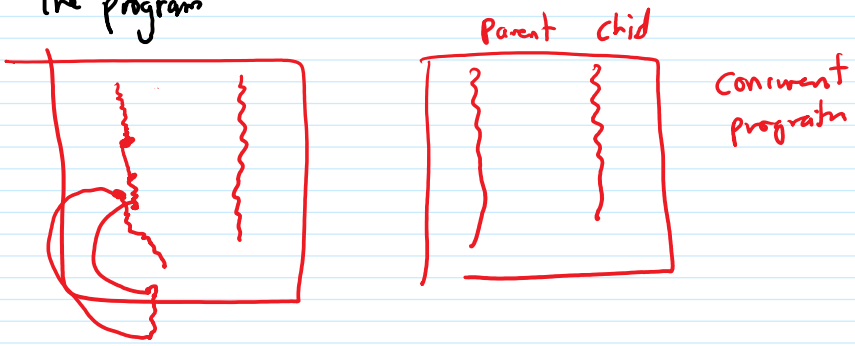
Concurrent Programming

- Until now ⇒ Program execution involved one flow of control throught the program



Example ⇒ can we write programs that run as multiple processes cooperating to achieve a common goal.

↳ A program has to achieve program.

- To cooperate, processes must somehow communicate.

Inter Process Communication (IPC)

Ideas ① Process can communicate using Files

Example: communication btw a parent and child process

Parent process creates 2 files before forking child process

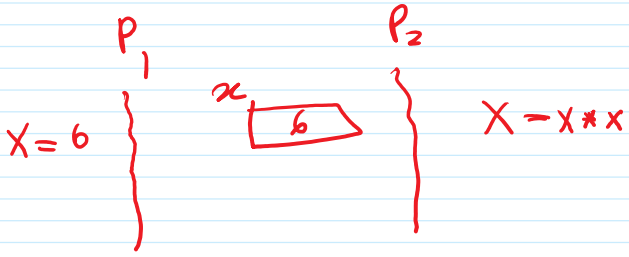
- Child inherits file pointer from parent and can use one file for the parent to write into and one for the child to read from.

Idea 2

- OS supports something called a pipe.
  ↳ corresponds to 2 file descriptors (int fd[2])

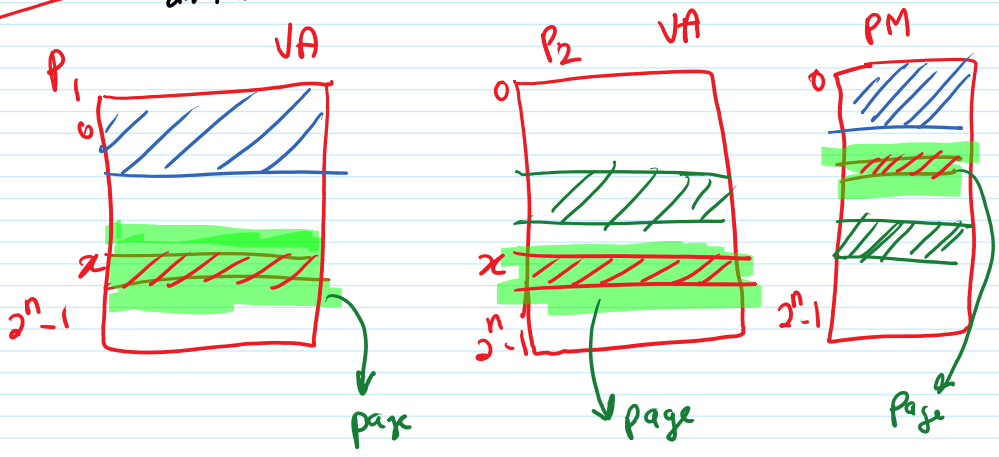
Idea 3

- Process could communicate through variable that are stored between them
  ↳ Shared variable
  Private variables



Problem ⇒ address translation to protect one process from another

OS provides mechanism



Idea 4

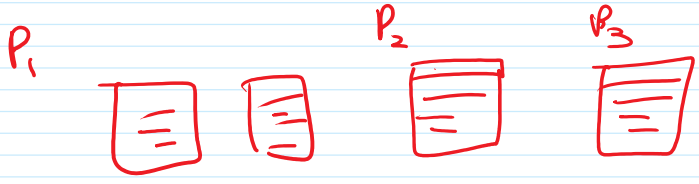
- Processes could possibly communicate by sending and receiving messages to each other.
  - ↳ OS provide mechanism

Idea 5

- Sometimes processes don't to explicitly share values to cooperate.

Example ⇒ synchronization

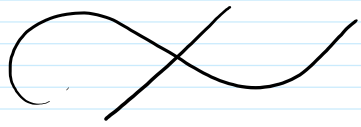
→ P_1 reads, 2 matrices → P_2 multiplies, 2 matrices → P_3 writes → a matrix



- Process 2 should not start work until Process 1 finishes reading
  - ↳ called process synchronization

- Synchronization primitives

  mutex lock, semaphore, barrier



← midterm up to here!

More Details on Idea #4

Program with shared variables

- Consider a 2 process program in which both processes increment a shared variable.

```
int X = 0;

P1                P2
X++;              X++;
{                 {
```

Question → What is the value of X? ⇒ 2!

Problem → X = 1 or 2!!

X++ ⇒ in instruction set

```
LDR   R1, [R2]
ADD   R1, R1, #1
STR   [R2], R1
```

} done in both P1 & P2

Why can X be 1?

↳ P1 loads X into R1, increment R1

↳ P2 load X into register before P1 stores new value into X.

* Need to synchronize processes that are interacting using shared variables.

* Critical Section ⇒ part of program where a shared variable is accessed

Critical Sections

- Must synchronize processes so that they access shared variables one at a time on a critical section
  - ↳ Mutual Exclusion

- Mutex Lock ⇒ sychronization primitives
  - Acquire Lock (L) ⇒ Done before a critical section code
    ⇒ Returns a value when safe for process to enter critical section
  - Release Lock (L)
    ⇒ Done after the critical section
    ⇒ Allows another process to acquire lock.

Implementing Lock

```
int L = 0;          // 0 ⇒ lock is available
                    // 1 ⇒ lock is in use.

AcquireLock(L):
    while (L==1);
    L = 1;

ReleaseLock(L)
    L = 0;
```