```
/**
 * Process descriptor. We must edit this struct and add the listen_reg boolean
 * field which is set to true when a process registers as a listener of the
 * global broadcast descriptor. For simplicity we will assume that for the whole
 * system execution only one process at a time will try to send one and only one
 * broadcast message and that the listen primitive() will be called by each
 * listener process one and only one time.
 */
struct des_proc
{
    // parte richiesta dall'hardware
    struct __attribute__ ((packed))
    {
        natl riservato1;
        vaddr punt_nucleo;
        // due quad  a disposizione (puntatori alle pile ring 1 e 2)
        natq disp1[2];
        natq riservato2;
        //entry della IST, non usata
        natq disp2[7];
        natq riservato3;
        natw riservato4;
        natw iomap_base; // si veda crea_processo()
        };

    //finiti i campi obbligatori
    faddr cr3;

    natq contesto[N_REG];

    natl cpl;

// EXTENSION 2016-09-20

    // true if the process is registered to the global broadcast descriptor
        bool listen_reg;

// EXTENSION 2016-09-20
};

// EXTENSION 2016-09-20

/**
 * Global broadcast descriptor struct.
 */
struct broadcast
{
    // number of registered listener processes
    int registered;

    // number of registered processes which have already called the listen()
    // primitive
    int nlisten;

    // last broadcast message sent
    natl msg;

    // processes waiting for the next broadcast message queue: processes which
    // have called the listen() primitive
    proc_elem *listeners;

    // broadcaster wait queue: the broadcaster process waits here until all
    // registered listener processes have called the listen() primitive
    proc_elem *broadcaster;
};

/**
 * Global broadcast descriptor. For simplicity we will have one single system
 * wide broadcast descriptor which can be used to broadcast messages. Also, we
 * are assuming that for the whole system execution one and only one process
```

```
 * will broadcast one and only one message. No listener process will call the
 * listen() primitive more than once.
 */
broadcast global_broadcast =
{
    0,  // registered
    0,  // nlisten
    0,  // msg
    0,  // listeners
    0   // broadcaster
};

/**
 * Implementation for the void reg() primitive. If the processes calling this
 * method is already registered to the global broadcast no action will be
 * performed.
 */
extern "C" void c_reg()
{
    // retrieve calling process descriptor, [0]
    struct des_proc *p = des_p(esecuzione->id);

    // retrieve global broadcast descriptor
    struct broadcast *b = &global_broadcast;

    // check if the process is already a registered listener
    if (p->listen_reg)
    {
        // if so, just return: nothing to do
        return;
    }

    // otherwise, register the process to the listeners
    b->registered++;

    // set the process broadcast listener flag: this will be checked when the
    // process calls the listen() primitive
    p->listen_reg = true;
}
// EXTENSION 2016-09-20

// SOLUTION 2016-09-20

/**
 * Called when all registered listeners are ready to receive the broadcast
 * message. This happens when registered == nlisten for the global broadcast.
 * Sends the current broadcast message to all waiting processes.
 */
void broadcast_all()
{
    // retrieve pointer to the global broadcast descriptor
    struct broadcast *b = &global_broadcast;

    // process descriptor
    struct proc_elem *work;

    // while there are still listener processes in the queue
    while (b->listeners)
    {
        // remove top process from listeners wait queue
        rimozione_lista(b->listeners, work);

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

        // deliver broadcast message
        w->contesto[I_RAX] = b->msg;

        // place process in the system ready processes queue
        inserimento_lista(pronti, work);
```

```
    }

    // all process have received the broadcast message
    b->nlisten = 0;
}

/**
 * All registered listeners must call this method to receive the broadcast
 * message. If the broadcaster has already sent the broadcast message this
 * function will deliver it to the calling listener and remove it from the
 * listeners processes queue and check if all registered listeners have called
 * the listen() primitive. If so it will call the broadcast_all method.
 * Otherwise it will insert the current process in the system ready processes
 * queue and wait for the next listener to call the listen() primitive.
 */
extern "C" void c_listen()
{
    // retrieve calling process descriptor, [0]
    struct des_proc *p = des_p(esecuzione->id);

    // retrieve global broadcast descriptor
    struct broadcast *b = &global_broadcast;

    // check if the calling process is registered as listener
    if (!p->listen_reg)
    {
        // print warning log message
        flog(LOG_WARN, "Process not registered as broadcast listener.");

        // abort current process under execution
        c_abort_p();

        // just return to the caller
        return;
    }

    // increase number of listeners awaiting broadcast message
    b->nlisten++;

    // if there is no process in the broadcaster queue yet: broadcast(natl msg)
    // not called yet
    if (!b->broadcaster)
    {
        // insert the process in the global broadcast listeners wait queue: a
        // new process is scheduled at the end
        inserimento_lista(b->listeners, esecuzione);
        }
    else
    {
        // otherwise, deliver the message to the current listener process
        p->contesto[I_RAX] = b->msg;

        // insert current process in the system ready processes list
        inserimento_lista(pronti, esecuzione);

        // check if all listener processes have called the listen() primitive
        if (b->nlisten == b->registered)
        {
            // if so, deliver broadcast message to all listener processes
            broadcast_all();

            // after that,we need a process descriptor
            struct proc_elem *work;

            // to retrieve the broadcaster process
            rimozione_lista(b->broadcaster, work);

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

```
        // if all the listener process have not called the listen() primitive we
        // will have to wait for the next listener process calling it and check
        // again if all listener processes are ready, deliver the broadcast
        // message to all of them and remove the broadcaster process from the
        // queue in order for it to be rescheduled
    }

    // schedule a new process
    schedulatore();
}

/**
 * Called by the broadcaster process to send the given broadcast message to all
 * registered listener processes. If all registered listener processes have
 * called the listen() primitive the broadcast_all method will deliver the
 * broadcast message to all of them. Otherwise, the broadcaster process will be
 * inserted in the broadcasyer wait queue waiting for all listeners to be ready.
 *
 * @param  msg  the message to be broadcasted.
 */
extern "C" void c_broadcast(natl msg)
{
    // retrieve calling process descriptor
    struct des_proc *p = des_p(esecuzione->id);

    // retrieve global broadcast descriptor
    struct broadcast *b = &global_broadcast;

    // check if the process is not registered as listener
    if (p->listen_reg)
    {
        // print warning log message
        flog(LOG_WARN, "Listener process can not send broadcast messages.");

        // abort current process under execution
        c_abort_p();

        // return to the caller
        return;
    }

    // set broadcast message
    b->msg = msg;

    // check if all listeners have invoked the listen primitive
    if (b->nlisten == b->registered)
    {
        // if so, insert the current process under execution (the broadcaster)
        // in the system ready processes queue: a new process is scheduled at
        // the end
        inserimento_lista(pronti, esecuzione);

        // send broadcast message to all listeners
        broadcast_all();
        }
    else
    {
        // otherwise, wait for all listeners to be ready
        // insert current process in the broadcaster process queue
        inserimento_lista(b->broadcaster, esecuzione);
    }

    // schedule a new process
    schedulatore();
}

// SOLUTION 2016-09-20

// [...]
```

```
/**
 * The method used to destroy processes must be edited in order to check if the
 * process being destructed is a registered listener and if so remove it from
 * the global broadcast listeners.
 */
void distruggi_processo(proc_elem* p)
{
        des_proc* pdes_proc = des_p(p->id);

// EXTENSION 2016-09-20

    // check if the process is a registered listener
    if (pdes_proc->listen_reg)
    {
        // in that case, decrease registered processes
        global_broadcast.registered--;
    }

// EXTENSION 2016-09-20
        faddr tab4 = pdes_proc->cr3;
        riassegna_tutto(p->id, tab4, I_MIO_C, N_MIO_C);
        riassegna_tutto(p->id, tab4, I_UTN_C, N_UTN_C);
        rilascia_tutto(tab4, I_UTN_P, N_UTN_P);
        ultimo_terminato = tab4;
        if (p != esecuzione) {
                distruggi_pila_precedente();
        }
        rilascia_tss(id_to_tss(p->id));
        dealloca(pdes_proc);
}
```