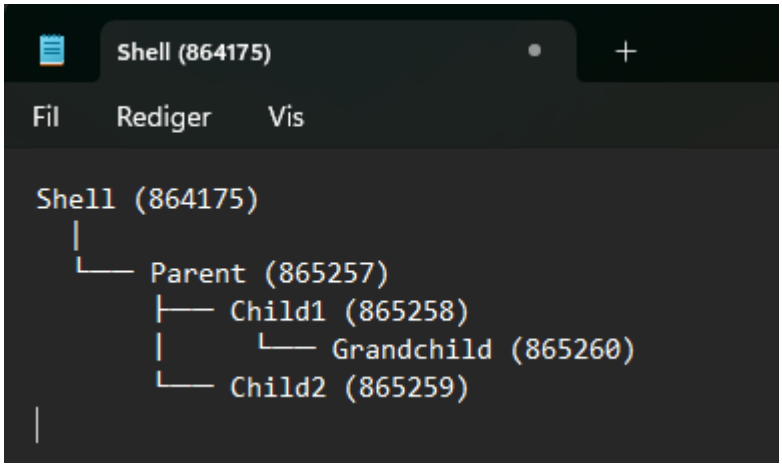
Oppgave 1:

Deloppgave A: Kompiler og kjør programmet

```
emilbe@itstud:~/htdocs/Intro2OS/oblig/4$ ./oppgave_1
Jeg er prosess 865257, min forelder er 864175
Jeg er prosess 865258, min forelder er 865257
Jeg er prosess 865259, min forelder er 865257
Jeg er prosess 865260, min forelder er 865258
```

Deloppgave B: Figur av prosestreet



Deloppgave C: Kort forklaring

Selv om *fork* kun kalles to ganger, opprettes det tre nye elementer. Første gangen *fork* blir brukt, opprettes det én ny prosess, nemlig det første barnet (Child 1). Andre gangen *fork* blir brukt, blir det opprettet enda et nytt barneelement (Child 2), men også et barnebarn av første *fork* (Grandchild). Dette er fordi sisten gangen vi kaller på *fork* kjøres den både av den opprinnelige prosessen og av barnet som ble opprettet ved første *fork*.

Deloppgave D:

```
emilbe@itstud:~/htdocs/Intro2OS/oblig/4$ ./oppgave_1d > oppgave_1d.txt
```

```
emilbe@itstud:~/htdocs/Intro2OS/oblig/4$ cat oppgave_1d.txt
Jeg er prosess 868015, min forelder er 864175
Jeg er prosess 868017, min forelder er 1
Jeg er prosess 868016, min forelder er 1
Jeg er prosess 868018, min forelder er 1
```

Over ser du at jeg har redirigert utskriften av c-programmet til en .txt fil som jeg deretter har åpnet med *cat*.

Vi ser på utskriften at mange av elementene har fått forelder PID 1. Grunnen til dette er at forelderen til noen prosesser avsluttes tidlig, og barneprosessene blir adoptert av init

(stamfar), som har PID = 1. Kort sagt blir egentlig disse barneprosessene foreldreløse, og systemet gir derfor foreldrerettighetene til «systemd» som adopterer disse prosessene. Hvis man vil unngå at dette skjer, må man «vente på tur», som vi si av hvert barneelement må få fullføre, før forelderen avslutter, eksempelvis ved å bruke «wait()».

Oppgave 2

Deloppgave A: Utskrift

```
emilbe@itstud:~/htdocs/Intro2OS/oblig/4$ ./oppgave_2
Jeg er barneprosessen med PID 872038
Jeg er forelderprosessen med PID 872037
Barneprosessen 872038 har terminert med returstatus lik 42
```

Her vil barneprosessen kjøre først, sove i ett sekund, og avslutter deretter med returstatus 42. Foreldreprosesen venter til barneprosesen er ferdig og skriver deretter ut returstatusen.

Deloppgave B: Utskrift uten wait()

```
emilbe@itstud:~/htdocs/Intro2OS/oblig/4$ ./oppgave_2
Jeg er forelderprosessen med PID 872191
Barneprosessen 872192 har terminert med returstatus lik 0
Jeg er barneprosessen med PID 872192
```

Forskjellen mellom utskriften i Oppgave A og B er flere. Først og fremst vil forelderprosessen kjøre først, ettersom den ikke må vente på at barneelementet blir ferdig, grunnet fjerningen av wait() og ettersom barneprosessen har en soveperiode på ett sekund. I tillegg vil den ikke inkludere returstatus 42, ettersom forelderprosessen kjører før barneprosessen, som vil si at barneprosessen ikke har rukket å returnere 42. Programmet kjører derfor nå i «feil» rekkefølge, med tanke på resultatene vi egentlig vil ha.

Oppgave 3

Deloppgave a: ps -l

```
emilbe@itstud:~/htdocs/Intro2OS/oblig/4$ ./oppgave_3 &
[6] 875077
emilbe@itstud:~/htdocs/Intro2OS/oblig/4$ Forelderprosessen med PID 875077 starter en evig løkke
Barneprosessen med PID 875078 avslutter
ps -l
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  3765  864175  864174  0  80   0 -  2560 do_wai pts/11   00:00:00 bash
0 T  3765  864718  864175  0  80   0 - 43662 do_sig pts/11   00:00:00 emacs
0 T  3765  865183  864175  0  80   0 - 43662 do_sig pts/11   00:00:00 emacs
0 T  3765  866098  864175  0  80   0 - 43662 do_sig pts/11   00:00:00 emacs
0 T  3765  869654  864175  0  80   0 - 43663 do_sig pts/11   00:00:00 emacs
0 T  3765  874406  864175  0  80   0 - 43697 do_sig pts/11   00:00:00 emacs
0 S  3765  875077  864175  0  80   0 -   615 hrtime pts/11   00:00:00 oppgave_3
1 Z  3765  875078  875077  0  80   0 -     0 -      pts/11   00:00:00 oppgave_3
0 R  3765  875085  864175  0  80   0 -  2805 -      pts/11   00:00:00 ps
emilbe@itstud:~/htdocs/Intro2OS/oblig/4$
```

Deloppgave B: Bli kvitt zombieprosessen

For å bli kvitt zombieprosessen kan man «drepe» forelderelementet til zombien. Dette kan gjøres ved å skrive: $ $kill < PID\ til\ forelderelement >$ eventuelt $ $kill - 9 < PID\ til\ forelderelement >$. Denne virker som "sure kill" og sikrer at prosessen faktisk blir terminert.

```
emilbe@itstud:~/htdocs/Intro2OS/oblig/4$ kill 875077
emilbe@itstud:~/htdocs/Intro2OS/oblig/4$ ps -l
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  3765  864175  864174  0  80   0 -  2560 do_wai pts/11   00:00:00 bash
0 T  3765  864718  864175  0  80   0 - 43662 do_sig pts/11   00:00:00 emacs
0 T  3765  865183  864175  0  80   0 - 43662 do_sig pts/11   00:00:00 emacs
0 T  3765  866098  864175  0  80   0 - 43662 do_sig pts/11   00:00:00 emacs
0 T  3765  869654  864175  0  80   0 - 43663 do_sig pts/11   00:00:00 emacs
0 T  3765  874406  864175  0  80   0 - 43697 do_sig pts/11   00:00:00 emacs
0 R  3765  878253  864175  0  80   0 -  2805 -      pts/11   00:00:00 ps
[6]   Terminated              ./oppgave_3
emilbe@itstud:~/htdocs/Intro2OS/oblig/4$
```