PROGRAM OCH PROCESSER

VAD HÄNDER NÄR MAN KÖR ETT PROGRAM?

- När man kompilerar ett program och sedan länkar ihop det så stoppar länkaren in lite extra kod i programmet. Det är denna kod som i sin tur anropar `main`-funktionen ungefär som ni anropar egna funktioner.
- Den extra kod som kommer in har bland annat två syften: Se till att ditt C-program får rätt omgivning att köras i och att städa upp efter ditt program när det är klart.

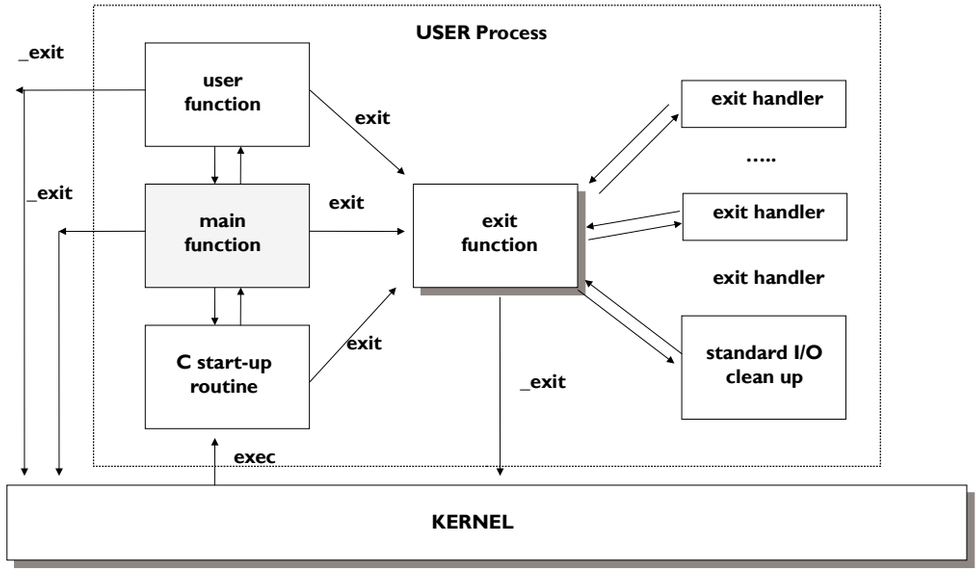
FUNKTIONSANROP

- Vad händer vid ett funktionsanrop
  - (Principiell beskrivning) Man lägger upp nuvarande adress och parametrarna på stacken, sedan hoppar man till funktionen. När funktion är klar plockar man fram den gamla stackpekaren och den gamla adressen och återgår till att exekvera den koden.
- På samma sätt med `main`
  - Den extra koden gör alltså likadant för `main`, ser till att kommandoradsargumenten går att nå som arg till `main`.
- Kommandoradsargument
  - Här gäller samma sak som vid ett vanligt funktionsanrop; extrakoden läser in argumenten och lagrar dem i en strängarray, sedan så skickar den upp antalet argument och pekaren på stacken följt av ett anrop till `main`

AVSLUTA PROGRAMMET

- Normal
  - Ett program kan avslutas normalt, dvs. genom att man "rasar ut ur" `main`, man gör `return` från `main`, man anropar `exit` eller `_exit`.
    - `exit`
      - Städar upp lite grann efter ditt program innan den låter kärnan ta över igen.
    - `_exit` `_Exit`
      - Här sker ingen städning alls utan man hoppar direkt tillbaka till kärnan
- Onormal
  - Detta sker genom ett anrop till `abort` eller att en signal gör att programmet avslutas. Vi kommer att gå igenom detta senare.
- Exithandlers
  - Genom funktionen `atexit` så kan man registrera funktioner som exekveras sedan man har anropat exit. Se exempel i boken.

HOW A C PROGRAM IS STARTED AND TERMINATED



PROCESSER

- Processer i kärnan
  - Varje program körs i sin egen process.
  - Använd `getpid` för att få reda på identifikationsnummer.
  - Unikt id, PID (heltal >= 0)

PROCESSER

- Barn
  - En process kan yngla av sig ("spawn") sub-processer genom att använda funktionen
    - `pid_t fork(void)`
  - Processerna kommer direkt efter anropet att se identiska ut (utom vad gäller returvärdet från `fork`)
    - 0 i barnet
    - Barnets PID i föräldern
    - Använd funktionen `getppid` för att få reda på förälderns PID
  - De delar ej minne med varandra. Innehållet i minnet kopieras, ofta först då vi försöker förändra det (för att undvika att kopiera saker i onödan)

PROCESSER

- Race conditions
  - Det finns en uppenbar risk när fler än en process kör och de kommunicerar med varandra. Det är att processerna hamnar i en "tävlan" om vilken som kommer först till en viss punkt i koden. Detta kan ställa till stora problem (vi kommer att titta mer på detta senare i kursen).

KILL OCH PS

- ps
  - Kommando som listar processer som körs.
  - Kolla `man ps` för lämpliga flaggor (varierar lite mellan system)
- kill
  - Kommando som kan användas för att avsluta en process
    - `kill pid`

EXEC

- Man kan starta ett program genom att först göra en `fork` och sedan göra ett `exec`-anrop som i sin tur startar programmet. Det intressanta är att detta nya program kör i den process som skapades av `fork`-anropet.
- Det finns flera olika `exec`-funktioner. Beroende på vad man vill göra, hur man vill skicka med parametrar och vilka omgivningsvariabler man vill att programmet ska använda sig av

- `int execl(const char *path,`
  `const char *arg0, .../* (char *)0 */);`
- `int execle(const char *path,`
  `const char *arg0, .../* (char *)0 */,`
  `char * envp[]);`
- `int execlp(const char *file,`
  `const char *arg0, .../* (char *)0 */);`
- `int execv(const char *path,`
  `char *const *argv);`
- `int execve(const char *path,`
  `char *const *argv,`
  `char * envp[]);`
- `int execvp(const char *file,`
  `char * argv[]);`

WAIT

- `pid_t wait(int *status)`
- `pid_t waitpid(pid_t pid,`
  `int *status,`
  `int options)`
  - ett anrop till `wait/waitpid` kan:
  - blockera om alla barn fortfarande körs
  - returnera omedelbart med statusen för barnet
  - returnera omedelbart med en felkod (om processen saknar barn)

WAIT

- Om barnet avslutas innan föräldern lagras info om processen (bl.a. status) tills dess föräldern avslutas eller anropar någon variant av `wait` för att läsa av infon
  - barnet blir en zombie process
- Om föräldraprocessen avslutas innan barnet kommer föräldrarollen att övertas av `init`-funktionen (PID 1)

WAIT - MACRON

- Används för att undersöka statusen
- `WEXITSTATUS(status)`
  - Tar fram exit status
- `WIFEXITED(status)`
  - Kollar om processen avslutades normalt
- `WIFSIGNALED(status)`
  - Sant om status indikerade att processen avslutades felaktigt (genom en signal)
- `WIFSTOPPED(status)`
  - Sant om barnet stoppat