\unprivileged instruction of changing the pointer to the stack -> since its only messing up the own space.

the threads implemented as the user-level threads, but with all the functions of kernel threads, and with the support of the kernel.

## introduction

threads: user-level && kernel-level

user-level threads(not built on the top of kernel threads): no modification on the kernel or process, takes process as a virtual processor. Flexible but sometimes are under the control over the processor performance influence. That is, the scheduling of these threads are indirect, the scheduling of their relied processes will largely influence them. OS only knows one thread although multiple threads may be contained in user space, so if there is page fault, these all threads will be blocked all. And since it’s like N:1, its not that suitable for multiprocessors.

kernel-level threads: the threads are directly scheduled by the kernel, however the thread management primitives have bad performance.

target in this work: provide a kernel interface and user-level threads package to provide:

* functionality(like kernel-level threads)
* performance(litke user-level threads)
* flexibility

why its hard: the scheduling and state information to achieve above is distributed in the kernel and the application’s address space.

solution:

application -> virtual multiprocessor

the kernel provides with every applications some processors, and can change the number

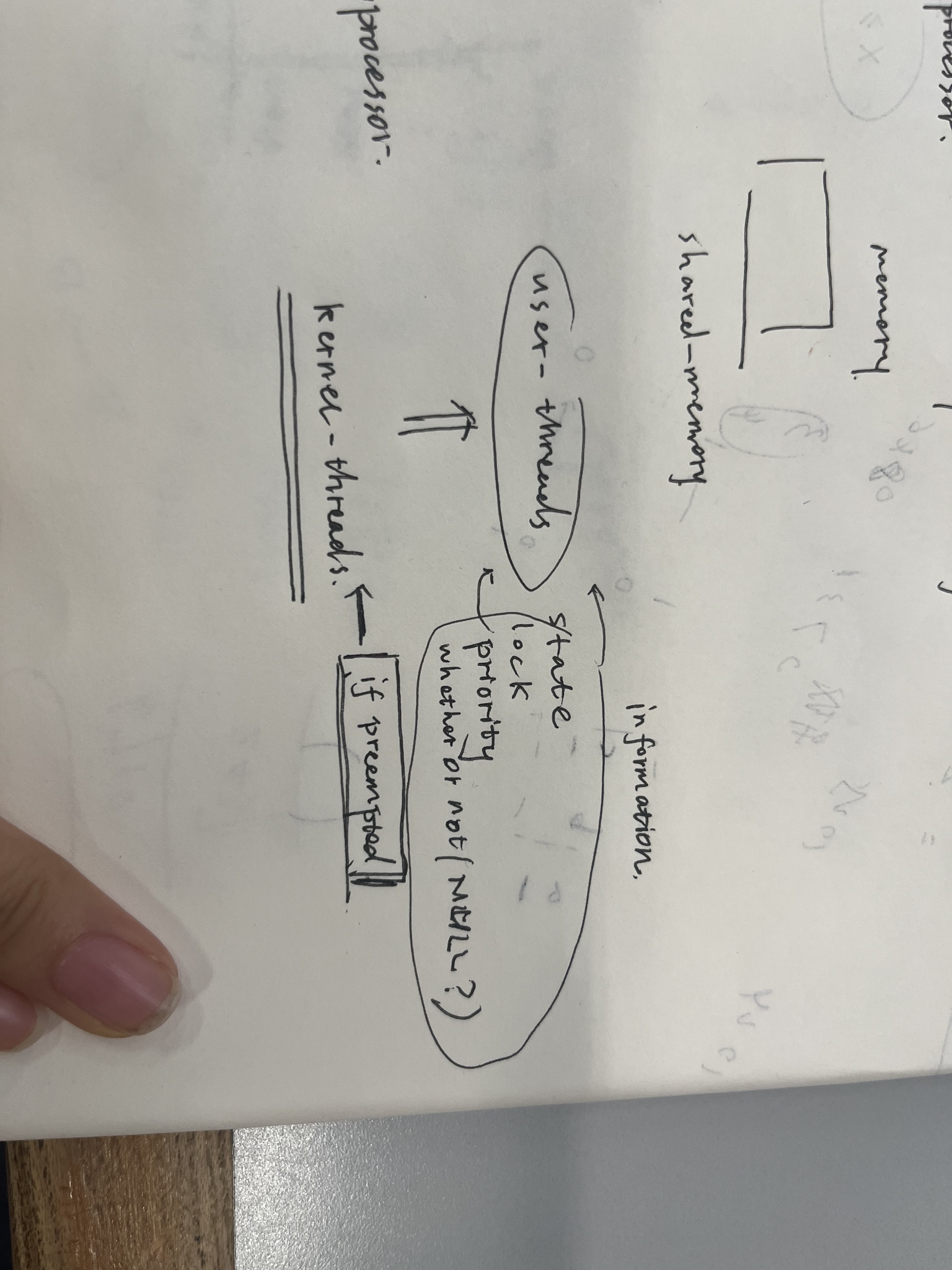
The application can decide which threads are running on the processors. Can tell the kernel about some events which influence the processor reallocation.

## user-level thread management

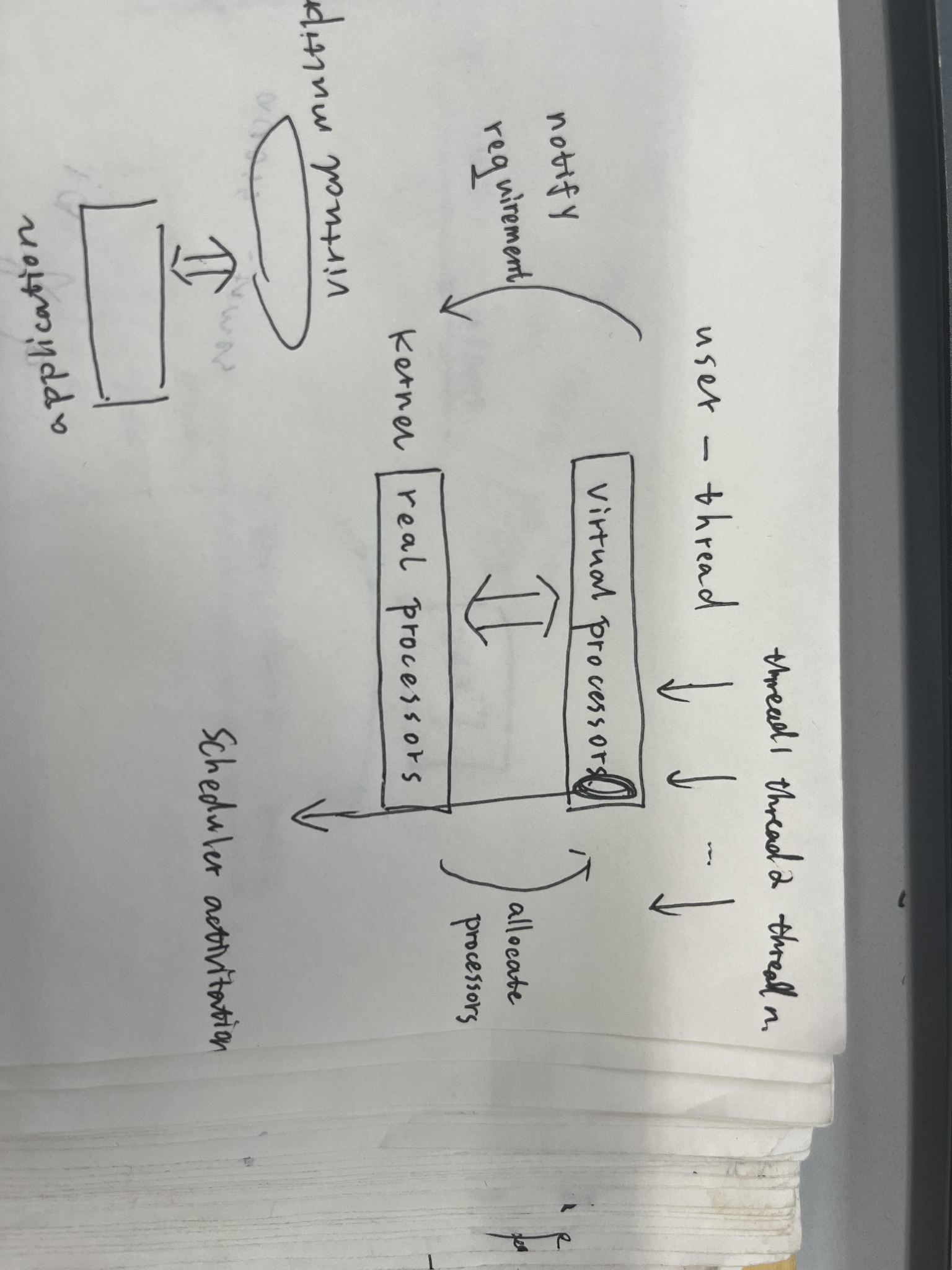
the drawbacks of the kernel-level threads

* unavoidable overhead for trapping into the kernel
* general features, which leads to overhead.
* the high-cost thread management

the drawbacks of user-level threads which built on the kernel threads:



## design and implementations of scheduler activations



what the threads notify the kernel: it has more threads than the (virtual) processors or it has more (virtual) processors than its threads. These requests do not promise that the application will get new threads, but signals.

\tip:

the upcalls -> sent from kernels to threads, but let the user space decide which thread to run in the user space.

the mechanism for protecting threads in critical sections: recovery -> the applications can do context switches when the kernel informs it that the thread has been preempted or unblocked.

Based on the scheduler activations, there are some ``policies``.

processor allocation policies:

* space-sharing. The processors are split evenly in the address-space, only with slices for not evenly divided.

thread scheduling policy: here FIFO, but can be modified by each address space.