110 std m(0), c ; std acf (1) >,>> , std w(2) <?, 2 >> (((#1)$0 >, $d >>) Tarcas Nichan - Cristian 911/1

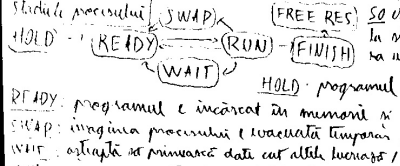
sytax | \' schimbă separat, [a-z] - a-z, [^0-9] - ^ nu e cifră, ^ inceput, $ sfârșit, \< cuv >\ ;
or sau, + sau, * max ori ; (m,m) - min m(m,), max m(,m), (m) m ori, | "sau" logic
AFD | afișă linii din ieșire, -E - nu reg. exp, -v invers, -c. câte inmatchuri, -c. count, -n numerotare, -x linii, -o only matching
1 | sed -'s/old/text/ input text - replace old text with new text in file input. txt
2 | limbaj de prog interpretat (≈C); cmd auto [OPT] '{ prog./instr }' [FILE], -F'f': set delim., -f source file, if, then, while
-f: for/exp. card. exp) { instr }, for (var in arr) { instr }, $0 - curr line, $1, $2 ... fields of $0; NR: nr ordine al $0, NF nr
fields of $0: BEGIN { } SEND { }, -v: var. externă, e.g. auto '{print NR, $0, NF}' input.txt - display line nr, line and last field
-FILE H!/bin/bash, interp. - caminarda from caminarda line; string, array, bas '\0' at end, char array no, cast la c
-mul fișier dintr-o ar program, /dev/NULL = NULL (windows), 0 - True, 1 - False - False, for var in. do. done, while/
until do done, M = 'cmd' (= in var the output of cmd, exit code ≠ output; ghilimele - caract special interpretate.
Tre apostroafe: se ex asa cum este, if test "cmd" (= if [ cmd ] (= if [[ cmd ]] then. elif. then else fi (mai test)
$0 - numele fișier, $1, $2, $3 argumente din com., $# nr arg, $@ lista arg, $* nr arg, $!: last exit code, $$: PID, $? - last cmd
length s: ${#s}, index s,c: prima apariție a c în s; substr s,p,l: substring in s from pos p of len l, script: file w/
exec. rights that contains shell/bash commands; $(( exp )) = math exp, $[ exp ] = 'exp' $(cmd): command substitution.
permisiuni (- normal, d - directory, l - pipe/link, s - socket) + rwx u-x a-r = chmod u+x g-r test. sh, man bash
first ones in prim. strings"

Commands | apropos, whatis, man, man -k manual-related; find, wc, cut etc. ; | pipe between commands on line

Procese | Un proces este un program în execuție. Procesele sunt concurente dacă rulează simultan. Race conditions reprezintă situații în care mai multe procese concurează pentru accesul resurselor, iar rezultatul nu este mereu cel așteptat. Procesul zombie este un proces fiu al cărui exit code nu a fost colectat, apare în lista de procese, dar a fost terminat (consumă resurse). Semafor: wait(0), signal(SIG_CHLD, SIG_IGN), SIG_CHLD trimite părintelui când copilul termină execuția. Procesul orfan este cel care încă rulează, dar părintele a terminat, acest părinte va fi INIT. Fork: mecanismul de creare a unui proces (existența diversiunii celular). Exec system calls - folosite pentru a suna o altă com. pe locul procesului curent. Semafor. fork() și exec in copil/părinte. Fork creează copii ale date. Exec. copilul modifică independent de copil și invers. Semafor. Mecanisme de comunicare (buffer in memorie, date dischis ca un fișier, de două ori, un capăt pentru read și unul pt write (r = 0, w = 1). read: preia și șterge date din PIPE, dacă e gol, așteaptă date sau să se mai fie nicium scriitor. Este cât poate și returnează ce și cât a citit. Write: adaugă date în PIPE, dacă este plin, așteaptă. Se face loc sau să se mai citească și returnează cât a scris. Acestea se pot bloca, dacă PIPE se închide cât mai rapid, copilul moștenește ce e înainte de fork(), și adaugă trimite date. Deci pe PIPE un FIFO. buffer în memorie vs. fișier de disc. Crearea PIPE il deschide implicit FIFO implicit. PIPE se distruge la închiderea capetelor, FIFO explicit. FIFO poate crea problema dacă mai multe procese cineva pe el. PIPE permite comunicarea între procese, chiar dacă nu au aceleași părinți / nu sunt părinte - fiu. FIFO asemănă cu PIPE, dar se poate deschide. poate cauza deadlock. Shared memory: zonă de memorie care nu aparține niciunui proces, dar care poate fi accesată de orice proces ca memorie proprie.

Threads | mecanisme de a implementa programe concurente cu partea creation, consum de memorie redusă, comunicare simplificată. pthread_create: execută funcția pthread_join: așteaptă terminarea threadului, hiccup. Fiecare propriu Resurse critice: resurse care intră în race conditions. Zone/section critice: porțiunea din thread care nu poate accesa resurse folosite de alte thread. Variabile condiționale: mecanism de sincronizare. Semafor: wait, post, init, destroy. Folosim thread-uri este core-uri avem, fiecare thread ca un core. Deadlock: apare în timpul morții. Deadlock: Apare când un thread/proces intră într o stare infinită de asteptare, deoarece se afla într-un ciclu de procese ce asteptă unul după altul. Cauze: Mutual exclusion (resursele nu se pot folosi în paralel), Hold & Wait (se blochează o resursă și asteptă o altă), Non-preemption (resursele nu se pot lua de la altii), Circular wait. Soluții: Silent: se ignoră un proces restrictiv, crearea unui graf de procese și resurse → detecție - avoid → reinvoit deadlock. Soluții: [illegible]

Starile procesului




SO este conceput so rulese programe / so facilitese accesul la sistemul unuia / mai multor utilizatori si de a asigura utilizarea eficienta a resurselor de calcul.

HOLD: programul se afla intr-o coada si asteapta sa fie planificat;

READY: programul e incarcat in memorie si poate fi "servit" (attended to) de catre procesor;

SWAP: imaginea procesului e evacuata temporar pe disc cand vreau so facem loc altui proces;

WAIT: asteapta so primeasca date cat altele lucreaza / asteapta terminarea unei operatii I/O, deci loc loc cât asteapta

NW: un proces executa efectiv instructiuni (majorita) ale procesului; FINISH: procesul a-a terminat, le lucrate resursele

Planificarea proceselor (scheduling): (1) FCFS (first come, first served), (2) SJF (shortest job first): le stiute durata, durata mica; (3) Priority: e cele mai prioritare primesc mai multe proc; Rotation (Round robin); (4) Deadline: e intru cel care le so termine cel mai rapid (A/2/10, B/1/15, C/1/11 ⇒ B,A,C) (5) Reward - Rate (time sharing): Toate proc. indeplinesc deadline, maj. sunt in READY

Sistemele memoriei

- real: (legendar)
  - mono - tasking sys. (1)
  - multi - tasking sys.
    - fixed partitions - absolute (2)
    - relocat. (3)
    - variable part. (4)
- virtual
  - paged (5)
  - segmented (6)
  - paged - segm. (7)

(1) un singur program ste incarcat daca vreau so dezvoltam altul; (2) (4) mai multe procese, (-) se ocuteaza e limitat de m. part., prog. rulează intr-o parte prestabilita; (3) apare calculul de adrese, adresele sunt alocate ca offset intr-o partitie, la compilare nu stim unde va rula prog. (+) la rentrare trebuie facut un calcul de adrese (AE = offset in exe, AF = inceput partitie = AE), nu putem rula mai multe programe deci avem partitii; (6)(+) memoria se fragmentează (spatiu gol, trebuie compactata), (+) adresa din executabil va fi calculata când lansam procesul, calcul simplu; (5) memoria RAM este impartita in n pagini fizice iar programul are o tabela de pagini virtuale (+) fragmentarea dispare, (-) calcul de adresa complicat (le stiut pt. fiecare prog. o tabela de pagini proiectii sa aceze, calcul adrese); tabela segmente; (7) grupez o codul si datele cu o semnificatie este controlat de SO, (+) protectie a datelor la acces, (+) nu se aloca fragmente, AE - virtuala (seg. offset); accesul la fiecare segment e impartit in pagini, (-) calcul adrese se mai complicat (seg. pag. offset)

(2) fiecare proces actiu are tabela segmente, fiecare segment tabela de pagini

Calcul de adrese: a.c → a.o → biv (executabil); intr-un compilator loc in fisier (obiect) link, editare → adresa din exe → adr. fizica

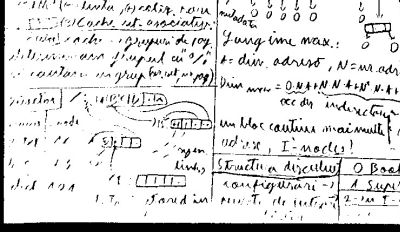
Politici de incarcare (load): (1) incarcam toate prog. de la inceput, (-) pornire lenta, incarcam pagini a nu se vrea pasa, (2) accesu rapid la rentrare, (3) incarcam paginile când e necesar, (-) pulan mai lenta, (4) pornire mai rapida, pagini nefolosite nu ajung in memorie; (3) Principiul vecinatatii: daca un program sfârsi o pagina, e probabil va cerri si o pagina vecina → prefetch (aduncam urmatoarea pagina ceruta si cativa pagini invecinate)

Politici de inlocuire (unload): Cum aligem o pagina pt a o muta in SWAP? (1) FIFO, (2) NRU (Not Recently Used): fiecare pagina este data cu o mica durata de viata, (3) LRU (Least - ly): 2 pag in RAM ⇒ NRU bits matrice, (and se accesa) pagina x → linia x = 1, col x = 0 ⇒ aligem victima pagina cu cea mai mica sumă a liniei (= 0)

Politici de plasare: Cum tinem contabilitatea spatiilor alocate / libere in mem? Unde plasam o cerere de alocare memorie

First fit (1) rapid, (-) mi se fragmentarea, (2) Best fit (1) economisesti zone de memorie mai si (-) mai lent, fragmentare foarte mici, (3) Worst fit (1) lasi spatii mari, aproape nu apare fragmentare, (-) mai lent, (4) Buddy fit (1) cream mereu cea mai mica putere a lui 2 mai mare ca spatiul cerut, intretinem liste ale spatiilor goale, (2) alocarea, alocam in cea mai mare zona potrivita si restul memoriei o distribuim in restul

Cache: (1) Cache direct. map: memoria RAM 104, x%16; (2) Cache set - associative: memoria ram x%sc; (3) Cache set - associative: grupuri de pag. distincte sunt grupate ci %; (4) cautam in grup fara set mapping



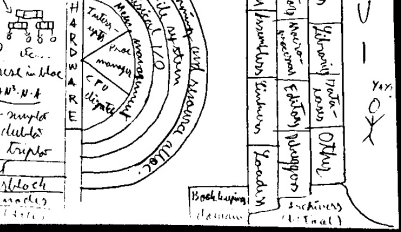
Sistemele de fisiere (UNIX; exe, x.d, w.f): Windows: FAT, NTFS (stiteman)

Structura unei partiunea de permanent




Jung ime max.: t = dim. adresă, N = nr adrese in bloc

Dim max = $\frac{0.N+N\cdot N+N^2\cdot N+N^3\cdot N}{\text{dec dy}}$ · t index cu grupuri

un bloc contine maximum 1024 x t, I - nodes!

Structura a dispozitivului: O Boot, 1 Superblock, 2 - n I - nodes