7.oblig

Oppgave 1

det finnes en gobalt variabel *FIFO_PATH* som definerer adressen til pipe programmet skal burke.

programmet main kjører 2 funksjoner: writer og reader.

Funksjonen *writer*

Denne funksjonen oppretter en array som inneholder en string, som senere skrives inn i pipe. Deretter brukes system call funksjonen *open* som oppgir stien til pipe og modus. I dette tilfellet er write-only mode.

Den verdien som funksjonen open returnerer lagres i variabelen *fd*. Dette er et lite non-negative tall som brukes senere som index som referer til posisjon.

Deretter vil funksjonen *write* skrive «string-en» (message) i pipe, og printer «message» på terminalen.
Programmet låser tilgang til posisjon til pipe.

Funksjonen reader

Denne funksjonen skal lage plass nok til 100 elementer (buffer). Deretter skal funksjonen *open* brukes på nesten samme måten som ble brukt i funksjonen *write.* Men her skal funksjonen *open* åpne pipe ved hjelp av *FIFO_PATH i read-only mode.*

deretter vil funksjonen read lese meldingen (message) skrevet inn i pipe og legre det i buffer.

Main funksjon

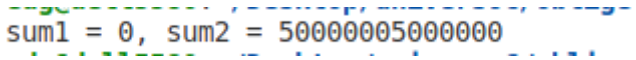
Oppretter en variabel med type PID, og et FIFO med retigheter 0666 (skrive og leseretigheter). Fork funskjonen gjør mulig å lage en kopi av PID objektet(Parent). Kopien kalles child og har null som verdi. Child kjører funksjonen *write* og Parent prosessen skal kjøre *reader* funsjonen.

«named pipe» er et kommunikasjonspunkt mellom 2 forsklellige prosesser (child og parent) som utføres først en funksjon som skriver en melding og andre process som leser Meldingen.

Oppgave 3

a)

./read_shm ; ./write_shm



forklaring: verdien til sum1 er null fordi programmet read leser en tomt array. Write programmet har ikke skrevet noe før read programmet ble kjørt.

b)

./write_shm ; ./read_shm

sum1 = 50000005000000, sum2 = 50000005000000

forklaring: write programmet skriver inn i array A (felles array) alle tall fra 1 til 10000000. deretter kjøres Read programmet som skal lese verdien til hvert element i Arrayen A og lagre en summering av disse i variabel sum1.Deretter write printer verdien til sum1 og sum2, begge er like.

c)

(./write_shm &) ; ./read_shm

forklaring: Dette minner meg oppgaver om koordinering mellom to eller flere funksjoner.Jeg kjørte flere ganger samme kommando, og fikk forskjellige verdier som resultat av sum1.

Her kjøres barneprosesser uten mekanismer for å unngå race condition i minnehåndtering. Når to prosesser kjøres samtidig uten f.eks. Mutex, wait, signal og andre metoder for koordinering.