Godkendelsesopgave 3 i "Styresystemer og multiprogrammering"

Generelt

Denne ugeopgave stilles fredag den 18. februar 2011 og skal afleveres senest mandag den 7. marts 2011 klokken 23:59:59. Den kan løses af grupper på op til 2 personer (3 kan tillades, men frarådes). Besvarelsen af opgaven vil resultere i enten 0, 1/2 eller 1 point. Pointene uddeles efter følgende retningslinjer:

- 0 point: besvarelsen har flere store mangler.
- 1/2 point: besvarelsen opfylder i store træk kravene, men har flere mindre mangler.
- 1 point: besvarelsen opfylder kravene til opgaven med kun få eller ingen mangler.

Det er en betingelse for at gå til eksamen på kurset at man har opnået mindst 4 point i alt.

Besvarelsen skal indleveres elektronisk via kursushjemmesiden (Absalon). Besvarelsen bør ske ved aflevering af en enkelt fil. Brug 'zip' eller 'tar.gz' til at samle flere filer. Opgavenummer og navne på gruppemedlemmer skal fremgå tydelig af første side i besvarelsen. Når I indleverer bør efternavne på alle gruppemedlemmer indgå i filnavnet - desuden skal opgavenummer fremgå af navnet:

```
efternavn1-efternavn2-efternavn3-G1.<endelse>
```

Jeres aflevering skal være i et format der kan læses på DIKUs systemer uden problemer (således bør formatet f.eks. ikke være MS Word dokumenter). Se i øvrigt "Krav til G-opgaver" siden på kursushjemmesiden under "G-opgaver" menupunktet.

I skal kun aflevere én fuld besvarelse per gruppe. Personen, som afleverer skal markere sine gruppemedlemmer via Absalon.

Om dokumentation af jeres løsning

Afleveringen skal indeholde en kortfattet rapport der dokumenterer hvilke vigtige observationer I har gjort jer, antagelser jeres løsninger afhænger af (som I enten har valgt, eller som er valgt af designerne af BUENOS) og desuden skal I redegøre kortfattet for designbeslutninger i den kode I har implementeret. I skal også huske at kommentere jeres kildetekst så den er let at forstå. Især er det vigtigt at dokumentere hvis I har foretaget ændringer i allerede implementerede dele af BUENOS.

1 Denne uges tema: Implementering af synkroniseringsmekanismer

I denne afleveringsopgave skal I implementere et antal funktioner og typer til at håndtere låse og betingelsesvariable. Første del af G3 afleveringsopgaven handler om at implementere funktioner og typer til at håndtere låse og betingelsesvariable for kernetråde. Anden del af afleveringsopgaven handler om at tilføje nye systemkald til BUENOS kernen sådan at brugerprocesser kan benytte sig af låse og betingelsesvariable. Kapitel 5 fra BUENOS guiden, samt kapitlerne fra "Operating System Concepts" (SGG) om tråde og synkronisering kan være nyttige at (gen)læse.

G3.1: Implementering af låse og betingelsesvariable

Opgave 1 Implementér følgende funktioner og tilsvarende typer til håndtering af låse for kernetråde:

- "**int** lock_reset(lock_t *lock);"

  Initialisér en *allerede allokeret* "lock_t" struktur sådan at man efterfølgende kan låse og oplåse låsen. Funktionen returnerer 0, hvis låsen kunne initialiseres og et negativt tal, hvis låsen ikke kunne intialiseres.

- "**void** lock_acquire(lock_t *lock);"

  Låser låsen "lock". En simpel løsning af "lock_acquire" kunne benytte *busy-waiting* til at lade kernetråde vente, men dette er ikke en effektiv løsning. En bedre løsning benytter ventekøer til at lade kernetråde vente.

- "**void** lock_release(lock_t *lock);"

  Et kald til denne funktion låser låsen "lock" op.

Definitionen af "lock_t" typen er op til jer.

Opgave 2 Implementér nedenstående funktioner og tilsvarende typer til håndtering af betingelsesvariable med MESA-semantik (kaldes "*signal and continue*" side 247 i [SGG]) for kernetråde. For betingelsesvariable med MESA-semantik gælder det at tråden, som kalder "condition_signal" (S) *ikke* sættes til at vente med kaldet til "condition_signal" og at de ventende tråde (W), der har kaldt "condition_wait" forinden, først bliver startet når S enten afsluttes eller selv venter. Således kan trådene W kun vide at den betingelse, de ventede på, *har* været opfyldt–men ikke at den nødvendigvis stadig er det.

```
int condition_reset(cond_t *cond);

void condition_wait(cond_t *cond, lock_t *condition_lock);

void condition_signal(cond_t *cond, lock_t *condition_lock);

void condition_broadcast(cond_t *cond, lock_t *condition_lock);
```

Definitionen af "cont_t" typen er op til jer.

Jeres funktioner og typer skal placeres i filerne `kernel/lock_cond.c` og `kernel/lock_cond.h`. For at jeres funktioner oversættes med sammen med resten BUENOS skal I tilføje `lock_cond.c` til listen `FILES`, der er defineret i `kernel/module.mk`.

G3.2: Implementering af nye systemkald

Anden delopgave af denne afleveringsopgave handler om at implementere systemkald. I skal implementere et eksisterende systemkald samt tilføje et antal nye. I *behøver ikke* sikre at de systemkald i implementerer er "skudsikre".

For at brugerprocesser skal kunne benytte låse og betingelsesvariable på en fornuftig måde, skal der være en måde at to brugerprocesser begge kan få adgang til samme data. To brugerprocesser har imidlertid ikke adgang til den samme del af lageret. Vi kan løse problemet ved at introducere *brugertråde*. Tråde oprettet af en brugerprocess kan dele data, men har separate stakke. I BUENOS er systemkaldet "**int** syscall_fork(**void** (*func)(**int**), **int** arg)" måden hvorpå en brugerprocess/tråd kan starte nye tråde. Et kald fra en brugerprocess/tråd, P til "syscall_fork" har følgende effekt:

- Der oprettes en ny (kerne)tråd, som udføres i samme adresserum som P.
- Den nye tråd starter ved funktionen "func" med argumentet "arg" og tråden afsluttes når "func" returnerer. Det er imidlertid *ikke* meningen at afslutning af en brugertråd resulterer i afslutning af brugerprocessen som oprettede tråden. Først når alle tråde oprettet af en brugerprocess er afsluttet kan brugerprocessen afsluttes. I denne opgave er det ligeledes et krav at funktionen "func" *afslutter tråden med et kald til "syscall_exit"* og ikke ved brug af almindelig "**return**".
- Systemkaldet returnerer 0, hvis den nye tråd kunne oprettes og en negativ værdi, hvis der skete en fejl.

Opgave 3 Implementér systemkaldet "syscall_fork".

Da en rigtig implementation af "syscall_fork" skal interagere med den del af BUENOS, som omhandler virtuel hukommelse, har vi udleveret en referenceløsning dokumenteret i filen `flertraadning.txt`. Denne løsning benytter en specifik udgave af en procesabstraktion (PCB/-process control-block), der formentlig er forskellige fra jeres egen fra jeres løsning af G2. Det er således jeres opgave at tilpasse jeres G2 løsning til den givne referenceløsning. I må naturligvis også ændre jeres løsning fra G2 så den passer til den givne referenceløsning, hvis det er nemmere for jer.

Opgave 4 Implementér et lille bibliotek af funktioner, som brugerprocesser/tråde kan kalde for at få adgang til låse og betingelsesvariable. Det skal som minimum være muligt for brugertråde/processer at:

- oprette låse med et systemkald:
  - "**int** syscall_lock_create(usr_lock_t *)"

  Et kald til "syscall_lock_create(1)" skal gøre en lås tilgængelig for brugerprocessen / trådene som udførte systemkaldet. Det ene argument til "syscall_lock_create" er en peger til den repræsentation af låse, brugeprogrammer ser. Systemkaldet kan da udfylde værdier den har brug for via den givne peger. Der er med andre ord *mulighed* for at typen "usr_lock_t" kan være forskellig fra "lock_t", men man kan også vælge at lade typerne være de samme:

  "**typedef** lock_t usr_lock_t"

Systemkaldet returnerer 0, hvis låsen kunne oprettes og ellers et negativt tal.

- låse låse med et systemkald:
  - "**void** syscall_lock_acquire(usr_lock_t *)"
- låse låse op med et systemkald:
  - "**void** syscall_lock_release(usr_lock_t *)",
- oprette betingelsesvariable med et systemkald:
  - "**int** condition_create(usr_cond_t *)"

  Et kald til "syscall_condition_create(l)" skal gøre en betingelsesvariable tilgængelig for brugerprocessen/trådene som udførte systemkaldet. Det ene argument til systemkaldet "syscall_condition_create" er en peger til den repræsentation af betingelsesvariable, brugeprogrammer ser. Systemkaldet kan da udfylde værdier den har brug for via den givne peger. Der er med andre ord *mulighed* for at typen "usr_cond_t" kan være forskellig fra "cond_t", men man kan også vælge at lade typerne være de samme:

  "**typedef** cond_t usr_cond_t"

  Systemkaldet returnerer 0, hvis betingelsesvariablen kunne oprettes og ellers et negativt tal.

- vente på betingelsesvariable med et systemkald
  - "syscall_condition_wait(usr_cond_t *, usr_lock_t *)"
- signalere på betingelsesvariable (både signalere til netop én ventende tråd og signalere til *alle* ventende tråde) med systemkaldene:
  - "syscall_condition_signal(usr_cond_t *, usr_lock_t*)"
  - "syscall_condition_broadcast(usr_cond_t *, usr_lock_t*)"

For at tilføje et *nyt* systemkald til BUENOS-kernen, skal man angive et systemkaldsnummer i `proc/syscall.h`. Nummeret for det nye systemkald skal være forskelligt fra alle de andre numre for systemkald. Ligeledes skal funktionen "syscall_handle" i `proc/syscall.c` udvides til at håndtere de nye systemkald. Endelig skal man tilføje kald til "_syscall" i `tests/lib.c` og `tests/lib.h` lige som det er gjort for de eksisterende systemkald. (Se også kapitel 6 af BUENOS guiden)