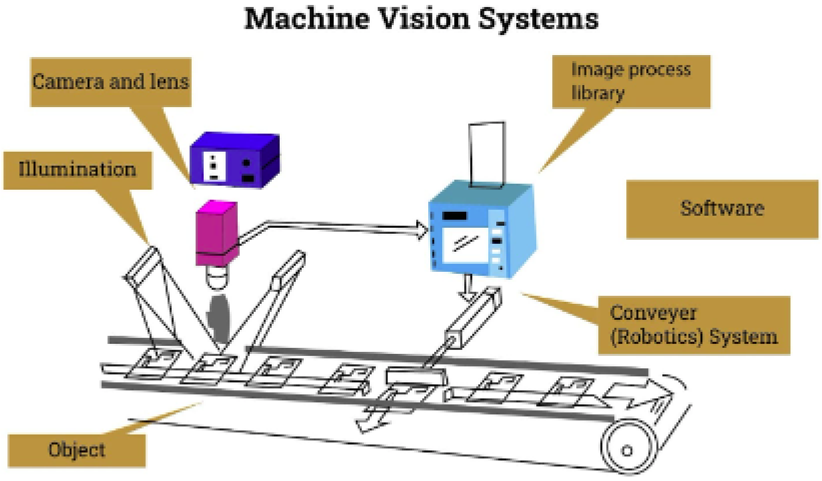
ARP 2022-2023 - SECOND ASSIGNMENT - general specs.

The code to design, develop, test and deploy is an interactive simulator of a (simplified) typical vision system, able to track an object in a 2-D plane. This is a true example of an industrial vision system in a robotic manufacturing workshop, like for example:



This assignment requires the use of a **shared memory** in which two processes operate simultaneously, as happens in reality in similar applications.

In our case we don't have a camera, so we will simulate the creation of the moving image using an *ncurses window*. Using arrow keys, we will move a spot in a window to simulate the perception of the camera. The spot that we will see by moving will produce the creation of a realistic RGB image (a circle, or a square, or similar) in the simulated, shared, video memory (see the next page).

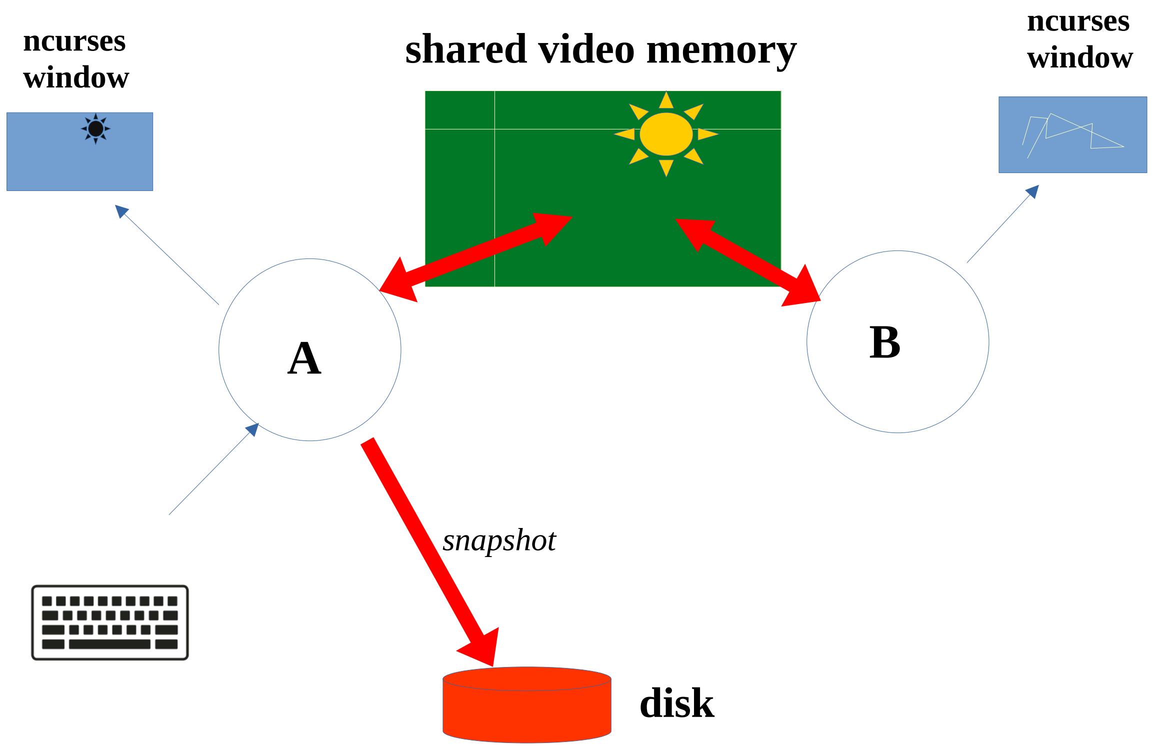
We will use an 80 x 30 *ncurses window*. The video memory, for realism, will be 1600 x 600 pixels *RGB true color 3 (3 bytes/pixel).* There will then be a factor of 20x in the location of the interactive spot and the corresponding image in the video memory.

The above will be performed by **process A**, which simulates capture from a video camera and fills the video memory.

A second **process B** will simulate the extraction of a feature from the acquired moving image instead. For simplicity, the image in the video memory will be scanned and the center of the image will be extracted. In a second *ncurses window*, also 80 x 30, the **position trace** of the center of the image will be shown.

There will be an additional function conducted by process A, useful for debugging. By pressing a key, or by operating the mouse on a button, a **snapshot** of the image memory will be saved on a *.bmp file*.

The structure of the code is as follows:

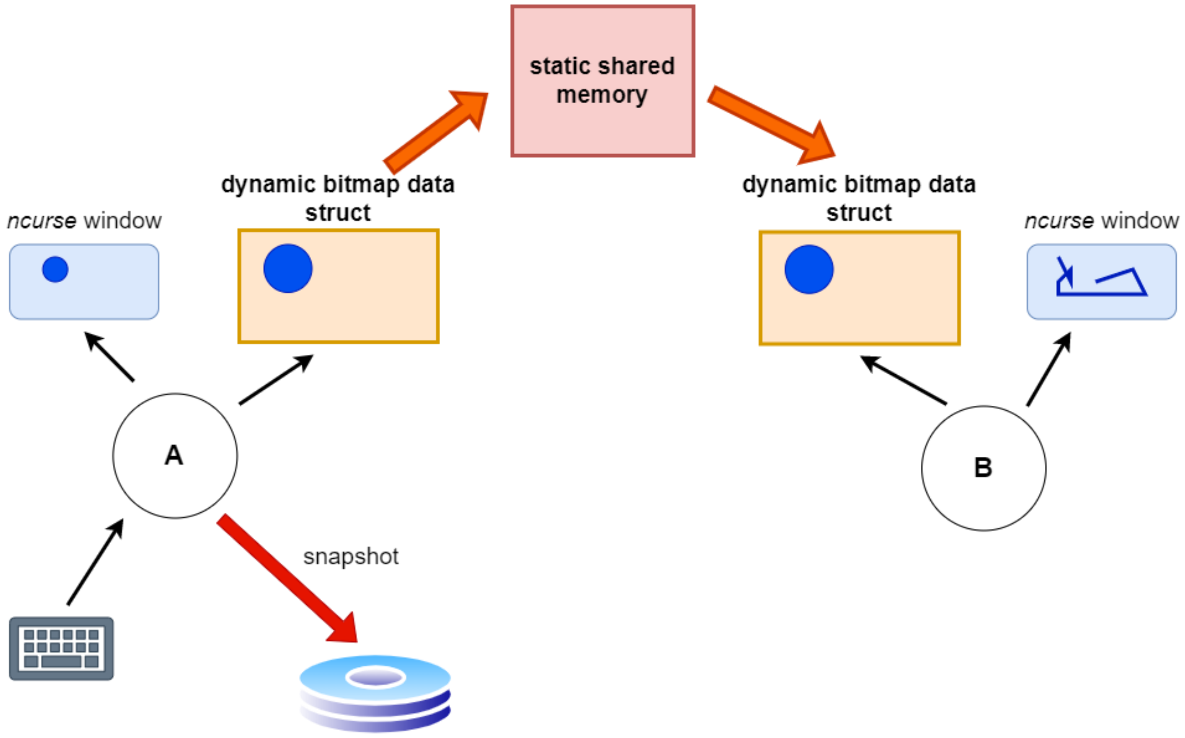


Two GitHub libraries will be used:
ncurses, already used in the first assignment
libbitmap (warning: it requires some hints not clearly explained in GitHub).

Core topics:
Posix shared memory
Posix semaphores
RGB images

ASSIGNMENT 2 UPDATE - 29/11/2022

After delving into the details of *libbitmap,* it turned out that the library makes use of **dynamic memory** (malloc) to allocate the space needed to represent the `bmpfile_t` data type. Unfortunately, **dynamic memory is**, by definition, **private to the process that allocates it**, therefore there is no possibility of sharing it between multiple processes.
Therefore, we propose the following variation of the software architecture to cope with this issue:



The two processes A and B, instead of sharing the bitmap representation, **will have their own private copy of the `bmpfile_t`** . However, **the processes must maintain a consistent representation of the overall bitmap, therefore they will exploit static shared memory to *continuously* (and *synchronously*) update their own copy of the `bmpfile_t`.**

Keeping this small adjustment in mind, the solution you are requested to implement remains exactly the same as in the original file.