Mentimeter

Instructions



Mentimeter

Funksjonskall vs systemkall

Systemkallet read() forsøker å lese count bytes inn i bufferen pekt på av buf fra filen tilknyttet den åpne filendeskriptoren fd. Ved suksess returnerer den antall leste bytes. Ved feil returnerer den negativ én.

ssize_t read(int fd, void *buf, size_t count);

Som nesten alle systemkall i Linux, aktiveres read() gjennom en bibliotekfunksjon med samme navn. Det finnes også andre bibliotekfunksjoner som er avhengige av read(). Hvis dette virker forvirrende, les man-sidene man 2 read (systemkall) og man 3 read (biblioteksfunksjon).

ssize_t read(int fildes, void *buf, size_t nbyte);

Mentimeter

Funksjonskall vs systemkall

Systemkall fungerer som inngangspunkter til kjernen, hvor selve «tungløftingen» av for eksempel lesing av bytes fra en fil lagret på disken inn i RAM utføres. Kjernefunksjonen som tilsvarer read() er sys_read().

Uansett hvordan det kan se ut på overflaten fungerer funksjonskall og systemkall forskjellig internt. Systemkall utsteder TRAP-instruksjonen for å gå inn i kjernen, mens funksjonskall utsteder CALL-instruksjonen for å kalle en annen funksjon.

På hvilke to måter er oppførselen til disse instruksjonene forskjellig?

I'm cooked

Same

ene leser og andre rdigerer

Sys_call går inn i kjernen og krever derfor admin-rettigheter?

Kan du utdype hva du mener med dette spørsmålet? Ble det ikke besvart av de forrige to slides?

User space og kernel space?

Funksjonskall kaller en annen funksjon Systemkall går inn i kjernen

Den ene fungerer kun på brukernivå og den andre fungerer på kjernenivå

Mentimeter

På hvilke to måter er oppførselen til disse instruksjonene forskjellig?

Abstraksjon

Ene kaller på funksjoner i kjernen og andre kaller på en annen funksjon?

Funksjonskallet er dyrere enn systemkallet. Man har mer kontroll med systemkallet. Men grensesnittet til funksjonskallet er mer innspisset og spesielt, mens systemkallet er mer generelt.

Systemkall kalles via API gitt av OS

Sys_call har tilgang til flere ressurser/informasjon

read kjører i brukermodus og kaller på sys read som kjører i kjernemodus

Systemkall er funksjonskall til kjernen

Hvilket kall er dyrere ?

Mentimeter

På hvilke to måter er oppførselen til disse instruksjonene forskjellig?

Når får vi karakter på hjemmeeksamen?

Funker read() uten sys_read()?

Korte oppgaver blir behandlet først og må ikke vente like lenge.

2^20 * 32 bit

Mentimeter

Shortest Job First (SJF)

For ulike typer datamaskiner og forventet arbeidsmengde vil vi bruke skeduleringsalgoritmer med svært forskjellige egenskaper. Vennligst vurder følgende spørsmål.

Mentimeter

Hva er fordelen med en SJF skedulering?

flere oppgaver gjort på kortere tid

Korte oppgaver trenger ikke å vente lenge for at store oppgaver skal bli ferdig

Mindre prosesser må ikke vente på større prosesser.

man får fullført flere oppgaver tidligere

CPU blokkeres ikke av en lang jobb, så viktige små prosesser kan kjøres

Lav gjennomsnittlig ventetid

De raskeste oppgavene blir ferdig først, men ulempen er "starvation"

Fordelen er at små prosesser blir utført først og kan på den måten utføre mange oppgaver på kort tidsintervall

Hva er fordelen med en SJF skedulering?

Kan kan kjøre prosesser som tar kort tid først

Får unnagjort flere oppgaver på kortere tid

Høy throughput

Sjf vil ta korteste jobb først, og hvis det er en viktig oppgave som har kort tid vil den prioritere den først

Ulempe: risikerer at større jobber blir utsatt for lenge

Mindre prosesser utføres først, da de krever mindre tid og ressurser enn store prosesser

kortere oppgaver blir fullført tidligere. Lengre oppgaver må kanskje vente lengre for å bli fullført.

går kjapt, de korte oppgavene trenger ikke vente på de større, fordi de korte blir prioritert

Mentimeter

Hva er fordelen med en SJF skedulering?

Effektivt hvis alle jobbene er korte - mindre ventetid

Jobber med korte CPU-krav krever ofte umiddelbar behandling (f.eks. I/O). SJF Kan dermed gi FLERE utførte jobber, pr. tid.

Man passer på at lange prosesser ikke opptar mye tid der mange mindre prosesser kunne kjørt

Minimalt gjennomsnittlig ventetid

Hva er preemption?

Er den vanskeligere/mer komplisert å implementere kontra fifo?

Hvorfor bør man implementere denne måten istedenfor andre? Hvilken historie har den?

Kan SJF ta hensyn til prioritet hvis prosessene er like små?

Mentimeter

Hva er fordelen med en SJF skedulering?

Er SJF non-preemtive eller preemtive? Hva er forskjellen?

Heap er dynamisk og stack er statisk

I stack: Parametre og variabler Heap: Allokere minne (malloc)

Hvis det er minne som skal refereres til flere ganger på tvers av funksjonskall, legges det på heap, mens for minne som har kort levetid, kun for én scope/funksjonskall, på stack

2^31 pages

Mentimeter

Minne/Layout

Heapen og stakken inneholder begge prosessens kjøretidsdata.
Å velge hvilken som bør brukes når er ikke alltid åpenbart for programmereren.

Mentimeter

Hvordan drar en prosess nytte av disse to minnestrukturer?

Stack: data som brukes innad i enkelte funksjoner, gjerne mindre datastrukturer Heap: større datastrukturer som kan brukes flere ganger i prosessen

Heap er «global» for programmet, mens stack er unik for hvert funksjonskall

malloc(1024) vs char buf[1024]

Lokale/midlertidige verdier lagres i heapen og globale/verdier med lenger levetid og bruksområde skal lagres i stacken

Stack: Parametre og variabler. Blir større jo flere funksjonskall Heap: Allokere minne (malloc)

Heapen brukes til verdier hvis størrelse er ukjent før programmet kjøres(dynamisk allokering). Den er veldig treg i forhold til stacken. Stackens størrelse er mer forutisgbar mht. funksjonskall, vars,

Stack er nyttig hvis man kun vil holde data lokalt i en kontekst (scope), for eksempel i en funksjon.

Stack - brukes for lokale variabler og funksjons kall Heap - for dynamisk minneallonering og for data som skal leve lengre enn funksjonshemmet de ble opprettet i

Mentimeter

Hvordan drar en prosess nytte av disse to minnestrukturer?

heap bør brukes av data som er store eller av uviss størrelse, eller for å bevare data utenfor funksjonskall. stack bør brukes for lokale data av kjent størrelse, hvor scope ikke er viktig

Heap er dynamisk og vil inneholde lokale variabler mens stack vil inneholde variabler som er lagret i programmet

Ting lagret på heap har lengre levetid enn lagret på stack. På stacked må man derimot ikke frigjøre minne manuelt, slik som man må på heap, fordi det frigjøres sammen med funksjonskallet

Stack kan returnere data

Hvorfor tar man ^2?

Circular scan

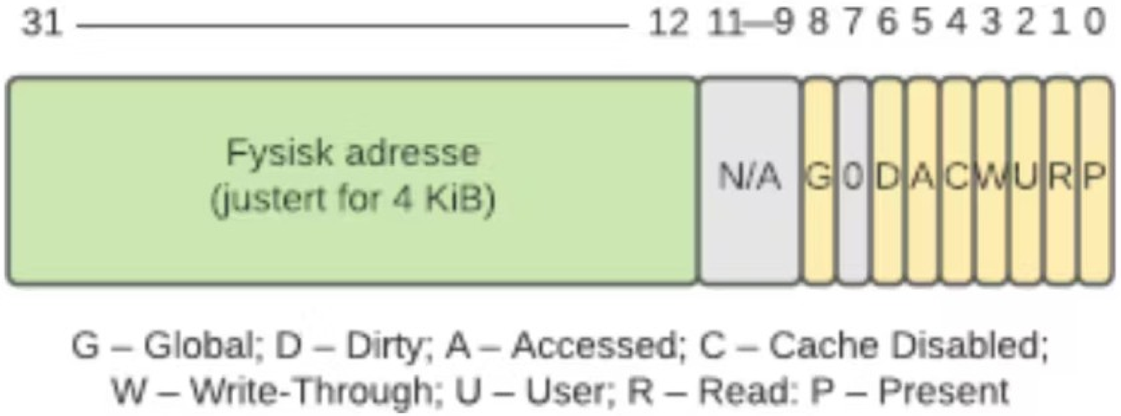
Mentimeter

Memory/Multi-level paging

En sidetabelloppføring på IA-32-arkitekturen er vist i Figur 1.

Gitt at sidestørrelsen er 4 KiB, hvor stor er sidetabellen?

(Husk at antall sider kan utledes fra sidestørrelsen.)



Mentimeter

Hvor stor er sidetabellen?

1048576 pages

2^64

2^64

2**32 / 2**12 = 2**20

Sidetabellen har plass til 20 entries. 2^20 =1 048 576

2^32 = 4294967296

2^32/4096

2^32 / 4096

Mentimeter

Hvor stor er sidetabellen?

Hva er en sidetabell? Hvilken kontekst er det snakk om?

2^32 / 4096 = 1 048 576

2^32 / 4096

4MB

4096 bytes

262 144

((2^32)/(2 ^12)*4=4* (2^20)=4194304 = 4MB

Ulike mekanikk - platters, sector og sylinders

Mentimeter

Hvor stor er sidetabellen?

Når får vi karakter på hjemmeeksamen?

Mentimeter

Memory/Disk Scheduling

Vanligvis gjøres det ulike tiltak for å forbedre ytelsen til mekaniske disker i dag, spesielt når det gjelder tilgangsforsinkelser og dataoverføringer.

Mentimeter

Nevn en tilnærming og forklar den.

SCAN - Les data fra disk fram og tilbake med diskhodet

scan- les data fra d isk fram og tilbak md diskhodt

SCAN, CSCAN, LOOK, CLOOK, CSF, FIFO

Man prøver å redusere antall "hopp" disk-hodet gjør ved å legge minne som henger sammen så nærme som mulig hverandre (..? longshot)

La hodet bevege seg i C-SCAN: la det svøpe over hele disken, og så hoppe tilbake. Dette kan gjøres raskt, og det er relativt «fair».

C-SCAN som forbedrer rettferdigheten og responstiden

CSCAN - sorterer dataen slik at den kan leses på en «tur» bortover, før leseren flyttes helt tilbake til start

Noe med å ta hensyn til fartsendringen til lesehodet, at den bruker tid på å akselerere?

Mentimeter

Nevn en tilnærming og forklar den.

Layers

SCAN: Frem og tilbake, C-SCAN: Lese en vei, full fart andre vei, LOOK: Scan men bare til ytterste aksess. C-LOOK: Som C-SCAN men for LOOK. SSTF: nærmeste blokk. FIFO: vanlig fifo

C-SCAN - Send diskhodet kontinuerlig frem og tilbake over disken, gir laveste gjennomsnittlig lesetid

Flere platters fører til at man kan få tilgang til mye data på relativt kort tid

Skrive på under-og oversiden av diskene, og la hodet lese i begge retninger?

Shortest Seek Time First (SSTF)

C- LOOK - diskhodet leser data i en retning fra disk fra start, ytterste forespørsel fra kant ved start, til ytterste forespørsel på omvendt side. Den flyttes tilbake til start uten å lese data tilbak

Mentimeter

Storage/Disk layout

Kjeding i media og kjeding i kart er to teknikker for å holde oversikt over hvilke blokker som tilhører hvilke filer.

Hvordan kjeding i kart forbedrer effektiviteten for tilfeldig aksess?

🤷‍♂️

Kjeding i blokker krever iterativ gjennomgang gjennom en lang lenket liste. Kjeding i kart lar deg gjøre indekserte oppslag.

Du kan forholde deg til kartet i stedet for å lete i blokken

Fordi man da kan finne f.eks. slutten av en fil ved å slå opp i kartet i stedet for å måtte sjekke alle blokkene filen består av. Litt som at oppslag i indeksert liste er raskere enn lenket liste

Skriving er raskere med kjeding i kart, fordi du ikke tar hensyn til hvor du sist skrev.

Kartoppslagg gir presis minnelokasjon etter oppslag, som korter ned letetid?