I4ISU
Indlejret software udvikling

Indholdsfortegnelse

OS and programming in Linux

1.1 Lektion 28-03-2017

1. OS
2. Compilering
3. Makefiles

Process and Threads

2.1 Lektion 04-04-2017

1. Shared data problem
2. Processer
3. User vs. kernelspace
4. Call backs

2.1.1 Processer

En proces er en aktiv enhed, en eksekvering af et program. Den består udover programkode af en program counter, stackpointer, variableværdier, datasektion (globale variabler), indholdet af en processerens registr, MMP. Processer kommunikerer med hinanden ved hjælp af IPC, styret af OS. Hver proces har sin egen virtuelle hukommelses adresse space. Processer kan ikke tilgå andre processers hukommelse.

Oprettelse af processer

- Systeminitialisering
- En kørende proces opretter en ny proces (f.eks. fork)
- En bruger opretter en ny proces
- Initialisering af et batch job
- Init processen

Processernedlæggelse

- Normal exit: Processens job er udført

- Error exit: Når der er sket en fatal fejl, fx hvis brugeren vil kompile et ikke-eksisterende program, ulovlig instruktion, reference til ugyldig hukommelse, division med nul
- Nedlagt af en anden proces: Når f.eks. en proces anvender kill til at få OS til at nedlægge en proces i UNIX.

Fem stadier

- Nyt stadie: Processen bliver oprettet
- Running: Processen kører i CPU'en
- Blocked: Afventer et ekstern event, fx I/O for at kunne fortsætte
- Ready: Klar til at køre i CPU hvis denne bliver ledig
- Terminated: Har afsluttet eksekvering
  - En baggrundsproces hedder en daemon

2.1.2 Tråde

Kaldes også for tasks, jobs eller letvægtede processer. En tråd er en del af en proces, som skal eksekveres i CPU'en. Alle processer har minimum en tråd. CPU'en skifter imellem de forskellige tråde. Skaber illusionen om parallelitet (hvis der kun er en kerne). En tråd kan være i samme stadier som en proces. En tråd har: Stack, PC, registre. Imellem sig deler trådene: kodesektion, datasektion, opgaver (som fx open files og signaler)

Ligheder ml. tråde og processer

- Deler CPU
- Eksekveres sekventielt
- Kan lave børn
- En ny tråd overtager hvis en anden bliver blokeret

Forskelle ml. tråde og processer

- Tråde er ikke uafhængige af hinanden
- Tråde har adgang til de samme adresser i opgaven (problem)
- Altid designet til at hjælpe hinanden

- Processer kan hjælpe hinanden alt efter oprindelse

Fordele ved tråde

- Tråde koster ikke mange ressource (letvægtede processer)
- Behøver kun en stack og lidt hukommelse
- Behøver ikke adressespace el. globale data, programkode eller OS ressourcer
- Context switch er hurtigt (udskiftning af tråde i CPU'en)

2.1.3 User-threads

Hver proces bliver af kernen opfattet som enkelttrådede processere ligegyldigt hvor mange tråde den har. Ved trådskifte bliver der ikke lavet systemkald pga. implementationen i userspace bibliotekerne. Hver proces har et threadtable over sine tråde.

Fordele

- Anvendes i OS der ikke understøtter flertrådethed
- Alt er gemt i user space adresse space: PC, registers, stack og en controlblock
- Kernen bliver ikke forstyrret
- Ikke meget dyrere i ressourcer end et procedure call

Ulemper

- Kun et timeslice per proces ligegyldig antallet af tråde. Hver tråd skal overgive kontrollen til andre tråde når de er færdige (nonpreemptive)
- Hvis én tråd blokerer vil hele processen stoppe (selvom der er andre tråde i samme proces som kunne fortsætte)
- Kan ikke udnytte flerkernesystemer

2.1.4 Kernel-threads

Kernen kender til trådene og har kontrollen over dem. Kernen har thread table udover proces table. Tråde dannes og styres vha. system kald.

Fordele

- Prioritering af tid: En proces med mange tråde kan få mere CPU-tid tildelt af scheduleren
- Rigtig godt til applikationer der ofte blokerer
- Kan udnytte flerkernesystemer

Ulemper

- Langsomt og ueffektivt - 100 gange langsommere end userspacet-hreading
- Fuld thread control block TCB er nødvendig for hver tråd. Det skaber et stort overhead og kernekompleksitet.

Thread Synchronization I

3.1 Lektion 18-04-2017

1. Mutex og Semaphores
2. Conditionals
3. Producer/consumer problem

3.1.1 Mutex og Semaphores

For at løse samtidighedsudfordringen indføres mutex og semaphorer. Deres formål er at beskytte data således at de ikke tilgås af flere ad gangen. En lås til hver, uafhængig af andre data.

Problemer

1. Glemmer at tage
2. Glemmer at frigive
3. Tager den forkerte
4. Holder unødigt længe
5. Deadlocks

Mutex skal frigives af den tråd der har taget den.
Semaphorer har intet ejerskab. Kan være binær eller counting. Behøver ikke frigives af den tråd der tog den. Kan derfor bruges til signalering.

3.1.2 Conditionals

Conditionals anvendes sammen med mutex til at signalere tråde imellem at der er sket en opdatering af fælles data. En tråd kan sættes til at vente på netop sådan en opdatering, f.eks. i øvelsen park-a-lot. Man implementerer en cond_wait i en while-loop for at fange falske vækninger.

3.1.3 Producer/consumer problem

Consumer og producer deler en buffer af en fikseret størrelse. Consumer må ikke forsøge at tage noget ud af en tom kø, og producer må ikke lægge noget i en fuld kø. I stedet skal de sove til de kan fortsætte. Vi implementerer dette med counting semaphores og mutex (husk rækkefølge for at undgå deadlocks). Man kan også bruge monitors.

Thread Synchronization II

4.1 Lektion 02-05-2017

1. Deadlocks
2. Priority inversion

4.1.1 Deadlocks

Der er fire ting der skal være opfyldt for at en deadlock kan opstå:

- Mutual exclusion: Ressourcen kan kun bruges af en proces af gangen
- Hold-and-wait: Processer der allerede har en ressource kan forespørge flere
- No preemption: Ingen ressourcer kan fratvinges sine ressourcer
- Circular wait condition: Cyclisk afhængighed

4.1.2 Priority inversion

A (HP-task) deler semaphor med C (LP-task). C har semaphoren. B (MP-task) overtager CPU'en. A kan ikke komme til så længe B har CPU, fordi C ikke nåede at frigive semaphor.

1. **Priority inheritance:** When a thread holds a mutex it is temporarily assigned the priority of the highest-priority thread waiting for the mutex.
2. **Priority ceiling:** All mutexes are assigned a (high) priority (the priority ceiling) which the owner of the mutex is assigned while it holds the mutex.

Thread Communication

5.1 Lektion 09-05-2017

1. Event/Message
2. Handlers/Dispatcher