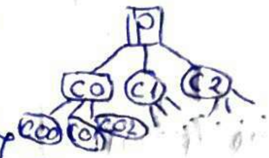
fork() => -1 err, 0 fiu, >0 părinte
for 1..n : fork() => $2^n - 1$ fii
fork() face copie!
Părintele așteaptă ca (wait) până când copilul moare (exit) ca să nu se transforme în zombi. killall(); kill(pid, semnal)

for 1..3
fork()
for 1..3
fork()
exit()
for 1..3
wait()
exit()
for 1...3
wait()



v - array
l - list (arg in exec)
p - Search for path
fără p => bin/grep

signal (SIG, f)
SIGINT = CTRL + C

exec[l/v][p]
└> oprește procesul curent ca să pornească un program (să fie exec în copil)

char* a[] = {"grep", "ceva path", "alt path", NULL};
execvp("grep", a);

Pipe -> 0 read, 1 write
flux de date; dacă e gol așteaptă până se trimit date, dacă e plin -//- șterg date
read -> șterge din pipe, write -> adaugă
} => close pipe fast!

orfan: nu s-a terminat, dar părintele da

mkfifo nume
- fifo sunt fișiere pe disc, pipe în RAM
- Toate procesele care comunică prin pipe trebe să fie descendenți ai proc. creator, fifo nu

---

0 - stdin
1 - stdout
2 - stderr
chmod u/g/o
$@ listă arg
$^ string arg
$# nr arg
$? last cmd exit code
NR - nr line
NF - nr fields
< input
<< input append
> output
>> output append
| pipe
2> error
find [nume] -type f -name "*.c"
find ... | while read F
... > a.txt 2>&1 (redirect output și erori în a.txt)

popen("ceva cmd shell", "r"); fp -> fp, fscanf(fp, "%d %s", &n, u); close(fp); printf("%s, %d", u, n);

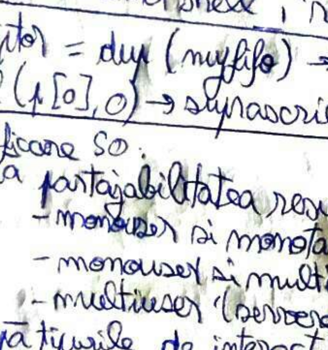
---

descriptor = dup (myfifo) -> copiază
dup2 (p[0], 0) -> suprascrie 0

Semafor: {v(s) - valoare, c(s) - coada de așteptare}
P(s) - wait
V(s) - post

Clasificare SO:
- după partajabilitatea resurselor
  - monouser și monotasking (vechile mobile)
  - monouser și multitasking (windows, android)
  - multiuser (servere)
- după tipurile de interact. permise:
  - seriale: execută un program ce nu poate fi gestionat (?)
  - cu timp-partajat: se poate interveni în execuția programului (cuante de timp)
  - calc. personale și stații de lucru: se partajează resurse între procese
  - sist. autonome: dedicate proceselor industriale (roboți, sateliți, supraveghere)
  - sist. portabile destinate comunicațiilor
  - sist. conectate în rețea: conectate la internet
- după org. internă a prog. ce compun SO:
  - monolitice
  - cu nucleu de interfață hardware
  - cu struct. stratificată
  - org. ca mașini virtuale

Stări proces:



1. Conversie de intrare [SPOOLING]
2. Alocare mem; creare proces; aloc. periferice
3. Alocare procesor la proces
4. Eliberare procesor
5. Lansare I/O + 4
6. Rezolvare întrerupere terminare I/O
7. Nu mai e loc în mem.; ocupare spațiu disc + 4
8. Aloc. mem; reîncărcare; eliberare disc
9. Elib. mem, periferice, procesor; ștergere proces
10. Conversie de ieșire [SPOOLING]

Fazele program:
- editare text editor
- compilare (.c => .obj)
- linkeditare (.obj + .obj => .exe)
- testare + depanare
- execuție

Lansare SO:
APASA START:
for i = 1...100:
read(M[i])
transferă la (M[0])

Program pe hard; proces = program în exec (pe RAM)
Existența unui proces e condiționată de: procedură (ce), procesor (cine), mem. (unde)
Sect. critică = sect. ce nu poate fi accesată simultan de mai multe procese (resursă critică)
Peterson sol. pt. protejarea sect. critice (spinlock)
Problema prod. și a consumatorului: vo (plin) = 0, vo (gol) = n, v (exclus) = 1
Deadlock (impas):
- detectare: ciclu în graful cu resurse și procese
- evitare: se blochează resursele în aceeași ordine (prob. filozofilor)
- ieșire: se omoară un proces, repune sistem, revenire la o stare anterioară (posibil livelock)

Multiprogramare: un singur core cu mai multe procese.

Mecanisme de translatare a memoriei:
OP (1) —compilare→ AM (2) —linkeditare→ AR (3) —executare→ AF (4)
1. spațiul de nume al obiectelor din sursă
2. adresa relativă
3. Adresa relocabilă
4. adresa fizică din mem. operativă

Gestiunea memoriei (viteza crește, dim. scade)
externă:
- de arhivare: fișiere etc (HDD/SSD)
- secundară: swap, mecanisme de mem. virtuală
internă:
- operativă: RAM, conține procese
- cache: între RAM și disc
CPU

Tehnici de alocare
1 reală (spațiu contiguu):
- monotasking: un task ocupă toată memoria
- multitasking:
+ cu partiții fixe: se împarte memoria în partiții. Un proces are o partiție până moare
- absolută: linkeditorul pregătește programe pt. a fi rulate într-o zonă prestabilită
- relocabilă: calcul de adresă (bază + offset)
- cu partiții variabile: fragmentare, trb. concatenate spații libere vecine; rămân zone mici și multă de mem. neutilizată

2 virtuală:
1- paginată: fiecare proces este împărțit în pagini și are o tabelă de pagini; paginile sunt distribuite necontiguu, toate cu aceeași dimensiune. Pagini fizice (din RAM), pagini virtuale (pagini în care este împărțit procesul) (rezolvă fragmentarea)
- segmentată: fiecare proces este împărțit în segmente (+ tab. de segm.). Segm. au dim. diferite și pot fi folosite în comun de mai multe procese. Refolosire memorie care are același conținut + protecție împotriva accesului unui proc într-altul
- S și P => proc => segmente => pagini

Metode de plasare:
- metoda primei potriviri (First Fit)
- metoda celei mai bune potriviri (Best Fit)
- metoda celei mai rele potriviri (Worst Fit) - mai greu de fragmentat
- metoda alocării cu camarazi (Buddy System) (se caută cele mai apropiate puteri ale lui 2)
$A = B + 2^k$ și $B \% 2^{k+1} = 0$

Metode de înlocuire:
① NRU (not recently used) → 0 nereferit nemodif, 1 neref. modif, 2 ref. nemodif, 3 referit modif. (2 biți alocați pt. fiecare)
② FIFO (se poate combina cu NRU)
③ LRU → matrice cu 0; accesare pg. => 1 pe linie și 0 pe coloană

cache proiectare spațiu:
- directă: S=A%C, S - nr. slot, A - adresă, C - nr. sloturi (dezavantaj: dacă două blocuri ocupă același slot va fi trafic mare între cele două memorii
- asociativă: plasează un bloc într-un slot oarecare
- set-asociativă: mem e împărțită în I seturi fiecare fiind compus din J sloturi

Legătură hard: inod către acel fișier
Legătură simbolică: pointer spre inod
Fișierele au contor de legătură care este incrementat la fiecare nouă legătură. Fișierul se șterge când e 0.
Inod (descriptor): pointer către blocuri de fișiere, accesare directă, indirectare simplă, dublă, triplă
Fișierul conține nume și inod.
Procese → tab. de fișiere deschise în sis → tab. de inods → fișier

FAT (File Allocation Table): tabel cu adrese de început a fișierelor; se face o copie a tabelului; fișierele sunt un linkedlist de clusteri (grupuri). Ultimul cluster => 0xFFF8 / 0xFFFF. Permite undo la ștergere. Se pot pierde date dacă se ia curentul.
NTFS (Windows NT file system): Fișiere mici în small file record. Fișiere mari în large file record. Reprezentare sub formă de tree spre clustere. Directoarele sunt reprez. sub formă de b-tree spre fișiere.

Regex: . caracter; \(\) grup; \1 nr. grup; ^ început de linie; $ sf. de linie; \< ... \> frontiere; ? cel mult o dată; * 0 sau mai multe ori; + 1 sau mai multe ori; {n, m} min n, max m.
Grep: -c count; -A after; -B before; -n line number; -m Num se oprește după Num; -f file; -e pattern; -o only matching.
Sed: -n suprimă afișarea; m, n de la linia m la n; m, ~step de la m din step în step; /g global; /p print; /d delete; s/regex/regex substitute; i insert; c change; a append; q quit
sed 's@.*/@@'; sed '2,4!d'; =' file.txt | sed 'N; s/\n/ /'
AWK: awk 'BEGIN{total=0}{total+=NF}END{print total/NR}'; awk '/pattern/{action}';
-F; -F'[: \t]'; x ~/regex/; x !~ /regex/
cera = $((cera + nr))
for arg in $@   for F in `find -type f -name "*.c"`   let cera = cera + nr