```
// [...]

// EXTENSION 2019-07-03

// traduce l'indirizzo virtuale ind_virt nel corrispondente
// indirizzo fisico nello spazio virtuale del processo di
// identificatore id (il processo deve esistere)
extern "C" faddr trasforma(natl id, vaddr ind_virt)
{
    natq d;
    for (int liv = 4; liv > 0; liv--)
    {
        d = get_des(id, liv, ind_virt);

        if (!extr_P(d))
        {
            flog(LOG_WARN, "impossibile trasformare %lx: non presente a livello %d", ind_
virt, liv);
            return 0;
            }

        if (extr_PS(d))
        {
            // pagina di grandi dimensioni
            natq mask = (1UL << ((liv - 1) * 9 + 12)) - 1;
            return norm((d & ~mask) | (ind_virt & mask));
        }
    }

    return extr_IND_FISICO(d) | (ind_virt & 0xfff);
}

/**
 * System global breakpoint descriptor struct.
 */
struct b_info
{
    /**
     * Wait queue of the processes which called the bpwait() primitive and are
     * waiting for a process to reach the breakpoint address.
     */
    proc_elem *waiting;

    /**
     * Wait queue for all the process which have reached the breakpoint and
     * which IDs have not been yet retrieved using the bpwait() primitive.
     */
    proc_elem *intercepted;

    /**
     * Wait queue for all the processes which have reached the brakpoint and
     * which IDs have already been retrieved using the bpwait() and need to
     * be rescheduled.
     */
    proc_elem *to_wakeup;

    /**
     * Breakpoint virtual address.
     */
    vaddr rip;

    /**
     * Original byte in the replaced instruction.
     */
    natb orig;

    /**
     * True when there is a breakpoint installed.
     */
    bool busy;
```

```
  // system global breakpoint descriptor
} b_info;

/**
 * Adds a breakpoint at the given virtual address.
 *
 * @param  rip  the address where to add the breakpoint.
 */
extern "C" void c_bpadd(vaddr rip)
{
    // retrieve calling process descriptor
    des_proc *self = des_p(esecuzione->id);

    // check if there is a global system breakpoint installed
    if (b_info.busy)
    {
        // if so, return false: breakpoint already present
        self->contesto[I_RAX] = false;

        // 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)
    {
        // print a warning log message
        flog(LOG_WARN, "rip %p out of bounds [%p, %p)", rip, ini_utn_p, fin_utn_p);

        // abort the calling process
        c_abort_p();

        // just return to the caller
        return;
        }

    // retrieve byte address by the given virtual address
    natb *bytes = reinterpret_cast<natb*>(rip);

    // save the given virtual address for later use
    b_info.rip = rip;

    // save the original byte being replace for later use
    b_info.orig = *bytes;

    // replace the retrieved byte with the opcode of the int3 instruction
    *bytes = 0xCC;

    // set system global breakpoint descriptor busy flag to true: one breakpoint
    // is already present and no more will be accepted
    b_info.busy = true;

    // return true: breakpoint successfully placed
    self->contesto[I_RAX] = true;
}

/**
 *
 */
extern "C" void c_bpwait()
{
    // retrieve calling process descriptor
    des_proc *self = des_p(esecuzione->id);

    // check if there is a breakpoint already placed
    if (!b_info.busy)
    {
        // if not, return no breakpoints present
        self->contesto[I_RAX] = 0xFFFFFFFF;
```

```
        // just return to the caller
        return;
    }

    // check if there is any process which have rached the breakpoint address
    if (b_info.intercepted)
    {
        // if so, we need a process descriptor
        proc_elem *work;

        // remove one of such processes from the queue
        rimozione_lista(b_info.intercepted, work);

        // return the process id to the caller
        self->contesto[I_RAX] = work->id;

        // place the process in the wakup list
        inserimento_lista(b_info.to_wakeup, work);
    }
    else
    {
        // otherwise, place the calling process in the waiting queue
        inserimento_lista(b_info.waiting, esecuzione);

        // schedule a new process
        schedulatore();
    }
}
// EXTENSION 2019-07-03 )

// SOLUTION 2019-07-03

/**
 * Removes the breakpoint and reschedules all the process which had reached the
 * breakpoint address and were placed in the intercepted wait queue.
 */
extern "C" void c_bpremove()
{
    // check if there is any process which have called the bpwait()
    // or the busy flag is set
    if (b_info.waiting || !b_info.busy)
    {
        // if not, no breakpoint can be removed either because there is none or
        // because there are process waiting
        flog(LOG_WARN, "Unable to perform bpremove().");

        // abort calling process
        c_abort_p();

        // just return to the caller
        return;
    }

    // retrieve address to the replaced byte
    natb *bytes = reinterpret_cast<natb*>(b_info.rip);

    // replace the byte with the original value
    *bytes = b_info.orig;

    // process descriptor
    proc_elem *work;

    // while there are processes which have reached the breakpoint address
    while (b_info.intercepted)
    {
        // remove them from the intercepted queue
        rimozione_lista(b_info.intercepted, work);

        // place them in the wakeup queue
```

```
        inserimento_lista(b_info.to_wakeup, work);
        }

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

    // while there are processes which have reached the breakpoint address and
    // need to be rescheduled
    while (b_info.to_wakeup)
    {
        // retrieve next process to wake up
        rimozione_lista(b_info.to_wakeup, work);

        // retrieve process descriptor
        des_proc *dp = des_p(work->id);

        // retrieve process virtual %rsp value
        natq rsp_v = dp->contesto[I_RSP];

        // retrieve physical address of %rsp
        natq *rsp = reinterpret_cast<natq*>(trasforma(work->id, rsp_v));

        // decrease it of one
        (*rsp)--;

        // place the process in the system ready processes list
        inserimento_lista(pronti, work);
    }

    //
    b_info.busy = false;

    // schedule a new process
    schedulatore();
}

/**
 * Called when a breakpoint exception occurs.
 *
 * @param  tipo    interrupt type (3);
 * @param  errore  error type (0);
 * @param  rip     current value address by %rsp.
 */
extern "C" void c_breakpoint_exception(int tipo, natq errore, vaddr rip)
{
    // check if there is any breakpoint in the system global breakpoint
    // descriptor
    if (!b_info.busy || rip != b_info.rip + 1)
    {
        // if not, the bpadd() primitive was not used: handle the exception and
        // abort the calling process
        gestore_eccezioni(tipo, errore, rip);

        // just return to the caller
        return;
    }

    // check if there is any process waiting for a breakpoint
    if (b_info.waiting)
    {
        // if so, we need to notify such processes that an external process has
        // reached the breakpoint
        proc_elem *work;

        // retrieve such process proc_elem
        rimozione_lista(b_info.waiting, work);

        // retrieve process descriptor
        des_proc *dp = des_p(work->id);
```

```
        // notify the waiting process that the current process in execution has
        // reached the breakpoint address
        dp->contesto[I_RAX] = esecuzione->id;

        // place the current process in the breakpoint descriptor to_wakeup
        // queue
        inserimento_lista(b_info.to_wakeup, esecuzione);

        // insert the waiting process in the system ready processes queue
        inserimento_lista(pronti, work);
        }
    else
    {
        // otherwise, just place the current process under execution in the
        // intercepted processes queue to wait for a process to call the
        // bpwait() primitive
        inserimento_lista(b_info.intercepted, esecuzione);
    }

    // schedule a new process
    schedulatore();
}

// SOLUTION 2019-07-03
```