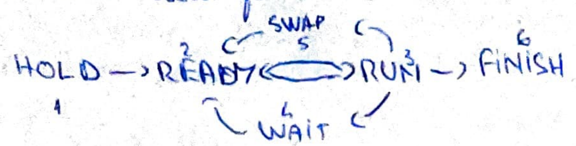
Starile unui proces:



1. waits to be entered in system
2. loaded in mem. wait for CPU
3. run the process
4. wait for resources
5. when mem. is full
the proc. is moved on HDD
6. the proc. is erased from mem.

Planificare:

1. First come first served
2. Shortest job first
   - we have to know the duration
   - not good for big clients
3. Priorities
   - starvation risk
4. Deadline scheduling
   - cel care trebuie să termine mai rapid.
5. Round Robin
   - eq. time periods
   - circular order
   - all in Ready 1 run

Tehnici de alocare:

REAL

1. Mono-tasking - one task at a time
2. Multi-tasking:

Fixed partitions:

a) absolute
+ more proc., but limited nr.
- limited dimension of a proc.
- compiles the addr. of each pos.

b) relocatable
+ no longer fixed RAM mem.
- addresses are stored as an offset
- new calcul. for each instr.

Phys. addr = offset + part_start

- lim nr. of proc.

Variable partitions:

+ no longer a predefined nr of part.
+ we can run any nr. of proc.
- memory fragmentation

VIRTUAL

Paged

+ mem. frag solved
- use a page for a very small proc.
- more difficult calculus for addr. (pag. offset)

Deadlocks - processes/threads entires on ∞ state bc. it wants a resource that is held by another

Causes: 1. Mutual exclusion
2. Hold and wait - block a resource while waiting
3. Non-preemption - resources can't be stolen
4. Circular wait

Prevent: Pick an order for resources and always work/lock in that order

Detect: using the graph of processes
cycle = deadlock

Loading methods:

V1: all prog. pages into RAM from the beg
+ execution is fast
- slow startup
- occupy data never used

V2: load 1st page, rest when needed
+ don't waste memory
- slow startup
- slow exec.

V3: Locality principle - a proc. is likely to need soon the pages next to the page just loaded
- prefetch neigh. pages

How to choose pages for SWAP?

- NRU - page has 2 bits r/w $(0\overset{0}{,}0)(0\overset{1}{,}1)(1\overset{2}{,}0)(1\overset{3}{,}1)$ - se elimină mai întâi clasa 0, dacă nu există, clasa 1...
- FIFO - oldest page
- LRU - N*N matrix - delete the page with least 1 on line

UNIX DISK

Block 0 - boot - small program
Block 1 - superblock - disk details
Block 2
Block n - blocks for inodes
Block n+1
Block n+m - blocks for data

Block - smallest size of data transfered

Segmented

+ more mem. than address size allowed
+ protect mem. access
+ reusable code
- mem. frag.

Paged-Segmented

- segmente împărțite în mai multe pagini
(segm, page, offset)
- procesul are o tabelă de segm.
- fiec. tabelă de segm. are o tab. de pag.

Threads vs Proc.

+ memory don't copy
one stack for every T
+ quicker to create
+ The communication is better (shared mem.)
+ every T execute a function
- no security between T
- if one T blocks, all T block

Zombie = proc that has completed exec. but still has an entry in the proc. table
Use wait to avoid or before creating signal (SIGCHLD, SIG_IGN)

Orphan = still exec. parent died

File types: regular, directory, hard/soft links, sockets, FIFO, periferic caract./block

Symbolic links: ln -s target
- can reference on another disk
- delete file => points to a file that doesn't exist

Hard link: ln target
- new inode for the same file
- have to be root
- file is deleted only when there are no more links to it.
=> we still have access to the data through any hard link.

How to handle malloc?

2 linked lists < free / occupied

- First-fit + fast (first free space) - fragmentation
- Best-fit + economisește zonele mari de memorie (best space) - slower - creates small spaces of free mem => fragm.
- Worst-fit + leaves large free spaces (biggest space) - slower
- Buddy-fit + speed allocate a chunk of the smallest pow of 2, greater than the req. size; keeps list of free chunks by pow of 2 $2^n = 2^k + 2^k + 2^{k+1} + \dots + 2^{n-1}$

PIPE: + easy to work with
! Trebuie moștenit

```
int p[2]; pipe(p);
r/w(p[0/1], &val, sizeof(1))
```

FIFO: poate fi folosit de oricine dacă se știe calea și are permisiuni

```
mkfifo("fifoa2b", 0666)
 int fa2b;
 fa2b = open("fifoa2b", O_RD/WRONLY)
 w/r(fa2b, &val, sizeof(1))
```

No proc. to read/write: wait until a proc. appears

Mutex:

```
pthread_mutex_t mtx;
pthread_mutex_lock(&mtx);
pthread_mutex_unlock(&mtx);

pthread_t t[ ];
pthread_create(&t[ ], NULL, func, NULL);
pthread_join(t[ ], NULL);
pthread_mutex_destroy(&mtx);
```

Semaphore:

```
sem_t name;
sem_init(&name, 0, val);
sem_wait(&name);
sem_post(&name);
sem_destroy(&name);
```

Binary sem.: has values 0/1, it works like a mutex and it is used to sync concurrent processes

inode - entry point to the list of blocks of a file

1-10 primele blocuri de câte 512 bytes

11 - indirectare simplă (următoarele 128 de blocuri a câte 512 bytes)

12 - indirectare dublă

13 - indirectare triplă

$$\left(S + \frac{B}{A} + \frac{B}{A^2} + \frac{B}{A^3}\right) \cdot B$$

B - size of block

A - size of address

Memory hierarchy:
- registers
- cache - temporary mem L1, L2, L3
- RAM
- SSD/HDD

L1: smaller, closer to the comp. unit, faster

L2: slower, bigger

L3: typically, off the chip

where should we place phys page K:

- Direct cache org.:
  page K -> K % N
  + fast
  - cache collisions (cache Trashing) => slow
- Set organising
  first free pos., we have to iterate => slow
- Set associative
  - organised in groups of pages
  - calculated as page_ad % cache_set
  + flexibility
  + avoid collisions
  - doesn't use all available cache lines

AWK: NR - current line
NF - nr. of fields
$0 - entire input line
$1, $2, $3 fields in curr. line

```
awk -F: '{ print $1 }' fișier
```

```
cut -d' ' -f1
```

Processes:

```
ps -ef | awk '{print $1}' | sort | uniq -c | sort -n -r
```

wc: -l linii -w cuvinte
 -c caractere

```
grep -E "." fișier
```

-c nr linii
-n pune nr linii în față
-o only part that matches

sort -r reverse
 -u elim duplicate

Read la FIFO: așteaptă până nu mai există proces deschis pt scriere sau până apar date; întoarce câte date a citit sau 0 dacă ∄ proces care să scrie

SED

a) search and substitute

```
sed -E "s/([ ])([ ])/\2\1/gi" fiș
           1    2
```

g - global
i - case insensitive

b) transliterate

```
sed -E "y/aeiou/AEIOU" fiș
```

c) delete

```
sed -E "/.../d" fiș
```

```
int dup(int vechi)
int dup2(int vechi, int nou)
dup2(p[0], 0);
```