Maria Caroline Miller, 040779, twq135
Søren Pilgård, 190689, vpb984

27. februar 2012

1 Disclaimer

Programmet er ikke gennemtestet særlig grundigt, så vi garanterer ikke for om det kodede virker, men det oversætter!

2 1.a: Datastruktur for processer

Vores processer har forskellige oplysninger, som de skal have. De skal have noget at lave aka en executable, som beskriver den opgave, som processen udfører. Derudover er det vigtigt at de har et id, som de defineres på. De har en resultatvariabel, og en programtæller. En process skal også have en definition af hvilken tilstand den er - vi var valgt at de kan være kørende, frie eller zombie. Endelig har vores process også oplysninger om en evt. forælder, evt. børn, og dens søster.

3 1.b: Processtabel

Ud fra datastrukturen tilføjes i process.c en tabel, der kan indeholde alle vores processer. Derudover defineres en spinlock, som skal holdes låst når man manipulerer med processtabellen.

4 1.c: Hjælpefunktioner

Der skal laves en række hjælpefunktioner til processstyring. Disse beskrives nedenfor.

4.1 Ændringer andre steder end process.c og process.h

Init_start_up_thread i main.c bruger process_run istedet for process_start, for at skabe en process.

Main.c kalder process_init.

Vi har indført en grænse, der hedder CONFIG_MAX_PROCESSES i kernel/config.h, som sætter en øvre grænse for antallet af processer vi kan have kørende.

4.2 process_spawn

Denne funktion opretter en ny kernetråd med en ny process. Der kaldes et interrupt, som låser cpu'en ved at sørge for at der ikke kan kaldes andre interrupts mens den arbejder. Derefter kalder den spinlock på processtabellen, så den kan arbejde i den. Der køres igennem processtabellen, indtil der findes en tom plads. Hvis tabellen er fuld løsnes spinlocken på tabellen, der åbnes op for interrupts igen, og systemet sender en fejlkode tilbage (aka pid er stadig -1).

Der er lavet en løsning, som sørger for at næste gang der oprettes en ny process, starter man med at kigge tabellen fra indgangen efter hvor man sidste gang oprettede en process, hvilket burde give bedre chance for hurtigt at finde en tom plads.

Tabellen initialiseres derefter med oplysningerne om den nye process, og der oprettes derefter en ny kernetråd med denne nye process (til dette bruges blandt andet en ny hjælpefunktion process_launch (se sektionen om denne)). Derefter løftes spinlocken på tabellen, og interrupts bliver igen mulige.

4.3 process_run

Starter en process. Der sikres at tråden, som processen startes i, ikke allerede kører en anden process. Derefter gøres næsten det samme som i process_spawn, bortset fra at der tilsidst ikke oprettes en ny tråd, men at processen bare sættes igang.

4.4 process_get_current_process

Denne funktion returnerer id for den nuværende process. Der kaldes et interrupt, og id'et for processen i den nuværende tråd hentes.

4.5 process_finish

Denne funktion afslutter en process, når den er færdig med at udføre sit arbejde. Det er processen selv der kalder denne funktion. Vi starter selvfølgelig med at stoppe for interrupts, og få fat i spinlocken til tabellen. Derefter henter vi id for den nuværende process (aka den som skal afsluttes).

Det skal undersøges om den process, der skal afsluttes har nogle børn, og hvad der skal gøres med disse. Hvis der er nogle børn, der er zombier, er de allerede færdige med deres arbejde, og kan slettes direkte, men der kan være nogle som stadig arbejder. Der er flere muligheder for hvordan man kan behandle disse

børneprocesser. Den ene mulighed er at fjerne dem, da det arbejde de laver ikke længere er interessant for forældre-processen, da denne jo har valgt at afslutte sig selv. Den anden mulighed er at betragte dem som forældreløse, og dermed lade dem udføre det arbejde de er igang med. Vi har valgt den sidste løsning. Hvis man sletter en process inden den er færdig med sit arbejde, kan man risikere at man kommer til at ødelægge noget andet. Enhver opgave, der udføres i et system har jo sideeffekter, og det ville være skidt, hvis de fik systemet til at crashe.

Hvis processen, der skal afsluttes har en forælder, skal resultatet jo sendes videre til forældren, inden at denne process afsluttes. Vi sætter derfor denne process til at være en zombie, som bare venter på at dens forældre beslutter sig for at køre en join, og dermed modtage resultatet.

Hvis forælderen sover (aka venter på at dens børn bliver færdige med deres arbejde), skal den nu vækkes i sovekøen og have at vide at dens barneprocess er færdig. Og så skal spinlocken åbnes og interrupt tillades igen. Derefter afsluttes tråden med denne process.

4.6 process_join

Join sætter en process til at vente på at dens barn bliver færdig med at lave sit arbejde, så den kan samle resultatet op, og bruge dette i sit videre arbejde. Oprindeligt var funktionen af typen uint32_t. Dette er lavet om til typen int for at skabe overensstemelse med hvad process_finish tager som argument. Når brugeren laver et systemkald til syscall_exit gives der en positiv int med, samt der forventes at syscall_join returnere en int. Vi betragter det derfor som en fejl at process_join returnerer en uint32_t.

Der checkes på om pid overhovedet er legalt (aka eksisterer i vores processtabel) og om den nuværende kørende process er forældre til det pid, som vi prøver at joine med. Dette er fordi man kun kan joine med sine egne børn.

Interrupt disables, og der hentes en lås til processtabellen. Hvis barnet ikke er en zombie, er processen ikke færdig med at køre endnu, og denne process vil derfor gerne vente. Den sætter derfor sig selv i sovekøen, låser spinlocken op, og fortæller scheduleren at den gerne må hoppe til en anden tråd. Derefter venter den bare på at den får at vide at dens barn er færdig med at arbejde, hvorefter den tager spinlocken tilbage.

Den gemmer resultatet, og fjerner barnet fra processtabellen, da der ikke er brug den proces mere ved hjælp af en hjælpefunktion process_clear (se sektionen om den). Låser tabellen op, og starter interrupts igen.

4.7 process_init

Funktionen løber processtabellen igennem, og clearer den for al information. Den sætter altså alt til sådan som det var i starten. Denne funktion kaldes i starten til at konfigurere alting til at starte med. Der bruges ekstra-funktionen process_clear til dette.

4.8 process_launch

Sætter tråden til at afvikle det program som processen skal køre. Den henter den nuværende tråds process id ind, og sætter derefter process_start til at køre med den nuværende process's executable. Der er selvfølgelig sørget for at tabellen er spinlocket mens der arbejdes, og der ikke kan kaldes interrupts.

4.9 process_clear

For at gøre det nemmere at instantiere en ny tabel er lagt ud i en ny hjælpefil.

5 Opgave 2: Systemkald

Den kode som vi har genereret ovenfor skal helst kun kunne kaldes af kernen selv, og ikke direkte af brugeren. Til dette skal vi bruge ongle såkaldte systemkald, som brugerdelen af vores maskine kan kalde, når de gerne vil have udført en funktion.

5.1 syscall_exec(const char *filename)

Exec bruger vores funktion process_spawn, som den kalder med den user_context der ligger i den pågældende tråd. Den information ligger gemt i register A1. Resultatet gemmes i registret V0, som brugeren kan tilgå.

5.2 syscall_exit(int retval)

Exit bruger funktionen process_finish, som den kalder med den information brugerdelen har lagt i register A1. Den returnerer ikke noget, da man i dette tilfælde dræber tråden, og der derfor ikke skal bruges noget resultat.

5.3 syscall_join(int pid)

Join bruger funktionen process_join, som den kalder med informationen der ligger i A1, og returnerer resultatet i V0.

5.4 syscall_read(int fhandle, void *buffer, int length)

Der er valgt ikke at bruge låse, da gcd-¿read i tty.c bliver sat til at køre funktionen tty_read, og denne sørger pænt for både at sætte interrupts og spinlocks. Denne funktion kræver alle tre argumenter i registrene A1, A2 og A3, og disse sender derfor med. Resultatet af læsningen gemmes i register V0. Det er dog vigtigt at bemærke at læsningen ikke indsætter \0 i slutningen af det læste, og hvis brugeren derfor skal bruges resultatet som en streng, skal man huske at sætte det bagefter, da brugerfunktioner typisk bruger dette til at markere slutningen af en streng.

5.5 syscall_write(int fhandle, const void *buffer, int length)

Write brugre det samme system som read, bortset fra der kaldes tty_write i stedet. Den låser også pænt interrupts og spinlock. Brugeren skal i dette tilfælde huske at den længde der skal skrives skal være uden afslutningen \0, da det muligvis vil give fejl senere hen.