The Structuring of Systems Using Upcalls

David D. Clark

Presenter: Haitham Gad
Layered Program Structure

- Layering is a program structure proposed for usage in operating systems and network protocols.
- System functions are organized as layers.
- Upper layers are thought of as clients to lower layers (lower layers provide services to upper layers).
- Acyclic dependency relationship among layers enhances modularity and facilitates verification.
Layered Program Structure

- Layers are traditionally implemented in the form of OS processes (or threads).
- Service invocation occurs from the top down through subroutine calls (probably RPC).
- Upward flow of control, when needed, is implemented using interprocess communication signals.
- This methodology, however, leads to serious performance and code complexity problems.
Motivation for Upcalls

- In a layered program structure, the natural flow of control is often upward not downward.

- In a process-based layer implementation, upward flow of control is inefficient due to:
  - The need for data buffering mechanisms;
  - The need for interprocess signals.

- Upward flow of control requires contaminating layer code with a lot of communication code.
Multi-task Modules

- A layer is organized as subroutines that live in a number of tasks.
- Invocation across layer boundaries is done only through subroutine calls.
- Shared state among subroutines is managed by a monitor lock.
- The group of collaborating subroutines from different tasks are called a multi-task module.
Multi-task Modules

- Intertask communication can only be used within a layer.

Figure 1: Illustration of System Organization
Example: Remote Login Service (Reception)

2a. The display or remote login layer

```plaintext
display-start():
    local-port = transport-open(display-receive)
end

display-receive(char):
    write char to display
end

2b. The transport layer

```plaintext
transport-open(receive-handler):
    local-port = net-open(transport-receive)
    transport-handler-array(local-port)=
        receive-handler
    return(local port)
end

transport-get-port(packet):
    //upcalled by interrupt layer
    extract port from packet
    return(port)
end

transport-receive(packet,port):
    //upcalled by net-layer
    handler = transport-handler-array(port)
    validate packet header
    for each char in packet do handler(char)
end
```

2c. The network layer

```plaintext
net-open(receive-handler):
    port = generate-uid()
    task-id = create-task (net-receive
        (port, receive-handler))
    net-task-array(port) = task-id
    return(port)
end

net-receive(port,handler):
    handler = net-handler-array(port)
    do forever
        remove packet from per port queue
        handler(packet,port)
        block()
end
end

net-dispatch(): //upcalled by interrupt handler
    read packet from device
    restart device
    port=transport-get-port(packet)
    put packet on per port queue
    task-id = net-task-array(port)
    wakeup-task(task-id)
end
```
Example: Remote Login Service (Reception)

2a. The display or remote login layer

```
display-start():
   local-port = transport-open(display-receive)
end

display-receive(char):
   write char to display
end
```

2b. The transport layer

```
transport-open(receive-handler):
   local-port = net-open(transport-receive)
   transport-handler-array(local-port) = receive-handler
   return(local port)
end

transport-get-port(packet):
   //upcalled by interrupt layer
   extract port from packet
   return(port)
end

transport-receive(packet,port):
   //upcalled by net-layer
   handler = transport-handler-array(port)
   validate packet header
   for each char in packet do handler(char)
```

2c. The network layer

```
net-open(receive-handler):
   port = generate-uid()
   task-id = create-task (net-receive (port,receive-handler))
   net-task-array(port) = task-id
   return(port)
end

net-receive(port,handler):
   handler = net-handler-array(port)
   do forever
      remove packet from port queue
      handler(packet,port)
      block()
   end
end

net-dispatch(): //upcalled by interrupt handler
   read packet from device
   restart device
   port=transport-get-port(packet)
   put packet on port queue
   task-id = net-task-array(port)
   wakeup-task(task-id)
end
```

Downcall
Example: Remote Login Service (Reception)

2a. The display or remote login layer

display-start():
  local-port = transport-open(display-receive)
end
display-receive(char):
  write char to display
end

2b. The transport layer

transport-open(receive-handler):
  local-port = net-open(transport-receive)
  transport-handler-array(local-port) = receive-handler
  return(local-port)
end
transport-get-port(packet):
  //upcalled by interrupt layer
  extract port from packet
  return(port)
end
transport-receive(packet,port):
  //upcalled by net-layer
  handler = transport-handler-array(port)
  validate packet header
  for each char in packet do handler(char)

2c. The network layer

net-open(receive-handler):
  port = generate-uid()
  task-id = create-task (net-receive (port, receive-handler))
  net-task-array(port) = task-id
  return(port)
end

net-receive(port, handler):
  handler = net-handler-array(port)
  do forever
    remove packet from per port queue
    handler(packet, port)
    block()
  end
end

net-dispatch(): //upcalled by interrupt handler
  read packet from device
  restart device
  port = transport-get-port(packet)
  put packet on per port queue
  task-id = net-task-array(port)
  wakeup-task(task-id)
end

Downcall
Example: Remote Login Service (Reception)

2a. The display or remote login layer

```
display-start():
  local-port = transport-open(display-receive)
end

display-receive(char):
  write char to display
end

2b. The transport layer

transport-open(receive-handler):
  local-port = net-open(transport-receive)
  transport-handler-array(local-port) = receive-handler
  return(local-port)
end

transport-get-port(packet):
  //upcalled by interrupt layer
  extract port from packet
  return(port)
end

transport-receive(packet, port):
  //upcalled by net-layer
  handler = transport-handler-array(port)
  validate packet header
  for each char in packet do handler(char)

2c. The network layer

net-open(receive-handler):
  port = generate-uid()
  task-id = create-task (net-receive
    (port, receive-handler))
  net-task-array(port) = task-id
  return(port)
end

net-receive(port, handler):
  handler = net-handler-array(port)
  do forever
    remove packet from per port queue
    handler(packet, port)
    block()
  end
end

net-dispatch(): //upcalled by interrupt handler
  read packet from device
  restart device
  port = transport-get-port(packet)
  put packet on per port queue
  task-id = net-task-array(port)
  wakeup-task(task-id)
end

```

Intra-layer Signal
Example: Remote Login Service (Reception)

2a. The display or remote login layer

```c
transport-open(receive-handler):
  local-port = transport-open(display-receive)
end

display-receive(char):
  write char to display
end

2b. The transport layer

transport-get-port(packet):
  //upcalled by interrupt layer
  extract port from packet
  return(port)
end

transport-receive(packet,port):
  //upcalled by net-layer
  handler = transport-handler-array(port)
  validate packet header
  for each char in packet do handler(char)

2c. The network layer

net-open(receive-handler):
  port = generate-uid()
  task-id = create-task (net-receive
    (port,receive-handler))
  net-task-array(port) = task-id
  return(port)
end

net-receive(port,handler):
  handler = net-handler-array(port)
  do forever
    remove packet from per port queue
    handler(packet,port)
    block()
  end
end

net-dispatch(): //upcalled by interrupt handler
  read packet from device
  restart device
  port=transport-get-port(packet)
  put packet on per port queue
  task-id = net-task-array(port)
  wakeup-task(task-id)
end
```

Upcall
Example: Remote Login Service (Reception)

2a. The display or remote login layer

```plaintext
display-start():
   local-port = transport-open(display-receive)
end

display-receive(char):
   write char to display
end

2b. The transport layer

transport-open(receive-handler):
   local-port = net-open(transport-receive)
   transport-handler-array(local-port)=
      receive-handler
   return(local port)
end

transport-get-port(packet):
   //upcalled by interrupt layer
   extract port from packet
   return(port)
end

transport-receive(packet,port):
   //upcalled by net-layer
   handler = transport-handler-array(port)
   validate packet header
   for each char in packet do handler(char)
end

2c. The network layer

net-open(receive-handler):
   port = generate-uid()
   task-id = create-task (net-receive
      (port, receive-handler))
   net-task-array(port) = task-id
   return(port)
end

net-receive(port, handler):
   handler = net-handler-array(port)
   do forever
      remove packet from per port queue
      handler(packet, port)
      block()
   end

net-dispatch(): //upcalled by interrupt handler
   read packet from device
   restart device
   port=transport-get-port(packet)
   put packet on per port queue
   task-id = net-task-array(port)
   wakeup-task(task-id)
end
```

Intra-layer Signal
Example: Remote Login Service (Reception)

2a. The display or remote login layer

```plaintext
display-start():
    local-port = transport-open(display-receive)
end

display-receive(char):
    write char to display
end
```

2b. The transport layer

```plaintext
transport-open(receive-handler):
    local-port = net-open(transport-receive)
    transport-handler-array(local-port)=
        receive-handler
    return(local-port)
end

transport-get-port(packet):
    //upcalled by interrupt layer
    extract port from packet
    return(port)
end

transport-receive(packet, port):
    //upcalled by net-layer
    handler = transport-handler-array(port)
    validate packet header
    for each char in packet do handler(char)
end
```

2c. The network layer

```plaintext
net-open(receive-handler):
    port = generate-uid()
    task-id = create-task (net-receive
    (port, receive-handler))
    net-task-array(port) = task-id
    return(port)
end

net-receive(port, handler):
    handler = net-handler-array(port)
    do forever
        remove packet from per port queue
        handler(packet, port)
        block()
    end
end

net-dispatch(): //upcalled by interrupt handler
    read packet from device
    restart device
    port=transport-get-port(packet)
    put packet on per port queue
    task-id = net-task-array(port)
    wakeup-task(task-id)
end
```

Upcall
Example: Remote Login Service (Reception)

2a. The display or remote login layer

```
display-start():
  local-port = transport-open(display-receive)
end

display-receive(char):
  write char to display
end
```

2b. The transport layer

```
transport-open(receive-handler):
  local-port = net-open(transport-receive)
  transport-handler-array(local-port) = receive-handler
  return(local-port)
end

transport-get-port(packet):
  // upcalled by interrupt layer
  extract port from packet
  return(port)
end

transport-receive(packet, port):
  // upcalled by net-layer
  handler = transport-handler-array(port)
  validate packet header
  for each char in packet do handler(char)
end
```

2c. The network layer

```
ett-open(receive-handler):
  port = generate-uid()
  task-id = create-task (net-receive
    (port, receive-handler))
  net-task-array(port) = task-id
  return(port)
end

nett-receive(port, handler):
  handler = net-handler-array(port)
  do forever
    remove packet from port queue
    handler(packet, port)
    block()
  end
end

nett-dispatch(): // upcalled by interrupt handler
  read packet from device
  restart device
  port = transport-get-port(packet)
  put packet on port queue
  task-id = net-task-array(port)
  wakeup-task(task-id)
end
```

Upcall
Example: Remote Login Service (Sending)

display-keyboard-handler():
  //upcalled by interrupt handler for keyboard
got character from keyboard device
  and put in keyboard-buffer
transport-arm-for-send
  (port, display-get-data)
end

display-get-data (packet):
  //upcalled by to send data
  copy data from keyboard-buffer into packet
end

net-send(packet,port):
  start net device to send packet
end

transport-arm-for-send (port, send-handler):
  transport-send-handler-array (port)=
  send-handler
  if ok-to-send(port)
    then wakeup-task(send-task-id)
  else want-to-send(port)=false
end

transport-send(port):
  //runs in task identified by send-task-id
  if ok-to-send(port)=false then block()
allocate packet and fill in headers
send-handler=
  transport-send-handler-array(port)
send-handler(packet)
  //upcall display level to get data
net-send(packet,port)
ok-to-send(port)=false
want-to-send(port)=false
end

transport-receive(packet,port):
  //upcalled by net layer
handler=transport-handler-array(port)
validate packet header
if packet authorizes sending then
  if want-to-send(port)
    then wakeup-task(send-task-id)
  else ok-to-send(port)=true
  for each char in packet do handler(char)
end
Example: Remote Login Service (Sending)

display-keyboard-handler():
//upcalled by interrupt handler for keyboard
get character from keyboard device
and put in keyboard-buffer
transport-arm-for-send
(port, display-get-data)
end

display-get-data (packet):
//upcalled by to send data
copy data from keyboard-buffer into packet
end

net-send(packet, port):
//start net device to send packet
end

transport-arm-for-send (port, send-handler):
transport-send-handler-array (port)=
send-handler
if ok-to-send(port)
then wake-up-task(send-task-id)
else want-to-send(port)=true
end

transport-send(port):
//runs in task identified by send-task-id
if ok-to-send(port)=false then block()
allocate packet and fill in headers
send-handler=
transport-send-handler-array(port)
send-handler(packet)
//upcall display level to get data
net-send(packet, port)
ok-to-send(port)=false
want-to-send(port)=false
end

transport-receive(packet, port):
//upcalled by net layer
handler=transport-handler-array(port)
validate packet header
if packet authorizes sending then
if want-to-send(port)
then wake-up-task(send-task-id)
else ok-to-send(port)=true
for each char in packet do handler(char)
end

Arming Downcall
**Example: Remote Login Service (Sending)**

```plaintext
display-keyboard-handler():
  // upcalled by interrupt handler for keyboard
  get character from keyboard device
  and put in keyboard-buffer
  transport-arm-for-send
    (port, display-get-data)
end

display-get-data (packet):
  // upcalled by to send data
  copy data from keyboard-buffer into packet
end

net-send(packet, port):
  start net device to send packet
end

transport-arm-for-send (port, send-handler):
  transport-send-handler-array (port) = send-handler
  if ok-to-send(port)
    then wakeup-task(send-task-id)
  else want-to-send(port) = true
end

transport-send(port):
  // runs in task identified by send-task-id
  if ok-to-send(port) = false then block()
  allocate packet and fill in headers
  send-handler =
    transport-send-handler-array(port)
  send-handler(packet)
  // upcall display level to get data
  net-send(packet, port)
  ok-to-send(port) = false
  want-to-send(port) = false
end

transport-receive(packet, port):
  // upcalled by net layer
  handler = transport-handler-array(port)
  validate packet header
  if packet authorizes sending then
    if want-to-send(port)
      then wakeup-task(send-task-id)
    else ok-to-send(port) = true
    for each char in packet do
      handler(char)
end
```

**Intra-layer Signal**
Example: Remote Login Service (Sending)

```plaintext
display-keyboard-handler():
//upcalled by interrupt handler for keyboard
get character from keyboard device
and put in keyboard-buffer
transport-arm-for-send
(port, display-get-data)
end

display-get-data (packet):
//upcalled by to send data
copy data from keyboard-buffer into packet
end

net-send(packet, port):
start net device to send packet
end

transport-arm-for-send (port, send-handler):
transport-send-handler-array (port)=
send-handler
if ok-to-send(port)
then wakeup-task(send-task-id)
else want-to-send(port)=true
end

transport-send(port):
//runs in task identified by send-task-id
if ok-to-send(port)=false then block()
allocation packet and fill in headers
send-handler=
transport-send-handler-array(port)
send-handler(packet)
//upcall display level to get data
net-send(packet, port)
ok-to-send(port)=false
want-to-send(port)=false
end

transport-receive(packet, port):
//upcalled by net layer
handler=transport-handler-array(port)
validate packet header
if packet authorizes sending then
if want-to-send(port)
then wakeup-task(send-task-id)
else ok-to-send(port)=true
for each char in packet do handler(char)
end
```

Upcall
**Example: Remote Login Service (Sending)**

```plaintext
display-keyboard-handler():
  // upcalled by interrupt handler for keyboard
  get character from keyboard device
  and put in keyboard-buffer
  transport-arm-for-send
    (port, display-get-data)
end

display-get-data (packet):
  // upcalled by to send data
  copy data from keyboard-buffer into packet
end

net-send(packet, port):
  // start net device to send packet
end

transport-arm-for-send (port, send-handler):
  transport-send-handler-array (port) = send handler
  if ok-to-send(port)
    then wakeup-task(send-task-id)
  else want-to-send(port) = true
end

transport-send(port):
  // runs in task identified by send-task-id
  if ok-to-send(port) = false then block()
  allocate packet and fill in headers
  send-handler =
    transport-send-handler-array(port)
  send-handler(packet)
  // upcall display level to get data
  net-send(packet, port)
  ok-to-send(port) = false
  want-to-send(port) = false
end

transport-receive(packet, port):
  // upcalled by net layer
  handler = transport-handler-array(port)
  validate packet header
  if packet authorizes sending then
    if want-to-send(port)
      then wakeup-task(send-task-id)
    else ok-to-send(port) = true
    for each char in packet do handler(char)
end
```

**Downcall**
Advantages of Upcalls

- Efficiency: Due to the usage of synchronous subroutine calls instead of asynchronous IPC signals for interlayer communication.

- Simplicity:
  - No data buffering needed;
  - Easier for lower layers to “ask advice” of upper layers.

- “Piggybacking” occurs naturally in outgoing packets.
Advantages of Multi-task Modules

- Programmers deal with subroutine interfaces instead of IPC interfaces.
- Decisions about how tasks are used can be made late in the design process.
Problems and Solutions

- **Problem**: Upcalls break the “depends on” relationship established by layering. If a client fails, all other clients are in risk of failure too.

- **Solution**: Make sure data shared among the upcall clients (upper layer instances) is consistent and unlocked before invoking the upcall.
Problems and Solutions

- **Problem**: In case of a failure, the task that invoked the upcall has to be recovered or terminated, which could be a problem in case the task is responsible for a precious resource.

- **Solution**: Make tasks expendable (separate tasks for upcalling each client).
Problems and Solutions

- **Problem:** In case of a client failure, per-layer resources have to be freed up.

- **Solution:** Provide layer-specific cleanup functions that the system upcalls in case a client fails.

- **Problem:** Infinite loops cannot be distinguished from long-running computations.

- **Solution:** Leave the decision to a timer.
Problems and Solutions

• **Problem**: Indirect recursive upcalls.

• **Solutions**:
  - Put layer’s variables in a consistent state before initiating the upcall;
  - Prohibit recursive downcalls;
  - Downcalls to queue work requests for later execution by the task holding the lock;
Problems and Solutions

- **Problem**: Indirect recursive upcalls.

- **Solutions**:  
  - Downcalls to do nothing but set flags that are checked at known times by other tasks including the task making the upcall;  
  - For upcalls triggering the same downcall, replace the downcall by extra return arguments or another upcall to query the client.
Swift

- The authors developed the swift system based on the concepts of upcalls and multi-task modules.

- Swift is written in a typesafe high-level language inside a single shared address space with threads representing tasks of a multi-task module.

- According to the authors, experiments on their system showed performance improvements by a factor of five to ten over process-based layering.
Conclusions

- Layering is a program structure that enhances modularity and facilitates verification.
- Mapping layers onto processes is a bad idea.
- Upcalls and multi-task modules provide a more efficient and simpler interface to layered architectures.
- Upcalls programming methodology imposes a new range of problems that need to be carefully dealt with.