# 527 — Computer Networks and Distributed Systems — Tutorial 1: Message Passing

E.C. Lupu

*Note that the solution notes below only briefly list (some of) the key points that should be included in an answer. They are by no means complete. In an exam, you are expected to spell out the solution more fully and include a detailed explanation of your reasoning.*

## 1 Messaging Primitives

The following message passing primitives are supported by a set of library calls:

**send**`(dest, msg)` — an *asynchronous* send message primitive, where $dest$ is the name of the process to which the message $msg$ is to be sent.

**receive**`(source, msg)` — this causes the receiving process to block waiting for a message from the process with name $source$. $msg$ is a buffer into which the incoming message is copied.

**receiveany**`(source, msg)` — the process is blocked waiting for a message from any source. The name of the sender is received in $source$ and the incoming message is received in $msg$.

a) Explain what is meant by an *asynchronous* send message primitive and why it may lead to buffer exhaustion at the receiver.
**Solution Notes:**
Asynchronous send: this is an unblocked send in which the sender process sends the message and continues once the message has been copied out of its address space. It does not know when or if the message is received by the destination. One particular sender ( or multiple processes) may send messages to a receiver at a rate faster than the receiver can process the messages. Each message has to be buffered at the receiver while waiting to be processed this can lead to buffer overflow if the receiver cannot process messages fast enough as there will be a finite number of buffers.

b) Explain why both the above *receive* and *receiveany* primitives are needed.
**Solution Notes:**
The receive permits the receiver to selectively receive messages from a single source. A server process such as a file server does not know which sources will be sending requests so needs the $receiveany$ to be able to receive a message from any source.

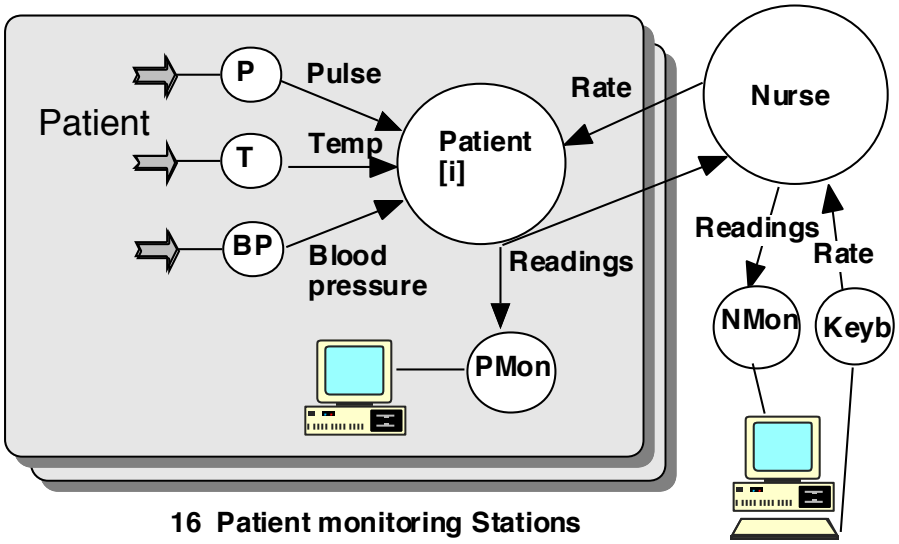
## 2 A Patient Monitoring System

The intensive care ward in a hospital consists of 16 beds. Patients in each bed are continuously monitored for a number of factors, such as pulse, temperature and blood pressure by means of sensors attached to their body and connected to a bedside computer, which displays current readings. A central nurse station displays the current reading for all beds and can also be used to set the rate of monitoring for each patient.

a) Outline the structure of the patient monitoring system, indicating your breakdown into software components (processes). Indicate the function of each component, the data flows between components in terms of the data types transferred.

A process component should be provided to handle each device (eg. sensor, display) and provide a message passing interface to the rest of the system. Do not try to minimise the number of processes; this can always be done later if necessary.

**Solution Notes:**




```
//Data
short pulse;
short blood_pressure;
float temp;
short rate;     //readings per min
struct readings {
        pulse p; blood_pressure bp; temp t; short bed}
```

P, T, BP could be polled or event driven. Polled is simpler as no timing required

b) For each main process component, outline the code (i.e. pseudocode) that you would provide to perform its function. Assume that the following communication primitives are available:

```
SEND message TO destination
        // Send a message and continue processing  (asynchronous)
RECEIVE message FROM source DELAY (n)
        // Receive message if available from source, otherwise wait
        // for up to n seconds n= -1 implies no timeout
source = RECEIVEANY message (DELAY n)
        // Receive a message from any source, otherwise wait for up to n seconds\\
```

**Solution Notes:**

Simplifed outline Processes. Assume every message includes a type field.

```
Process Patient {
        short delaytime = 5

        loop {
                receive msg from nurse delay (delaytime)
                case msg type
                rate: delaytime = 60/rate;
                null:  //timeout
                        send query to P;  receive pulse from P;
                        send query to T;  receive temp from T;
                        send query to BP;  receive press from BP;
                        set up readings message
                        send readings to PMon and Nurse
        }
}

Process Nurse {
        loop {
                source = receiveany msg delay (-1)
                case source
                        patient[i]: send msg reading to Nmon
                        Keyb:  send msg rate to patient[msg.bed]
        }
}
```

Other processes are left as an exercise for you!

# 3 Printer Service

Using the message primitives described in Question 1, design a simple printer service for a distributed system with multiple printers, each controlled by a process called *printer*. There are also multiple users. When a *user* process wants to print a document it sends a message containing its process type (i.e. user) to a single *coordinator* process which allocates free printers. The coordinator replies with the name of a free printer when one is available, and the user send a page at a time to the printer. When a printer is free it sends a message containing its process type (i.e. printer) to the coordinator to indicate it is now available for printing.

Give pseudocode outlines for the *user*, *coordinator* and *printer* processes, using the the message primitives described in Question 1. Your solution should describe any data structures needed by the coordinator process.

Assume the printer process has sufficient buffer space for a single message containing one page to print, and that communication is reliable so timeouts and retransmissions can be ignored.

**Solution Notes:**

```
Process User:    // when ready to print
        send (coordinator, userName, userProc)
        receive (coordinator, pn)
                        // get name (pn) of free printer
        loop
                send (pn, nextpage)
                receive(pn, ack)
        until EOF  //end of file
        send (pn, EOF)


Process Coordinator
        procnames:   a linked list of process names
        //either user processes waiting for a printer or available printers
        // assume 2 procedures addtail (name), removehead (name)
        printersavailable:  Boolean  // true indicates   printers on procnames

        set procnames to null
        printersavailable := false
        loop {
                receiveany (source, proctype)
                if (proctype = printer) {  // printer request
                if  (printersavailable = false) & (procnames != null) {
                                removehead (name)  //users  Q  d
                                send (source, name)  //send UserNname to printer
                                send (name, source) // send printerName to 1st user on Q
                        }
                else {  addtail (source) ; printersavailable := true}
                                // no users waiting so Q printer
                        }
                if ((proctype != printer) & (printersavailable = true)) {
                                //      message from user
                        removehead (name)  // remove printer from Q
                        if ((procnames == null)) {printersavailable := false;}
                        send (source, name) // send printerName to user  }
                        send (name, source) // send userName to 1st printer on Q }
                if ((proctype != printer) && (printersavailable != true)) // message from user
                {addtail (source)  }  // Q user
        }

Process Printer
        loop {
                send (coordinator, printerName, printerProc)   //printer available
                receive ( coordinator, name)  //get user name
                loop {
                        receive (name, msg)
                        print (msg)
                        send (name, ack)
                } until msg = eof
        }
```