Obligatorisk uppgift 1 - `mexec`

Din uppgift är att skriva ett C-program som kan exekvera en *pipeline*. En *pipeline* är en sammansättning av program som körs parallellt där ett programs utdata blir nästa programs indata.

Ett exempel på en sådan pipeline i **bash** är

```
cat -n mexec.c | grep -B4 -A2 return | less
```

som kommer att läsa och radnumrera innehållet i filen *mexec.c* för att sedan leta reda på stycken i koden som innehåller ordet `return`. Slutligen skickar det resultatet till kommandot `less` för enklare läsning av stora resultat.

I ditt program ska *pipelinen* läsa in rad för rad istället för att *pipe*-tecknet `|` används. Samma pipeline som ovan kommer i ditt fall vara:

```
cat -n mexec.c
grep -B4 -A2 return
less
```

Uppgiften ska utföras enskilt.

Gränsyta

Programmet ska heta `mexec` och indata ska kunna läsas in antingen direkt från *standard input* eller via en angiven fil. Programmet ska endast acceptera noll eller ett argument och om användaren skickar in fler argument ska ett lämpligt felmeddelande skrivas ut (till *standard error*) samt att programmet då ska avslutas med returkoden `EXIT_FAILURE`.

Inläsning från standard input

En fil omdirigeras till programmets *standard input* med hjälp av skalets operator `<`, observera att du inte ska göra omdirigeringen själv utan att detta görs av ditt skal.

```
./mexec < file
```

Det ska även gå att starta programmet utan en given indata med

```
./mexec
```

för att sedan skriva in kommandona rad för rad. Inläsning ska ske på exakt samma sätt som då operatorn `<` används. I båda fallen ska programmet läsa från *stdin* tills dess att ett `EOF`-tecken läses in.

För att skicka ett `EOF`-tecken och avsluta inläsningen när alla kommandon är inmatade använder du i de flesta terminaler *Ctrl-D*.

Inläsning från fil

Programmet ska även kunna ta indata från en fil som ges som första argument. En fil skickas då in till programmet med

```
./mexec file
```

och det är upp till programmet att öppna den angivna filen för läsning.

Indata

Formatet som ska stödjas för indata är:

- Varje kommando skrivs på en egen rad.
- Antal kommandon finns ingen specificerad övre gräns för.
- Indata innhåller inga blanka rader.
- Argument till kommandon separeras med ett blanksteg.
- Citattecken ska inte stödjas, utgå från att varje argument är ett sammanhängande ord.
- Maximal radlängd är 1024 tecken.

Utdata

Programmet i sig ska inte ge någon utdata på varken *standard input* eller *standard error* vid en korrekt körning. Utdata ska komma från de program som exekveras.

Kompilering

Programmet ska gå att kompilera utan varningar med hjälp av kommandot `make` och den *makefile* du levererar tillsammans med programmet. Separatkompilering, som även tar hänsyn till alla eventuella *header*-filer, ska användas. Observera att din *makefile* ska skrivas manuellt och ej genereras av något verktyg.

Vid kompilering ska åtminstone följande flaggor användas för alla steg:

```
-g -std=gnu11 -Werror -Wall -Wextra -Wpedantic
-Wmissing-declarations -Wmissing-prototypes -Wold-style-definition
```

Kontroll av anrop

Alla funktionsanrop som returnerar eller på annat sätt ger tillbaka en indikation för hur anropet gick ska kontrolleras. Detta innefattar till exempel `malloc` som returnerar `NULL` om minne ej gick att allokera.

Dessa funktioner behöver du *inte* kontrollera:

- `write`, `printf` och `fprintf` (eller annan utskriftfunktion)
- `close` och `fclose`
- `free`

Om ett anrop misslyckats ska programmet skriva ut ett lämpligt felmeddelande på *standard error* för att sedan avsluta med returkoden `EXIT_FAILURE`. Om anropet ändrar värdet på `errno` vid fel ska funktionen `perror` användas för att skriva ut felmeddelandet.

Kontroll av barnprocesser

När programmet väntar in sina barnprocesser ska returkoden för respektive barnprocess granskas för att se om den är skild från 0 (något fel har inträffat). Om någon barnprocess returnerar ett värde skilt från 0 ska *mexec* returnera `EXIT_FAILURE` vid avslut.

Rapport

Förutom kod och en *Makefile* ska en rapport lämnas in. Den rapport som ska skrivas ska ha formen av en vanlig rapport och innehållsmässigt vara en "utökad" *man*-sida. Målgruppen för rapporten är studenter med motsvarande kunskaper i programmeringsspråket C men som inte läst kursen. Rapporten ska innehålla följande delar:

- Försättsblad
- Namn (*Name*)
- Synopsis (*Synopsis*)
- Beskrivning (*Description*)
- Lösning
- Exit-status (*Exit status*)
- Diskussion och reflektion

Försättsbladet ska vara en separat sida och innehålla titel, ditt namn, din cs-användare samt kursens namn.

För att se vad delarna namn, synopsis, beskrivning och exit-status bör innehålla får du studera ett antal olika *man*-sidor.

Lösningsdelen är det som är den huvudsakliga utökningen jämfört med en *man*-sida. I denna del ska du ge en övergripande beskrivning av din lösning, inkluderande en beskrivning av huvudalgoritmen för programmet (tänk på vad uppgiften syftar till att behandla så du inte abstraherar bort viktig funktionalitet från din huvudalgoritm).

I diskussions- och reflektionsdelen kan du skriva om problem som du stötte på när du gjorde uppgiften, vad du tyckte om den och övriga åsikter som du vill framföra.

Övriga krav på rapporten:

- Väl strukturerad
- Väl formaterad
- Väl formulerad

- Utan språkliga fel
- Eventuella tabeller, figurer, etc ska hanteras korrekt
- Eventuella referenser ska hanteras korrekt enligt Harvard-stilen (https://www.umu.se/bibliotek/soka-skriva-studera/skriva-referenser/)

Övriga krav på lösningen

Förutom de krav som nämns i ovanstående beskrivning ska även din lösning uppfylla följande krav:

- Din lösning ska var så enkel som möjligt. Det gäller både algoritmiskt och kodmässigt. Din lösning ska enkelt förstås av annan student som läser denna kurs och som tagit till sig av kursmaterialet som examineras i denna obligatoriska uppgift.
- Programmet ska använda sig av följande funktioner:
  - `fork`
  - `pipe`
  - `close`
  - `dup2` (eller annan variant av `dup`)
  - `execvp` (eller annan variant av `exec`)
  - `wait` (eller annan variant av `wait`)
- Varje kommando ska läsa från föregående kommando i *pipelinen*, eller *standard input* för första kommandot.
- Varje kommando ska skriva till nästa kommando i *pipelinen*, eller *standard output* för sista kommandot.
- Alla kommandon ska köras parallellt.
- Programmet får inte ha några minnesläckor, använd verktyget *valgrind*. Detta gäller även vid felhantering.
- Programmet ska kunna köra ett godtyckligt antal kommandon och ska därför inte använda hårdkodade storlekar på datatyper för detta. Använd funktionen `realloc` eller en dynamisk datatyp för att lagra data kopplat till antal kommandon.
- Då programmet körs med en fil som indata får filen inte manipuleras under körningen.
- All eventuell utskrift i samband med felhantering ska ske till *standard error*.
- Koden ska ha god abstraktion, ha bra struktur, ha bra modularisering, samt ha en lämplig uppdelning i funktioner.
- Koden ska skrivas med god kodkvalité (indentering, variabelnamn, kommentarer, funktionskommentarer, etc.).
- Programmets huvudfil ska heta *mexec.c*.
- Inlämningen ska klara alla tester i labres.

Testkörningar

Eftersom ditt program, om det fungerar på ett felaktigt sätt, kan försöka skapa nya processer i en oändlig loop ska inte datorerna *salt* och *peppar* användas för testkörningar. Använd istället någon av datorerna *itchy* eller *scratchy*. Dessa två datorer är menade för testkörningar och det ska inte vara någon fara om ett trasigt program körs där.

Ett tänkbart problem är att ditt program skapar så många processer att du inte längre kan göra något på din användare och därmed inte kan stänga ned programmet. Vi rekommenderar därför att du kör kommandot `ulimit -u 50` innan du startar ditt program. Detta ställer om din *bash* session till att tillåta max 50 processer. Använder du något annat än *bash* får du själv ta reda på hur du gör motsvarande inställning.

Bedömningsmall

Tillsammans med denna specifikation finns den bedömningsmall som kommer att användas vid bedömningen av din inlämning (ligger i Canvas på samma sida som specifikationen).

Inlämning

Kod, *makefile* och rapport lämnas in via labres innan utsatt deadline.