```
/**
 * replace - Replaces all occurrences of character 'old' in the given string wit
h character 'new'.
 *
 * This function creates a new string in memory, with all occurrences of the cha
racter 'old' in the input
 * string replaced with the character 'new'. The input string remains unchanged.
 The new string is created
 * dynamically in memory using the malloc() function. The function returns a poi
nter to the new string.
 *
 * @param x0: Address of the input string to be modified.
 * @param x1: Character to be replaced.
 * @param x2: Character to replace 'old' with.
 * @return x0: Pointer to the newly created string.
 *
 * Registers used: x0, x1, x2, x3, w4
 * Registers saved: lr
 */

 .global replace
    replace:
        // Save the link register and the input string pointer on the stack
        stp lr, x0, [sp, #-16]!

        // Save the 'old' and 'new' characters on the stack
        stp x1, x2, [sp, #-16]!

        // Get the length of the input string
        bl  length

        // Increment the length by 1 to make space for the null terminator
        add x0, x0, #1

        // Allocate memory for the new string using malloc
        bl  malloc

        // Restore the 'old' and 'new' characters from the stack
        ldp x2, x3, [sp], #16
        ldp lr, x1, [sp], #16
        stp lr, x0, [sp, #-16]!

    loop:
        // Load the next character from the input string
        ldrb w4, [x1], #1

        // If the character is null, we have reached the end of the string
        cmp  w4, #0
        b.eq end

        // If the character is the 'old' character, replace it with the 'new' ch
aracter
        cmp  w4, w2
        b.eq swap

        // Otherwise, copy the character to the new string
        strb w4, [x0], #1
        b loop

    swap:
        // Copy the 'new' character to the new string
        strb w3, [x0], #1
        b loop

    end:
        // Restore the input string pointer and the link register from the stack
        ldp lr, x0, [sp], #16
        ret
```