The task is about rearranging the address space so as to place the stack at the high end. Currently, xv6 address space is set up like this:

code
stack (fixed-sized, one page)
heap (grows towards the high-end of the address space)

In this part of the xv6 project, you'll rearrange the address space to look more like this:

code
heap (grows towards the high-end of the address space)
... (gap)
stack (at end of address space; grows backwards)

Hints:

1. Figure out where xv6 allocates and initializes the user stack.
2. Figure out how to change that to use a page at the high-end of the xv6 user address space, instead of one between the code and heap.
3. Some tricky parts: one thing you'll have to be very careful with is how xv6 currently tracks the size of a process's address space (currently with the sz field in the proc struct). There are a number of places in the code where this is used (e.g., to check whether an argument passed into the kernel is valid; to copy the address space). We recommend keeping this field to track the size of the code and heap, but doing some other accounting to track the stack, and changing all relevant code (i.e., that used to deal with sz) to now work with your new accounting.
4. You should also be wary of growing your heap and overwriting your stack. In fact, you should always leave an unallocated (invalid) page between the stack and heap.
5. The high end of the xv6 user address space is 640KB (see the USERTOP value defined in the xv6 code). Thus your stack page should live at 636KB-640KB.
6. One final part of this project, which is challenging: automatically growing the stack backwards when needed. Doing so would require you to see if a fault occurred on the page above the stack and then, instead of killing the offending process, allocating a new page, mapping it into the address space, and continuing to run.

Test:

Testing your code requires making a new system call that shows you value of two pointers sz (for code and heap) and sp (for stack). Make a program in which you demand heap and stack memory and use your system call to show that sz and sp changes accordingly.