می‌شه ... child باید بشه ↓ voice

همزمان سازی : یک رشته ... بن بست ... ترتیبی

```c
#include <stdio.h>
"         <unistd.h>
"         <pthread.h>
"         <semaphore.h>

sem_t sem1, sem2;

void * func1(void * input){
  char * str = (char *) input;
  int i=0;
  for(i=0; i<10; i++){
    sem_wait(&sem1);
    printf("B\n");
    sleep(1);
    sem_post(&sem2);
  }
}
void * func2(void *input){
  char * str = (char *)input;
int i=0;
  for(i=0; i<10; i++){
    sem_wait(&sem2);
    printf("A\n");
    sleep(1);
    sem_post(&sem1);
  }
}
int main(){
pthread_t p1, p2;
sem_init(&sem1,0,0);
sem_init(&sem2,0,1);
pthread
//
void *output
  pthread_join(p2,&output);
  printf("ret:%s\n",(char*)output);
  return0;
sem_destroy(&sem1);
sem_destroy(&sem2);}
```

sem1=0;
sem2=1;

ABAB

tA	tB
D(sem2)	D(sem1)
printf(A)	printf(B)
u(sem1)	u(sem2)

ABAB

IDEA