```cpp
// [...]

// SOLUTION 2019-06-12

/**
 * Sysmte global breakpoint descriptor struct.
 */
struct b_info
{
    /**
     * Wait queue for the first process which calls the breakpoint() primitive.
     * All other processes calling the breakpoint() primitive must be ignored.
     */
    struct proc_elem *waiting;

    // %rip
    natq rip;

    // original byte addressed by %rip
    natb orig;

  // system global breakpoint descriptor
} b_info;

/**
 * @param  rip  the address where the breakpoint should be placed.
 */
extern "C" void c_breakpoint(natq rip)
{
    // retrieve calling process descriptor
    struct des_proc *self = des_p(esecuzione->id);

    // check if there is already a process which called the breakpoint()
    // primitive and is waiting
    if (b_info.waiting)
    {
        // if so, set return value
        self->contesto[I_RAX] = 0xFFFFFFFF;

        // just return to the caller
        return;
        }

    // check if the given address belongs to the user process shared memory area
    if (rip < ini_utn_c || rip >= fin_utn_c)
    {
        // if not, print a warning log message
        flog(LOG_WARN, "rip %p out of bounds [%p, %p)", rip, ini_utn_p, fin_utn_p);

        // abort calling process
        c_abort_p();

        // return to the caller
        return;
        }

    // retrieve the first byte pointed by %rsp
    natb *bytes = reinterpret_cast<natb*>(rip);

    // save %rip
    b_info.rip = rip;

    // save original byte pointed by %rip
    b_info.orig = *bytes;

    // replace the original byte with the int3 opcode
    *bytes = 0xCC;

    // queue the calling process
    b_info.waiting = esecuzione;
    // schedule a new process
    schedulatore();
}

/**
 * Called when the breakpoint exception occurs.
 *
 * @param  tipo         interrupt type (3);
 * @param  errore       error code (0);
 * @param  p_saved_rip  content of %rip.
 */
extern "C" void c_breakpoint_exception(int tipo, natq errore, vaddr* p_saved_rip)
{
    // check if there is a process waiting in the system global breakpoint
    // descriptor wait queue
    if (!b_info.waiting || *p_saved_rip != b_info.rip + 1)
    {
        // if not, handle breakpoint exception: the calling process is aborted
        // in the gestore_eccezioni()
        gestore_eccezioni(tipo, errore, *p_saved_rip);

        // just return to the caller
        return;
    }

    // otherwise...

    // retrieve byt pointed by the value of %rip saved in the global breakpoint
    // descriptor
    natb *bytes = reinterpret_cast<natb*>(b_info.rip);

    // write the original byte back
    *bytes = b_info.orig;

    // decrease %rip for the calling process
    (*p_saved_rip)--;

    // retrieve process descriptor for the process in the wait queue of the
    // system global breakpoint descriptor
    des_proc *dest = des_p(b_info.waiting->id);

    // set return value for such process (which is the process that originally
    // called the breakpoint() primitive)
    dest->contesto[I_RAX] = esecuzione->id;

    // place the calling process in the system ready processes queue
    inspronti();

    // place the process which called the breakpoint() primitive in the system
    // ready processes queue
    inserimento_lista(pronti, b_info.waiting);

    // clear system global breakpoint descriptor wait queue (the breakpoint()
    // primitive can now be used by another process)
    b_info.waiting = 0;

    // schedule a new process
    schedulatore();
}

// SOLUTION 2019-06-12
```