01.10.2019

Lab – 30p; min. 15p pt. examen
- activitate 20p
- colocviu 10p (săpt. 7/8)
- prezență obligatorie la toate lab.
- fără prezență nu intră în examen

Proiect – 20p.
- echipe de 2-3
- punctaj identic pt. fiecare membru
- ales dintr-o listă
- lucru la lab. după colocviu

Examen – 50p., min. 25p; cu materiale; 2h.

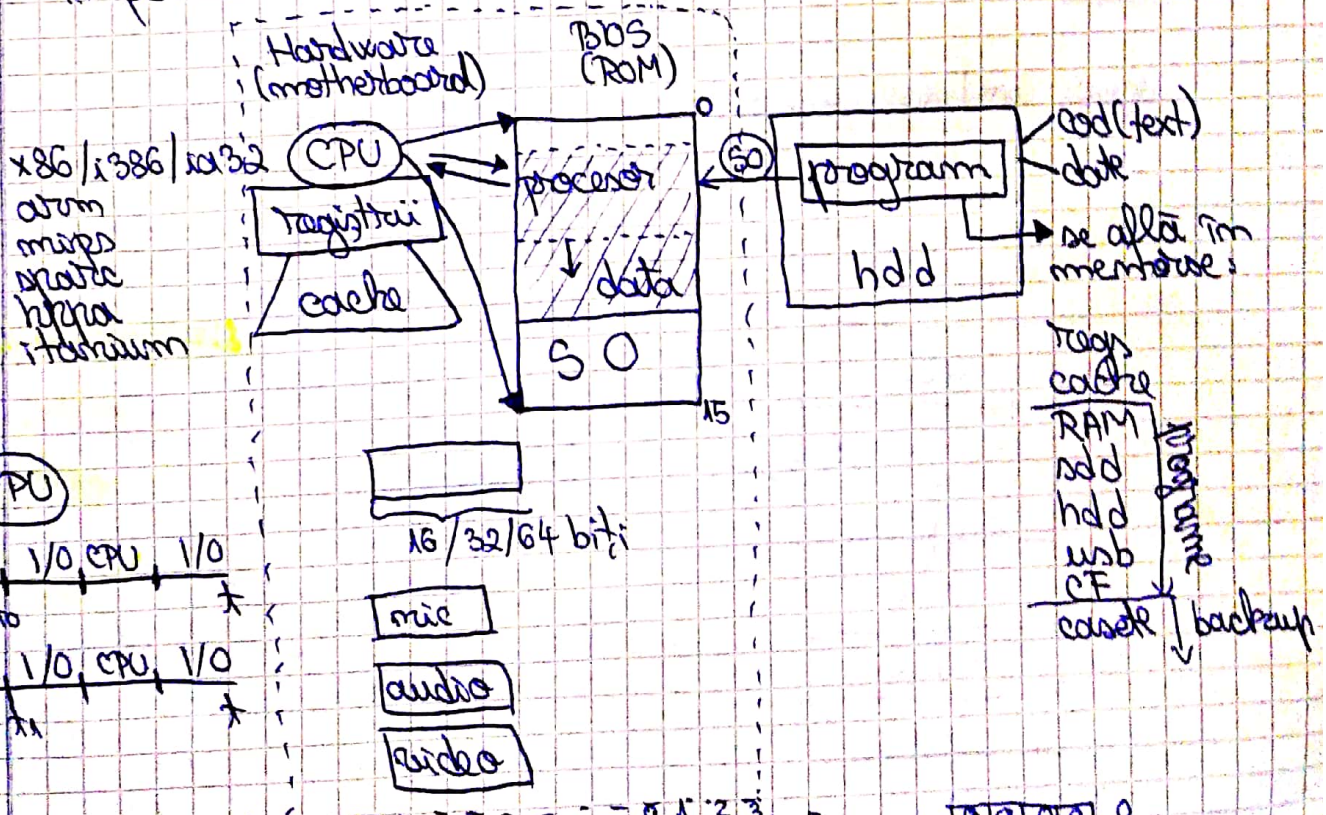
total min. 50p. pt. promovare

https://cs.unibuc.ro/~pirofti/so.html

Operating system concepts essentials

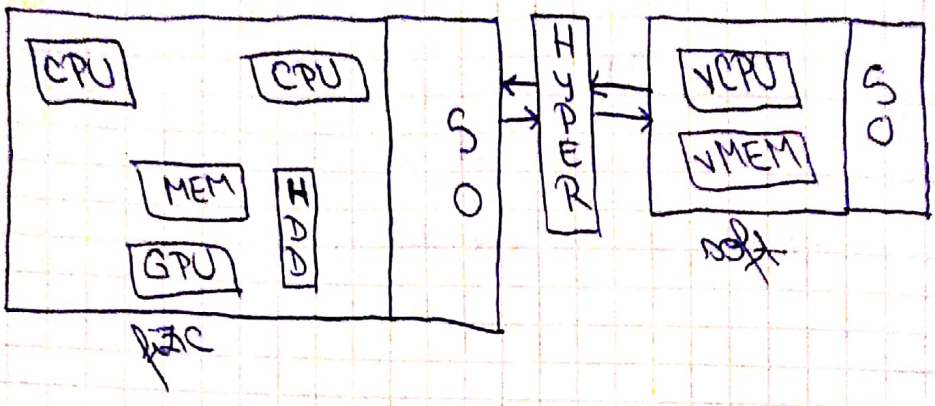
codex.cs.yale.edu/avi/os-book/OS9/

implementare în OPEN BSD



pt.: arhitectură pe 4 biți:

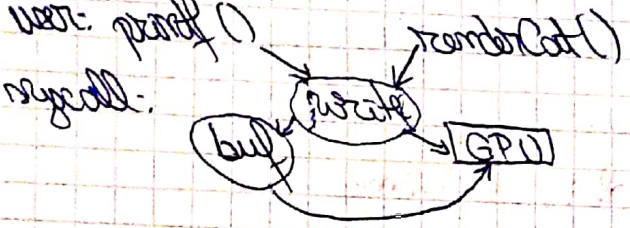
08.10.2019.

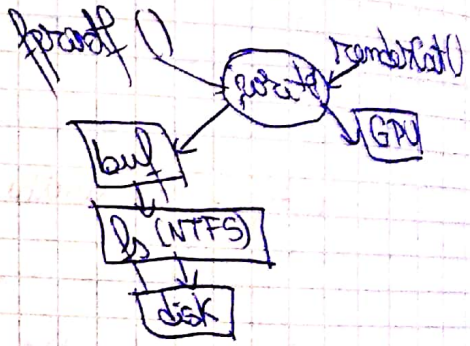


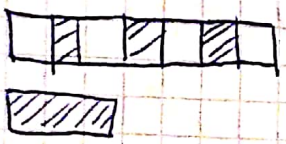
Oracle VM VirtualBox Manager

New → creez mașină virtuală nouă

după ce o faci → setări

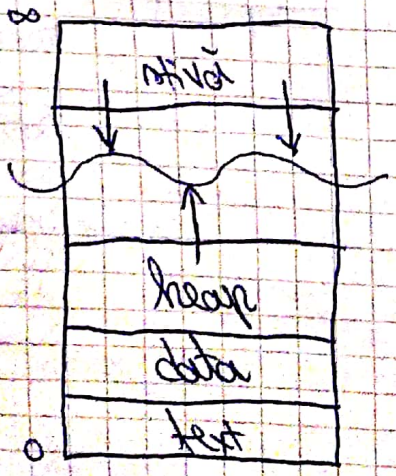


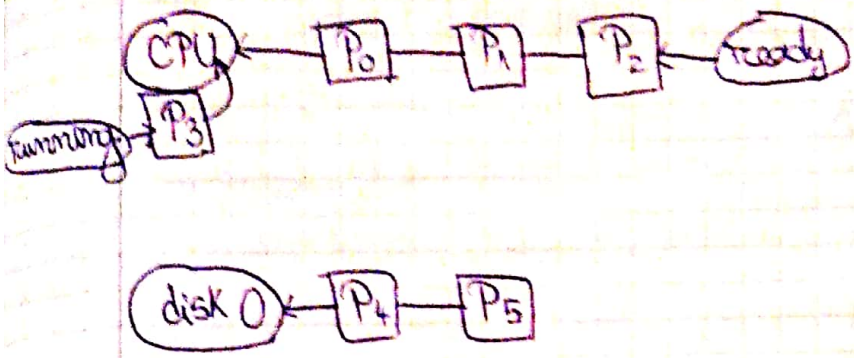




github.com/openbsd/blob/master/sys/kern/syscalls.master.

15.10.2019.

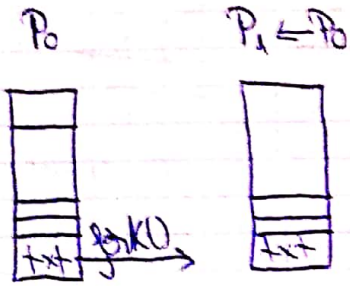




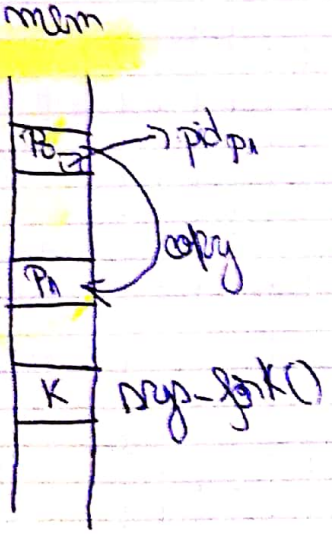
22.10.2019.

4

fork()



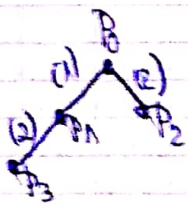

```
...
pid_t pid = fork();
printf("hello!");
if (pid == 0)
    printf("Child");
else if (pid > 0)
    printf("Parent");
```



EXAMEN:

(1) pid = fork()

(2) pid = fork()



	P_0	P_1	P_2	P_3
pid 1	$\neq 0$	0	$\neq 0$	0
pid 2	$\neq 0$	$\neq 0$	0	0

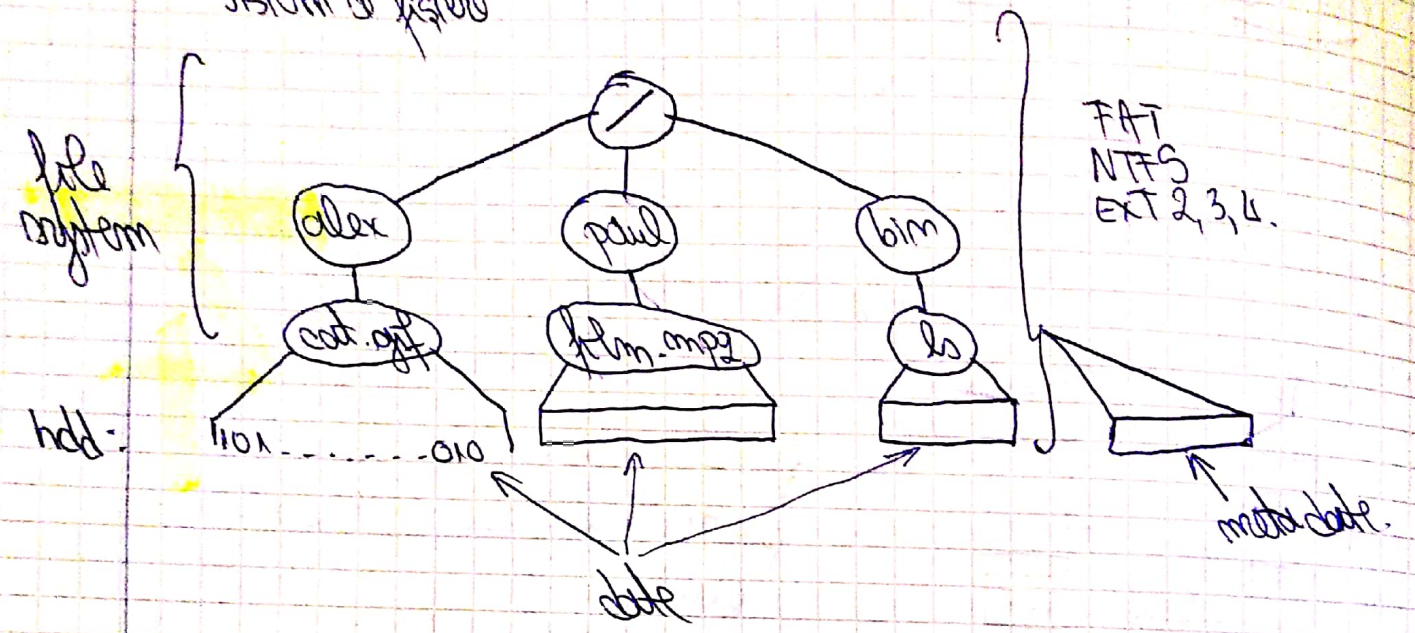
părinte | copil

pid = fork();
if (pid > 0)
    fork();

if (pid == 0) {
    exit(0);
    fork();
} metodă alternativă, anonimă

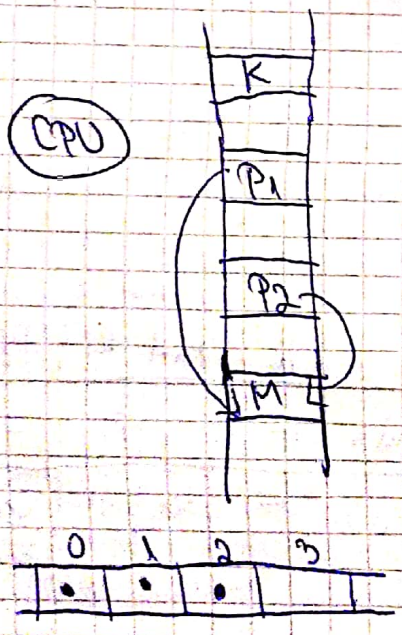
```
pid= fork();
if(pid==0)    //child
   pid= getpid();
```

Sistem de fișiere



29.10.2019.

5



P_1: M[0] = 'a'

P_2: M[0] = 'b'.

t_0: im=0, out=0

t_1: im=1, out=0

t_2, t_3: im=3, out=0.

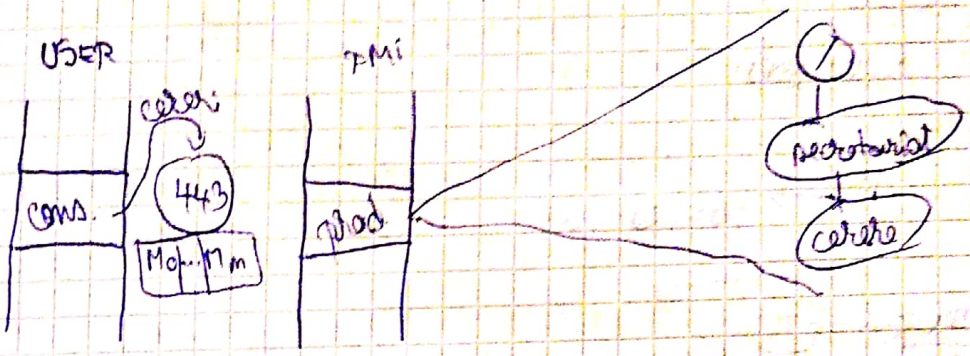
t_4: im=3, out=1

t_5, t_6: im=3, out=3

t_4: im=3
out=1

t_5: im=0
out=1.

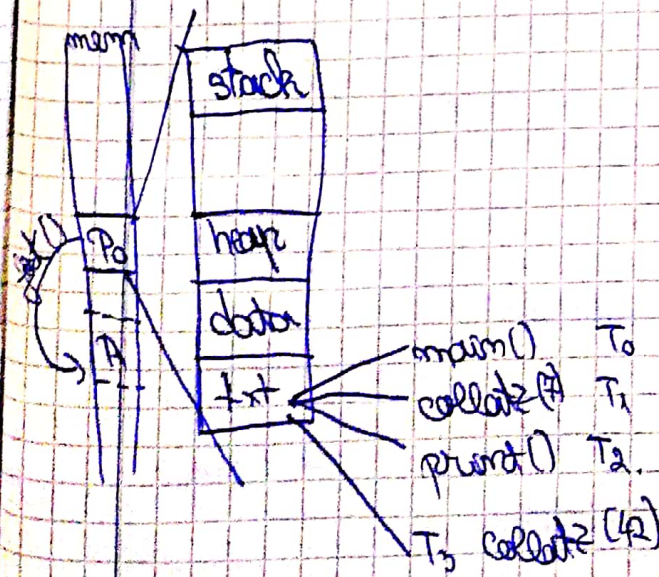
https://fmi.unibuc.ro:1443/secretariat/citire.pdf.

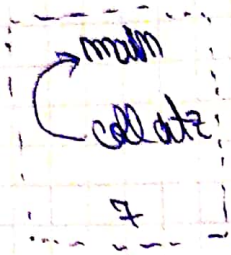


user	group	others
rwx	rwx	rwx
6	6	6
rw-	rw-	rw-
rw-	r--	---
6	4	0

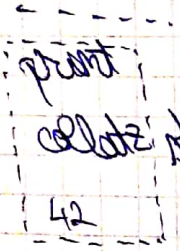
05.11.2019

Fire de executie

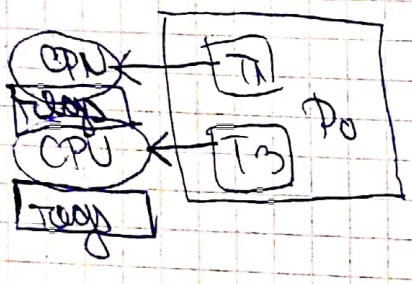




1 → rax, eax, ax
64 32 16



1 → rax, eax, ax.



T1
lock()
v = pod[i]
i--
unlock()

T2
lock()
pod[i] = v
i++
unlock()

T1 v = pod[i]
T2 pod[i] = v

atomic: v = pod[i--]
pod[i++] = v

18.11.2019.

Sincronizare de procese

P0 | count=3. | P1

(1) count++ c=c+1

(2) print count=>4

(3) count-- $c^{(1)}=c^{(0)}-1$

(4) print count=2

$t=c^0$

$c \leftarrow t-1$.

instr.	1	2	3	4
count	3	4	4	3

=> 3

3	4	1	2
3	2	2	3

=> 3.

1	3	2	4
3 → 4	4 → 3	3	3

=> 3

3	1	4	2
3 → 2	2 → 3	3	3

=> 3.

P1: count--
(1) reg1 ← read c
(2) sub ~~add~~ reg1, 1
(3) c ← write reg1

P2: count++
(4) reg0 ← read c
(5) add reg0, 1
(6) c ← write reg0

Ordinea ideala: 456 123 | 123 456.

c=3

instr.	1	2	4	5	3	6
c	3	3	3	3	2	4
reg1	3	2	2	2	2	2
reg0			3	4	4	4

```
pthread_mutex_t mtx;
mutex_lock(&mtx);
/* critical section */
mutex_unlock(&mtx)
/* reminder section */
```

Pi

P1, P2, ..., Pn

zoothr C=AB ; D=EF | X=CD.

init S=? zoo S=wait(S) | S

P_j

```
do{ PRE: while (turn == i)
    CS: /* critical sect. */
    POST: { turn = i;
            /* ... */
    }while(1).
```

P_i P_j

turn = {i, j}

turn = i.	PRE	CS	POST
P_j	P_j	blocat	
P_i	trece ✓		

turn = j			
P_j	trece ✓		o
P_i	blocat ∅		bine

26.11.2019

P_i

```
while(turn==j);
.....
turn=j;
---
```

P_j

```
while(turn==i);
.....
turn=i;
---
```

P_i

```
flag[i]=true;
turn=j;
while(flag[j] && turn==j);
---
flag[i]=false;
```

P_j

```
flag[j]=true;
turn=i;
while(flag[i] && turn==i);
---
flag[j]=false;
```

Presupunem P_i și $P_j \in$ Entry Section.

$\Rightarrow P[i] \vee P[j] \rightarrow$ Critical Section

(turn = j ∨ turn = i).

FALSE

Presupunem $P_i \in$ Entry Section, $P_j \in$ Reminder Section

$\Rightarrow$ flag[i] = true

turn = j

flag[j] = true

$\Rightarrow P_i \in$ Critical Section

$P_i \wedge P_j \in$ EntrySection

flag[i] = flag[j] = true;

turn = i ∨ turn = j

```
testandset (bool *target) {
    bool rv = *target;
    *target = true;
    return rv;
}
lock = false;
while (testandset (&lock));
...
lock = false
...
```

P_0, P_1 | $P_0 \in$ CriticalSection ⇒ lock = true

$P_1 \in$ EntrySection ⇒ $P_1 \in$ EntrySection

$P_0 \in$ Reminder ⇒ lock = false

$P_1 \in$ EntrySection ⇒ $P_1 \in$ CriticalSection, lock = true

$P_0, P_1 \in$ EntrySection ⇒ lock = false

$P_0, P_1 \Rightarrow P_0$

$P_1, P_0 \Rightarrow P_1$

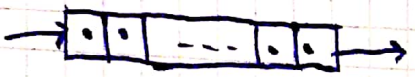
```
int compareandswap (val, exp, new) {
    int rv = *val;
    if (*val == exp)
        *val = new;
    return rv;
}
```

compareandswap (&lock, 0, 1) ⟺ testandset (&lock)

key = true;
while (waiting[i] && key)
    key = tas(& lock);
waiting[i] = true; (⇐ flag[i])

03.12.2019.

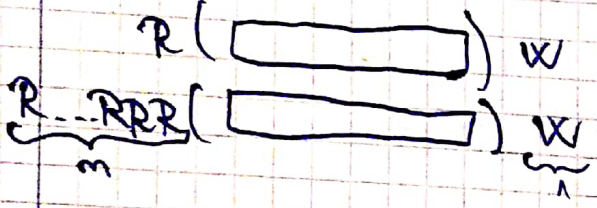
BOUNDED BUFFER PROBLEM

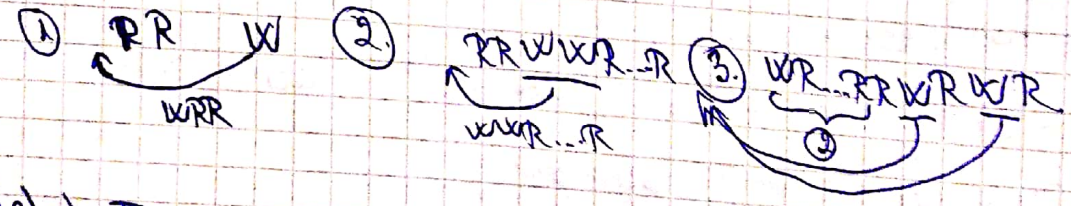


9

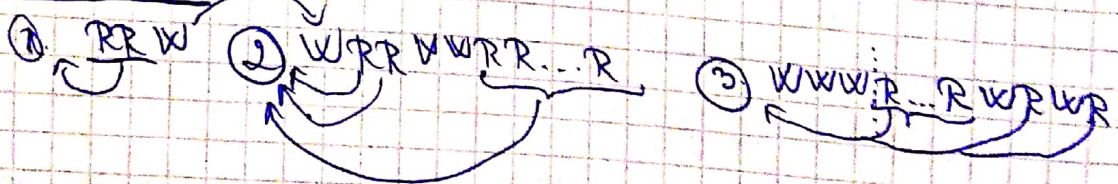
READERS/WRITERS PROBLEM.

Metoda I:

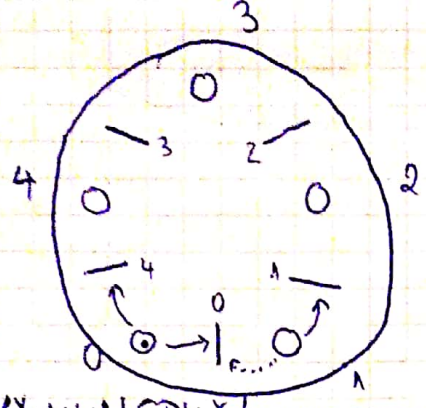




Metoda II:



EXAMEN!



THINKING, HUNGRY, EAT.

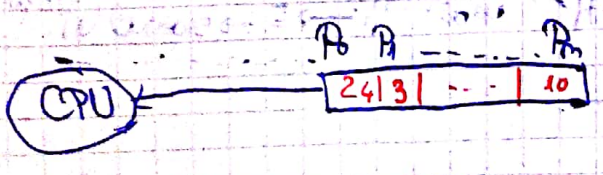
```
/* HUNGRY */                          ← lock
do{ wait(chopstick[i])
    wait(chopstick[(i+1)%5])
    /*EAT*/
    signal(chopstick[i])
    signal(chopstick[(i+1)%5])        ← unlock
    /* THINKING */
}while(1);
```

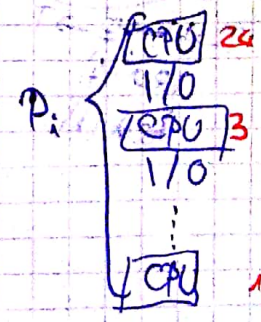
chopstick[5] = {1, 1, 1, 1, 1}

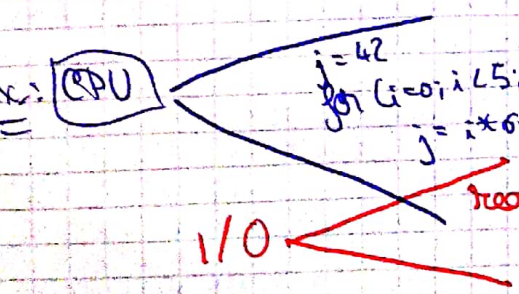
```
decrease(val)
{ if(val > max)          ← lock ✓
                         ← unlock ✓
    return;              ← lock X
  c = c - val;           ← unlock X
  return;
}
```

t_0 { T_0: c=5, val=3
T_1: c=5, val=3.

t_1 { C = 2
T_1: c=5, val=3 } $\Rightarrow$ C = -1

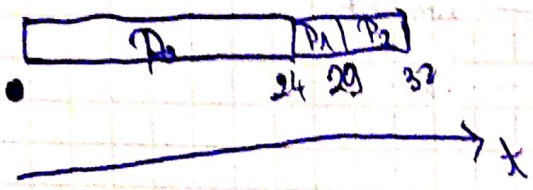




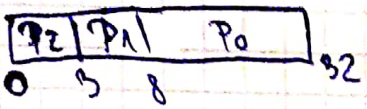


Proces	CPU
P_0	24
P_1	5
P_2	3

ALGORITMUL FCFS (FIRST COME FIRST SERVED)

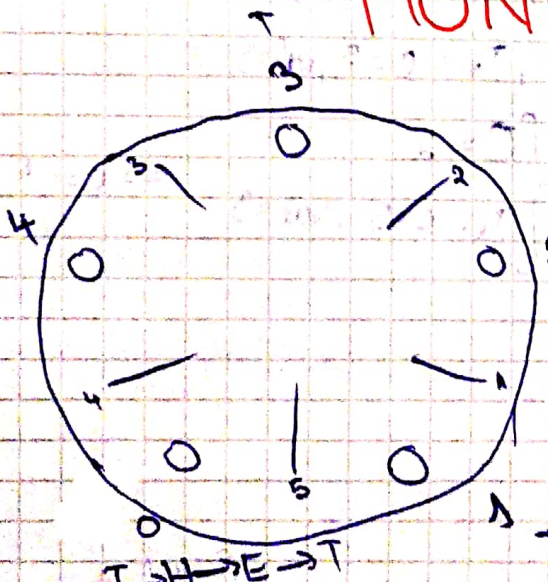


SJF (Shortest Job First)



MONITOARE

10.12.2019.



T→H→ wait(1) → E → signal(1)! executat de filosoful 0.

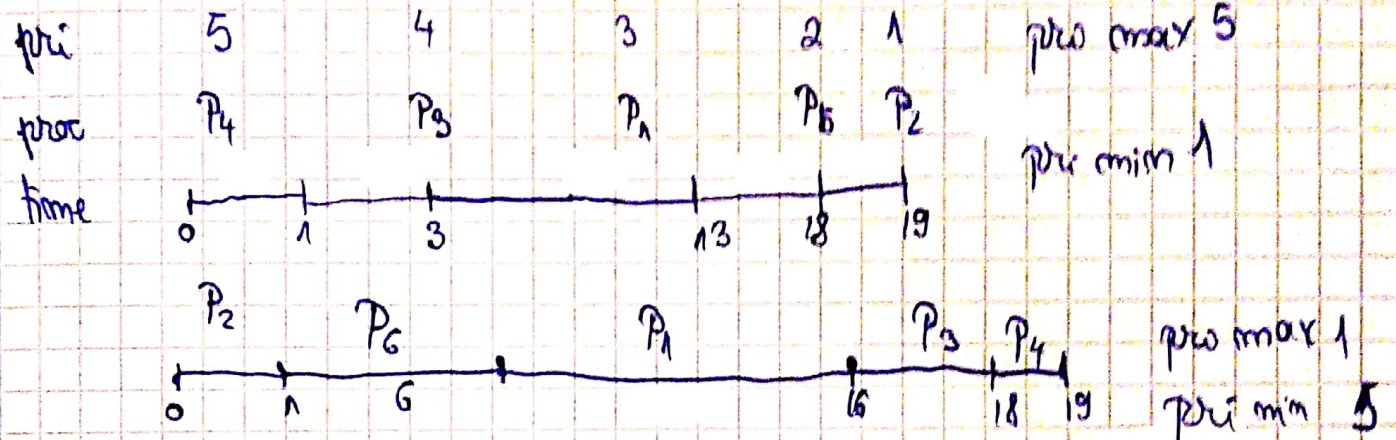
PROGRAMARE PE CPU A PROCESELOR.

PRIORITĂȚI

P_1 at, $P_1.t = 1$.

$\alpha = \frac{1}{\sum_{j=1}^{n} j}$; $\alpha = 0,1$

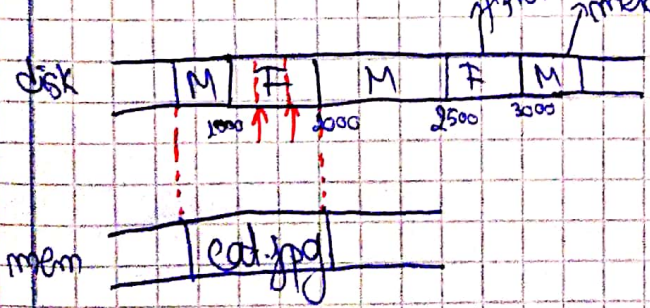
$\alpha = 0,999\ldots9 \longrightarrow \alpha = 0,1$



ROUND ROBIN

SISTEME DE FIȘIERE

17.12.2019.



OPEN FILE LOCKING.

04.10.2019.

linie de comandă

(I) Terminal | nume | param. | redirectoare | → > file (scrie)
>> file (apelează)
| program, trimite mtr-un alt program
tipul redirectoare

ex.:

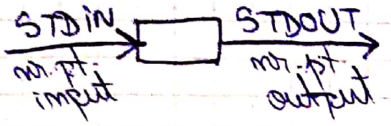
echo hello > hello.txt
(nume prog.) (param) (unde vreau)

=> scrie hello în hello.txt

pwd..
ls
echo (afișează)
copiază → cp <src> <dst> (sursă destinație)
mkdir <path> -p.
ls <path>

ex.: prog 1 | prog 2 | prog 3 > file
(STDIN, STDIN, STDOUT, STDOUT)

STDIN este output pt. prog 1 și input pt. prog 2



ex.: ls / | cat > hello.txt
(/ = root, path)

cat = afișează conținutul unui fișier, dar afișează outputul lui ls și îl pune în hello.txt.

a) ls ('/')

b) cat (STDOUT)

c) hello.txt.

(II) $ strace ./hello ← ktrace hello
(. = director curent; ../ caută?)

program | hello | cod / text
hello → RAM
→ ELF (linux) - format executabil (de ex. pdf.)

proces [ld. loader] ; folosim 1 pt. STDOUT.

gdb => debugger
gdb ./hello => pune prog. din hello în gdb
b main => break
r => run
n => next.

help → • man & <nume>

edităm → • nano hello.c
↳ Ctrl-X + Enter - Y

compilăm → • gcc hello.c -o hello

rulăm → • ./hello

depanăm → • gdb ./hello

0-input
1-output
2-error

1. magic
stat(bar)
2. open(foo) - read
3. read(foo) → buf
4. close(foo)
5. open(bar) → write
6. write(bar)
7. close(bar)
8. success
return ERR
<errno.h> ⇒ 0 → OK
<0 → err.

src - destinație.
foo bar → write.
↳ read

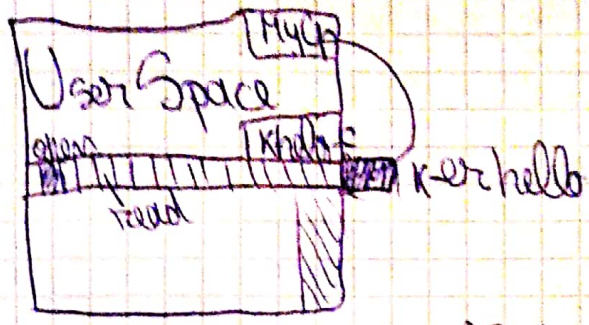
<0 - err
0 - DONE
>0 - return nr. de bytes

char buf[SIZE];

TEMĂ: DE TERMINAT

Var. Complexă:
stat(bar)
2. open(foo)
3. open(bar)
4. Transfer
read(foo)
write(bar),
5. close(foo, bar)

18.10.2019



sys-generic.c

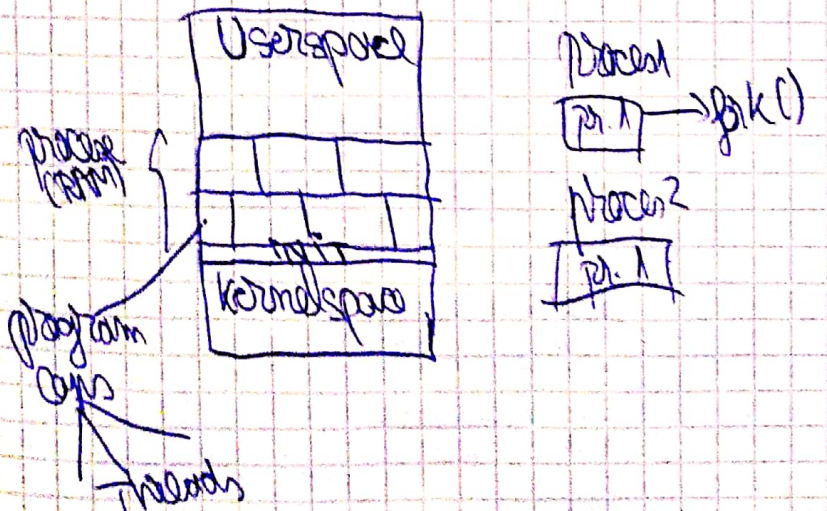
1) Build Kernel
   ↳ g mm.

target: amd64

2) Implement

todd - syscalls.master
+ declaration - syscallargs.h → struct / decl
+ impl. - sys_generic.c
+ naming - syscalls.c.

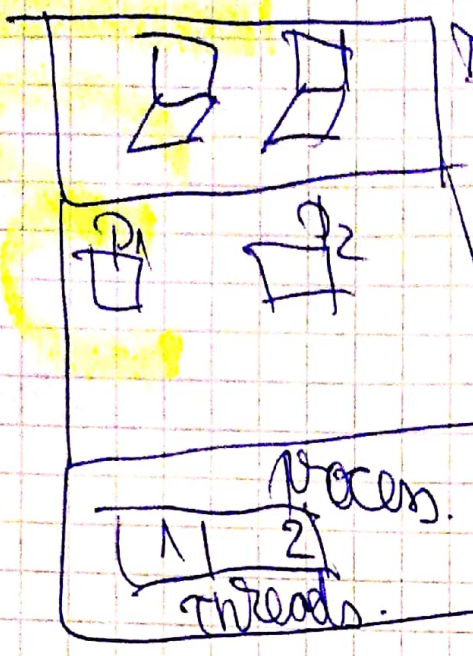
25.10.2019



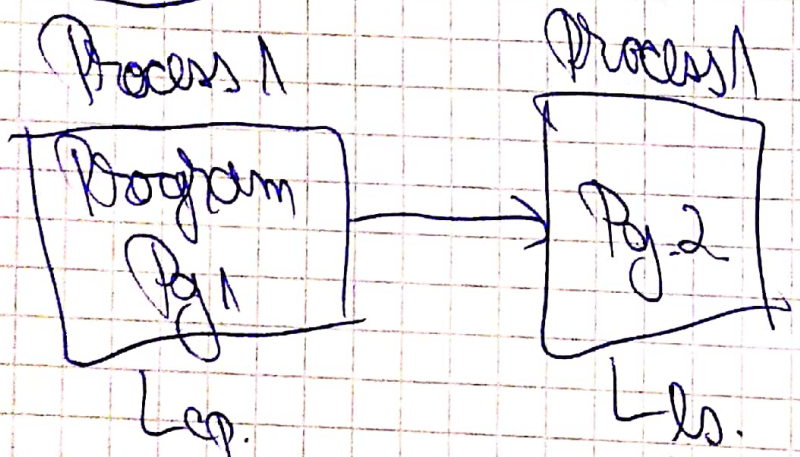
concurență = 2 procese lucrează să scrie aceleași fișiere.

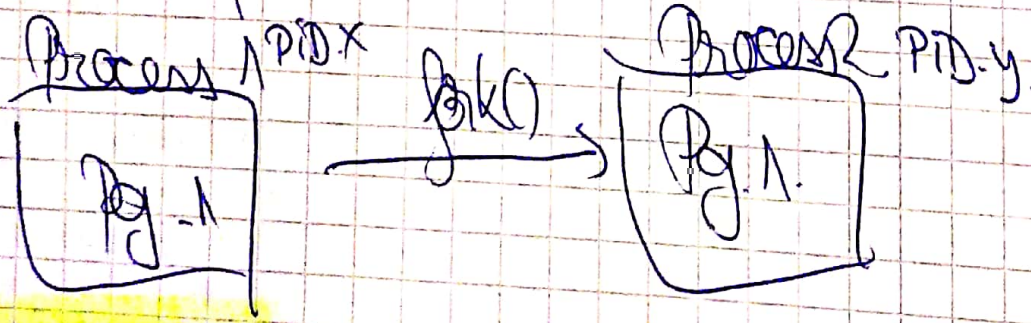
T1: open cp a->b a->c.
T2: open.

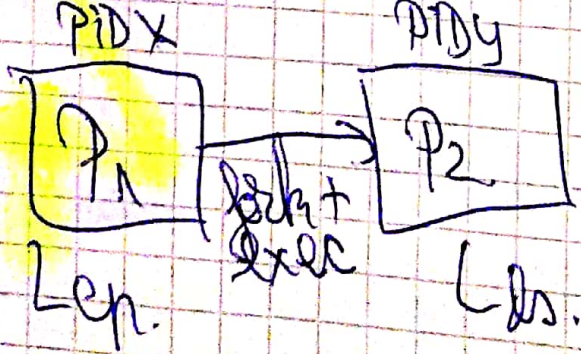
C -> b/fork=>a -> c.



execve



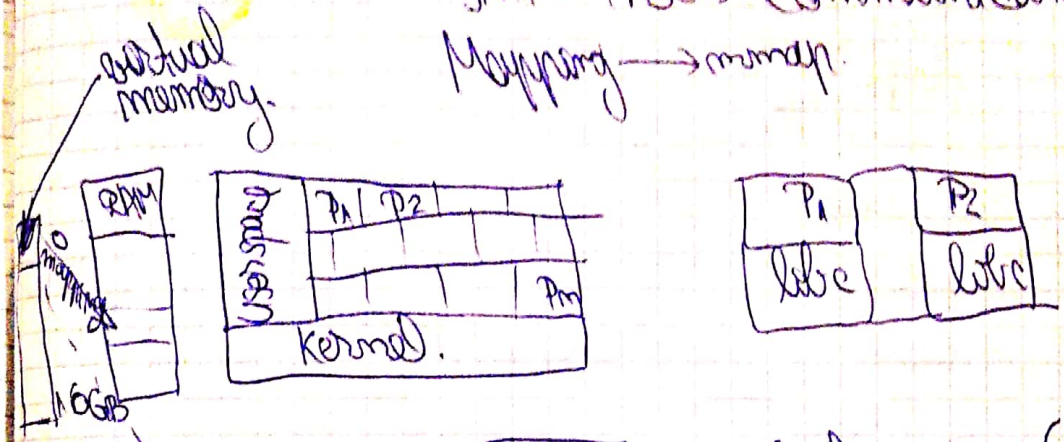


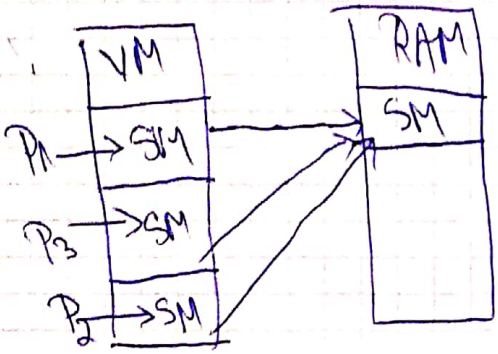


01.10.2019

Virtual Memory
Shared Memory
Inter Proces Comunication (IPC).

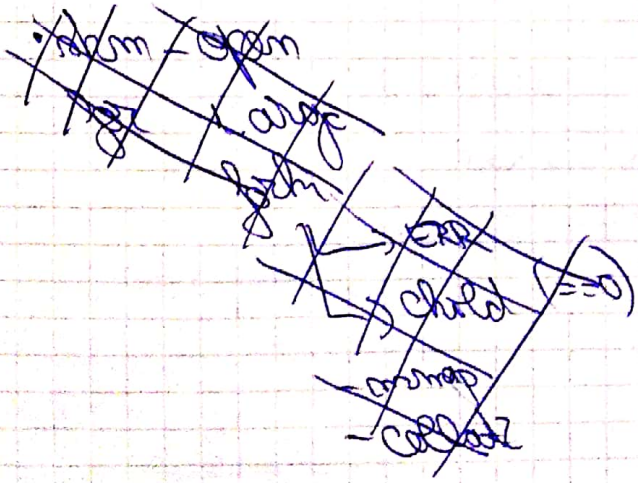
Mapping → mmap.





- shm_open (name). = file descriptor.
- mapping (VM → RAM) → fd
  - → mmap (RAM → VM).
  - → address

funcții: 1) callback (buffer, N).
2) print_callback (buffer).



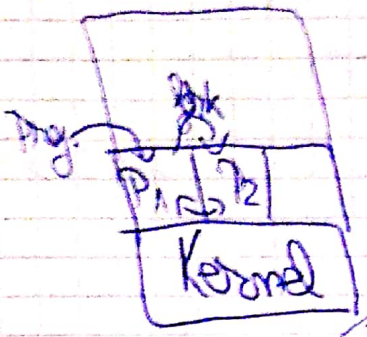
- shm_open
- fwrite
- for (1 → argc)
  - fork
  - if (fork error → EXIT).
  - if fork == 0 → child proces
    - mmap - buffer
    - callback (buffer)
    - EXIT

child proces

for (1 → argc) wait(0);
mmap → buffer
print_callback (buffer)
EXIT.

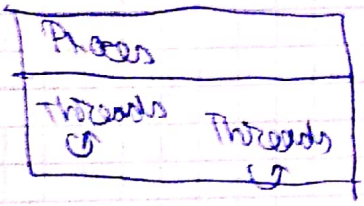
în loc de gcc ipc -o pr
folosc p -lrt

08.11.2019



Concurrency: - distributed

- process $P_1 | P_2 \dots | P_m$ (IPC) - inter process communication
- threads - multi-threading. (MT)



- pthread_open → new thread
- pthread_join → void (...)
  wait pid (...)

thread_t

P_1: 1) string in argv[]
2) create thread
3) thread fct. → inverse string
4) wait thread → join
5) print inverted string

P_2: Hpword()
2) init_matrix
3) thread_func → arg → i,j