# ISU: 7. Processes and IPC in Linux

## Processes in Linux

* En proces/program i linux
  + Har en stack med lokale variable, returværdier – LIFO
  + Heap med ”free-store”, dynamisk allokeret hukommelse
  + Memory mapping til fysisk hukommelse, inkluderede dynamiske libraries
  + Variables and ELF?

### fork() - What and how

* Skaber en ny proces ved brug af fork()
* Fork – forker en proces i en parent og en child proces – hvor child er identisk til parent
  + Parent for returneret PID fra child process
  + Child for returneret PID 0
  + Således kan man tjekke på om programmet bliver executed af parent eller child
* Forældre venter på at børn bliver færdige vha. wait()
* Og børn slutter ved at kalde \_exit(int) - ikke **exit(int)!!**

### Challenges associated with using fork()

* At vide hvor man er i programforløbet

### Challenges using process shared mutexes, conditionals & semaphores

* Det virtuelle adresse område er duplikeret fra parent til child
* Copy on write – data bliver kopieret og ændret, hvis der forsøges ændret
  + Inklusiv tilstand af mutexes, conditonals mm.
  + Inkluderer åbne fil descriptors 🡪 potentielt problem da ????????
  + Kun den kaldende tråd bliver duplikeret i barnet
* Ikke kopieret:
  + Memory map, memory locks, shared memory…
* 🡪 Kun asynkront sikre signaler må bruges!

Exec()-family?

## Shared memory, Queues & Pipes

* Metoder til Inter process communication

### Their design

**Pipes:**

* Names pipes og anonyme pipes
* En named pipe er en halv duplex måde at kommunikere mellem to processer
  + En skriver til pipe’en mens en anden læser fra pipe’en
  + Pipe’en holder data indtil de er læst
  + Enten blokerer skriveren eller læseren, indtil den anden deltager
  + I linux 🡪 named pipe = fil

**Message Queues:**

* Tråde blokerer – læser på tom kø og skriver på fuld kø
* Bruges til at sende data mellem tråde i separate processer
  + One way
* Tilgængeligt fra OS

**Shared memory:**

* Processer kan blive enige om at dele memory section
* Hvor de kan læse fra og skrive til

### How do you use them

**Pipes:**

* Reader:
  + Laver pipe node
  + Åbner pipe, læser fra pipe, lukker piper og sletter pipe node
* Write:
  + Åbner pipe, skriver til pipe og lukker pipe

**Message Queues:**

* Sender skal have msgID, en besked at sende, størrelse på beskeden og et besked flag
* Ligeledes skal modtageren have de samme parametre – plus information om beskedtype
* Message queue id skal oprettes med en nøgle??
  + msgsnd(mqId1, (void\*) &msg1, sizeof(MessageType1.data), 0);
  + msgrcv(mqId2, (void\*) &msg2, sizeof(MessageType2.data), 0, 0);

**Shared memory:**

* Shm segmentet er created og inkluderet vha. system kald
  + Create og attach
* Når det er oprettet kan begge tilgå segmentet
* Mutex skal implementeres i begge processor – mutex’en skal være pre-allokeret i shared memory
  + Ligeledes kan conditionals bruges – skal også preallokeres i shared memory

### What can and can’t they do - Design  considerations

**Pipes:**

* Man ved hvornår de er slettet
* Simpelt
* F.eks. DDD er frontend for gdb med pipes
* Ulemper:
  + Strøm orienteret – ikke altid godt – FIFO
  + Kun to processer kan bruge den
  + Synkronisering? Hvad sker der hvis en af processerne dør
  + Data skal være serialiseret
  + Man skal kende antallet af beskeder – ellers bliver det noget rod
  + 4 copy instruktioner (i kernels pace)
  + Hvem laver og nedlægger?

**Message Queues:**

* Struktureret data – hele bidder kan sendes – enten har man det eller ej
* Prioritetsbaseret – Ikke nødvendigvis FIFO!
* Flere processer kan læse og skrive til køen
* Event baseret
* Pipes er bygget på message queues 🡪 MQs er hurtigere
* Ulemper:
  + Synkronisering er nødvendigt 🡪 form for sockets
  + Hvis en af parterne dør
  + To køer er nødvendigt for duplex

**Shared memory:**

* Meget hurtigt eftersom det er den samme hukommelse der bliver tilgået
* Reducerer behovet for memory copies drastisk
* Flere processer kan dele det samme memory
  + Også tråde..
* Ulemper:
  + Synkronisering mellem tråde er nødvendigt!
  + Hvad sker der hvis en part dør under skrivning/læsning? Med mutex’en?
  + Hvem opretter og nedlægger? Og hvornår er det klart til brug?
  + Man ved ikke om deltagere er døde
  + Data skal være fladt

### Data Serialization or not? What is it and when to use it?

Pipes:

* Data skal være serialiseret – til fladt format
  + Ingen pointere, styr på endianness
  + Pain old data

**Pipes:** Point 2 Point communication, Streamed data, Very simple and very portable

**System:** V Message Queues, Structured/Packet based with priority, Event driven

**Shared Memory**: Extremely fast memory transfer between processes, Willing to pay for the extra coupling and control, N-way communication