Deluppgift 12 Systemanropet halt

Systemanrop i Pintos

Användarprogram kör med begränsad, “OS-kontrollerad”, tillgång till minne och instruktioner. Ett systemanrop är en begäran från en process, ett användarprogram, till OS att få tillgång till en OS-kontrollerad resurs (filer, minne, etc.).

Ett systemanrop startar alltid i användarprogrammet genom att en instruktion för interrupt exekveras. PintOS tillhandahåller en program-modul som specificerar ett antal systemanrop, och hur de skall gå till (från användarprogrammets kod) i form av vanliga funktioner som användarprogrammet kan anropa. Ett användarprogram får tillgång till dessa genom att inkludera `lib/user/syscall.h`. Implementationen av dessa finns i `lib/user/syscall.c`. Denna implementation konverterar bara det vanliga funktions-anropet till ett systemanrop genom att arrangera user-stacken och “trigga” ett mjukvaru-interrupt.

Stacken organiseras som för ett vanligt funktionsanrop, men returadressen byts mot ett nummer som identifierar systemanropet som skall utföras. Dvs, först (högst upp i minnet) placeras parametrarna till systemanropet (om några) och därefter ett heltal (istället för återhoppsadress). Systemanropet `exit(111)` skulle till exempel se ut som följer (vi antar att stackpekaren startade på `BFFFFFC0`):

```
BFFFFFBC 111    (int) <första parameter>
BFFFFFB8 1      (int) <systemanropsnummer>
```

Symboliska konstanter för systemanropsnummer finns definierade som uppräkningstypen `enum` i `lib/syscall-nr.h`. Detta gör att det går att skriva koden lite tydligare genom att skriva `SYS_HALT` istället för motsvarande heltal i koden.

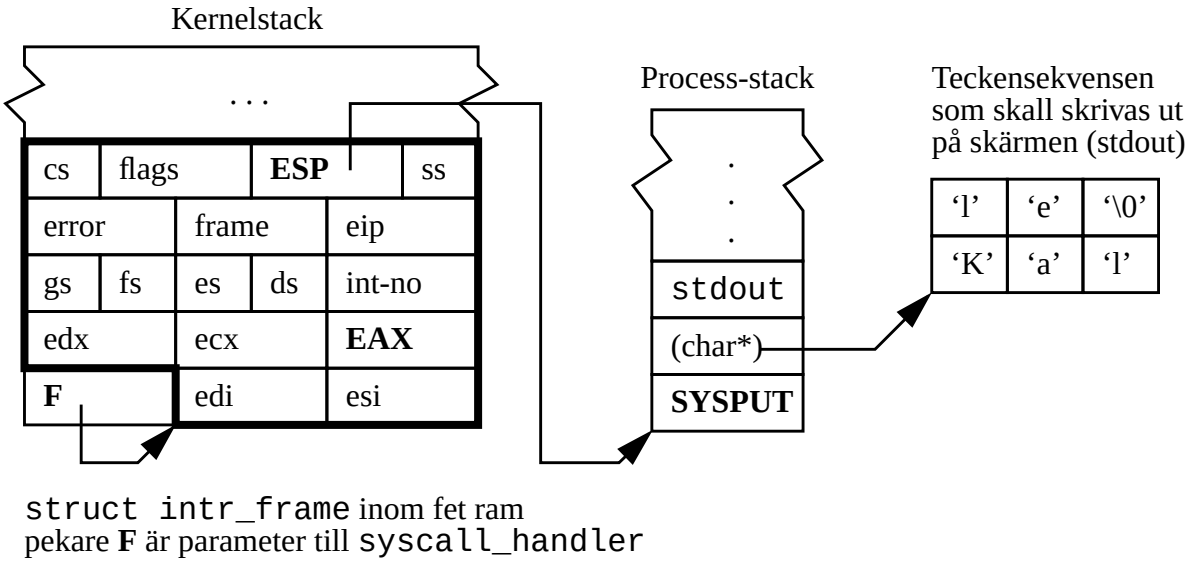
Mjukvaru-interruptet får till följd att processorn (CPU) byter till privilegierat läge (kernel-mode) och samtidigt byter stack så exekveringen i fortsättningen använder kernel-stacken. Nu sparas först alla register på kernel-stacken, så CPU när allt är klart kan återställas till det läge användarprogrammet, processen, hade innan interruptet. Koden för detta finns i `threads/intr-stubs.S` (assembler). Om du är intresserad kan du läsa koden, annars klarar du dig ändå.

När ovan CPU-context är sparat på kernel-stacken anropas en central interrupt-hanterare, `intr-handler`, som återfinns i `threads/interrupt.c`. Denna undersöker var interruptet kom ifrån och anropar en specifik interrupthanterare. En parameter skickas med. Det är en pekare till de data som lagrats på kernel-stacken (processorns register *som de såg ut* precis innan interruptet och *som de kommer återställas* när interruptet är klart). Pekaren är i form av en `struct intr_frame` som beskriver varje data. Det gör access av individuella delar enklare. Strukturen finns beskriven i `threads/interrupt.h`, och det är intressant för dig att titta vad som finns i den. Av speciellt intresse är medlemsvariabeln `esp` som kommer peka på user-stacken, samt `eax` som förväntas innehålla funktioners returvärden.

Den specifika hanterare som anropas vid mjukvaruinterrupt (för att göra systemanrop) heter `syscall_handler` och är definerad i `userprog/syscall.c`. Den nuvarande versionen skriver dock endast ut de två översta talen på user-stacken, samt avslutar tråden. Detta sista gör att systemanropet i nuläget inte kommer att returnera. Ett trådavslut innebär att OS byter till en ny tråd som helt enkelt tar bort tråden som skulle avslutas. Då kan den aldrig fortsätta exekverara. Därmed kommer användarprogramet som gör systemanropet att terminera (för tidigt).

Då du ordnat så `syscall_handler` returnerar med rätt resultat kommer vi så småningom att exekvera hela vägen tillbaka till `threads/intr-stubs.S` som ser till att återställa processorns register och återvända till “OS-kontrollerat” läge (user-mode). Därmed är systemanropet färdigt.

Stacken vid ett imaginärt systemanrop **SYSPUT** som skriver ut en sträng på `stdout` som det ser ut precis efter hopp till `syscall_handler(struct intr_frame *F)`



`struct intr_frame` inom fet ram
pekare **F** är parameter till `syscall_handler`

Testprogram

Ett testprogram som använder systemanropet `halt` finns i `examples/halt.c`. Alla testprogram i examples katalogen kompileras genom att skriva `make` i den katalogen. Du kan köra ett testprogram på samma sätt som du körde `sumargv`, byt bara ut programmets namn överallt.

För att lägga till egna testprogram måste filen `examples/Makefile` modifieras. Antag att du vill lägga till ett testprogram som är implementerat i filen `my_test.c`. Följande måste göras:

1) Lägg till `my_test` i `PROGS` listan.
2) Lägg till raden: `my_test_SRC = my_test.c`
3) Skriv `make` i examples-katalogen för att kompilera.

Debugutskrifter

För att lätt kunna sätta på och stänga av olika utskrifter, t.ex. debug-utskrifter, är det bra att definiera ett makro som ersätter `printf`:

```
#define DBG(format, ...) printf(format "\n", ##__VA_ARGS__)
```

Antag sedan att du vill göra följande debugutskrift:

```
printf("# exekverade rad %d i filen %s", __LINE__, __FILE__);
```

Med hjälp av makrot kan du istället skriva:

```
DBG("# exekverade rad %d i filen %s", __LINE__, __FILE__);
```

Finessen är att du senare kan stänga av alla utskrifter som gjorts via `DBG` genom att ändra makrot till:

```
#define DBG(format, ...)
```

Konstanterna `__LINE__` och `__FILE__` som används ovan byts automatiskt ut mod aktuell rad och aktuellt filnamn.

OBS! Notera att utskrifterna ovan startar med "# ". Detta gör att dessa rader senare ignoreras då de automatiska testprogrammen i uppgift 19 körs. Rader utan denna inledning kommer tolkas som att testet gick fel. Det finns redan ett makro `debug` i `lib/debug.h` som lägger till dessa tecken automatiskt, men det kan vara intressant att ha olika namn på `debug` i olika filer för att lättare kunna sätta på och stänga av olika meddelanden (`syscall_debug`, `process_debug` etc.).

Beskrivning av uppgiftens systemanrop

Systemanropet `halt` är deklarerat i `lib/user/syscall.h` och används av användar-programmen därefter:

`void halt (void) NO_RETURN;`

Skall *stänga av datorn* (emulatorn) med omedelbar effekt. Processen kommer inte fortsätta sin exekvering eftersom datorn stängs av.

Uppgift

Implementera systemanropet `halt` genom att lägga till nödvändig kod i interrupt-hanteraren för systemanrop (`userprog/syscall.c`). Detta systemanrop skall stänga av datorn. Pintos har en funktion `power_off` implementerad i `threads/init.c` som stänger av. Inkludera `threads/init.h` för att få tillgång till denna. Testa din implementation. Det kan vara bra att lägga till några debug-utskrifter så du ser mer av vad som händer. Testprogrammet `halt` finns i katalogen `examples`.