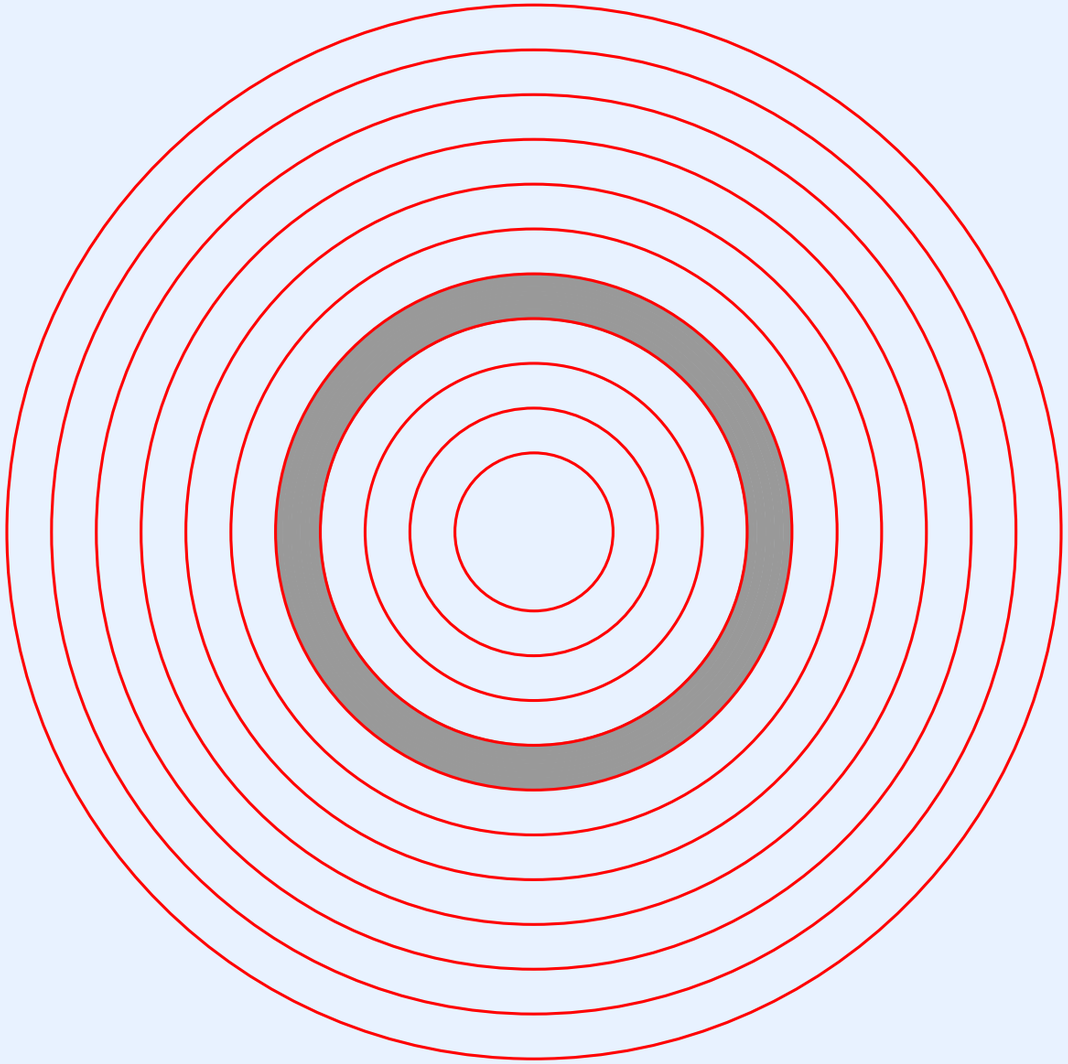
Module VIII

Inter-Process Communication (Message Passing)

Location Of Inter-Process Communication In The Hierarchy



Inter-Process Communication

- Can be used for
  - Exchange of (nonshared) data among processes
  - Some forms of process coordination
- The general technique is known as *message passing*

Two Approaches To Message Passing

- Approach #1
  - Message passing is one of many services the operating system offers
  - Messages are basically data items sent from one process to another, and are independent of both normal I/O and process synchronization services
  - Message passing functions are implemented using lower-level mechanisms
- Approach #2
  - The entire operating system is *message-based*
  - Messages, not function calls, provide the fundamental building block
  - Messages are used to coordinate and control processes
- Note: a few research projects used approach #2, but most systems use approach #1

An Example Design For A Message Passing Facility

- To understand the issues, we will begin with a trivial message passing facility
- Our example facility will allow a process to send a message directly to another process
- In principle, the design should be straightforward
- In practice, many design decisions arise

Message Passing Design Decisions

- Are messages fixed size or variable size?
- What is the maximum message size?
- How many messages can be outstanding at a given time?
- Where are messages stored?
- How is a recipient specified?
- Does a receiver know the sender’s identity?
- Are replies supported?
- Is the interface synchronous or asynchronous?

Synchronous vs. Asynchronous Interface

- A synchronous interface
  - An operation blocks until the operation is performed
  - A sending process is blocked until the recipient accepts the message being sent
  - A receiving process is blocked until a message arrives
  - Is easy to understand and use
  - A programmer can create extra processes to obtain asynchrony

Synchronous vs. Asynchronous Interface (continued)

- An asynchronous interface
  - A process starts an operation
  - The initiating process continues execution
  - A notification arrives when the operation completes
    * The notification can arrive at any time
    * Typically, notification entails abnormal control flow (e.g., “callback” mechanism)
  - Is more difficult to understand and use
  - Polling can be used to determine the status

Why Message Passing Choices Are Difficult

- Message passing interacts with scheduling
  - Process *A* sends a message to process *B*
  - Process *B* does not check messages
  - Process *C* sends a message to process *B*
  - Process *B* eventually checks its messages
  - If process *C* has higher priority than *A*, should *B* receive the message from *C* first?
- Message passing affects memory usage
  - If messages are stored with a receiver, senders can use up all the receiver's memory by flooding the receiver with messages
  - If messages are stored with a sender, receivers can use up all the sender's memory by not accepting messages

An Example Message Passing Facility

- We will examine a basic, low-level mechanism
- The facility provides direct process-to-process communication
- Each message is one word (e.g., an integer)
- A message is stored with the receiving process
- A process only has a one-message buffer
- Message reception is synchronous and buffered
- Message transmission is asynchronous
- The facility includes a “reset” operation

An Example Message Passing Facility (continued)

- The interface consists of three system calls

```
send(pid, msg);

msg = receive();

msg = recvclr();
```

- *Send* transmits a message to a specified process
- *Receive* blocks until a message arrives
- *Recvclr* removes an existing message, if one has arrived, but does not block
- A message is stored in the *receiver's* process table entry

An Example Message Passing Facility (continued)

- The system uses “first-message” semantics
  - The first message sent to a process is stored until it has been received
  - Subsequent attempts to send to the process fail

How To Use First-Message Semantics

- The idea: wait for one of several events to occur
- Example events
  - I/O completes
  - A user presses a key
  - Data arrives over a network
  - A hardware indicator signals a low battery
- To use message passing facility to wait for the first event
  - Create a process for each event
  - When the process detects its event, have it send a message

How To Use First-Message Semantics
(continued)

- The idiom a receiver uses to identify the first event that occurs

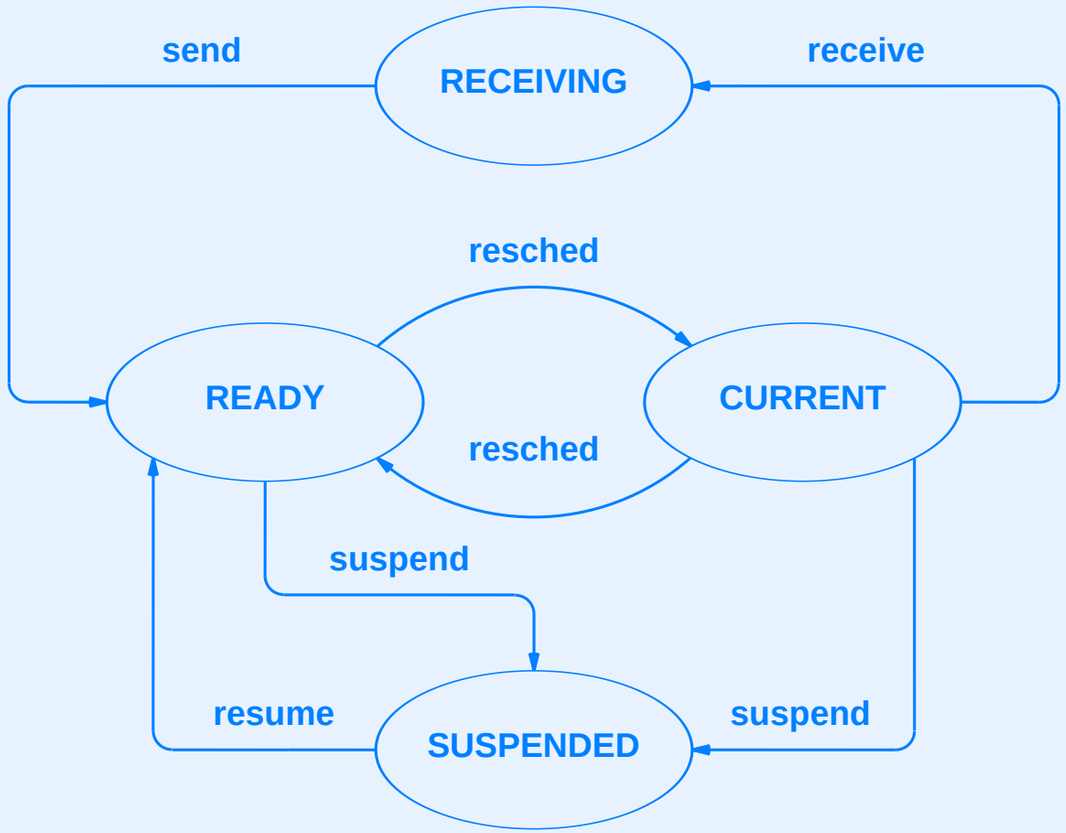
```
recvclr();  /* prepare to receive a message */
... /* allow other processes to send messages */
msg = receive();
```

- The above code returns first message that is sent, even if a higher priority process attempts to send later
- The receiver will block until a message arrives

A Process State For Message Reception

- While receiving a message, a process is not
  - Executing
  - Ready
  - Suspended
- Therefore, a new state is needed for message passing
- The state is named *RECEIVING*
- The state is entered when *receive* called
- The code uses constant *PR_RECV* to denote a *receiving* state

State Transitions With Message Passing



The Steps Taken To Receive A Message

- The current process calls *receive*
- *Receive* checks the current process's entry in the process table
- If no message has arrived, *receive* moves the calling process to the *RECEIVING* state to block until a message arrives
- Once a message arrives,, the process is moved to the *READY* state and execution of *receive* will eventually continue when resched chooses to run the process
- The code in *receive* extracts a copy of the message from the process table entry and resets the process table entry to indicate that no message is present
- *Receive* then returns the message to its caller

Blocking To Wait For A Message

- We have seen how the *suspend* function suspends the current process
- Blocking the current process to receive a message is almost the same
- *Receive*
  - Finds the current process's entry in the process table, *proctab[currpid]*
  - Sets the state in the process table entry to *PR_RECV*, indicating that the process will be receiving
  - Calls *resched*

Xinu Code For Message Reception

```
/* receive.c - receive */

#include <xinu.h>

/*------------------------------------------------------------------------
 *  receive  -  Wait for a message and return the message to the caller
 *------------------------------------------------------------------------
 */
umsg32  receive(void)
{
        intmask mask;                   /* Saved interrupt mask         */
        struct  procent *prptr;         /* Ptr to process's table entry */
        umsg32  msg;                    /* Message to return            */

        mask = disable();
        prptr = &proctab[currpid];
        if (prptr->prhasmsg == FALSE) {
                prptr->prstate = PR_RECV;
                resched();              /* Block until message arrives  */
        }
        msg = prptr->prmsg;             /* Retrieve message             */
        prptr->prhasmsg = FALSE;        /* Reset message flag           */
        restore(mask);
        return msg;
}
```

Message Transmission

- To send a message, a process calls *send* specifying a destination process and a message to send to the process
- The code
  - Checks arguments
  - Returns an error if the process already has a message waiting
  - Deposits the message
  - Makes the process ready if it is in the receiving state
- Note: the code also handles a receive-with-timeout state, but we will consider that state later

Xinu Code For Message Transmission (Part 1)

```
/* send.c - send */

#include <xinu.h>

/*------------------------------------------------------------------------
 *  send  -  Pass a message to a process and start recipient if waiting
 *------------------------------------------------------------------------
 */
syscall send(
          pid32         pid,            /* ID of recipient process      */
          umsg32        msg             /* Contents of message          */
        )
{
        intmask mask;                   /* Saved interrupt mask         */
        struct  procent *prptr;         /* Ptr to process's table entry */

        mask = disable();
        if (isbadpid(pid)) {
                restore(mask);
                return SYSERR;
        }
```

Xinu Code For Message Transmission (Part 2)

```
        prptr->prmsg = msg;             /* Deliver message              */
        prptr->prhasmsg = TRUE;         /* Indicate message is waiting  */

        /* If recipient waiting or in timed-wait make it ready */

        if (prptr->prstate == PR_RECV) {
                ready(pid);
        } else if (prptr->prstate == PR_RECTIM) {
                unsleep(pid);
                ready(pid);
        }
        restore(mask);          /* Restore interrupts */
        return OK;
}
```

Xinu Code For Clearing Messages

```
/* recvclr.c - recvclr */

#include <xinu.h>

/*------------------------------------------------------------------------
 *  recvclr  -  Clear incoming message, and return message if one waiting
 *------------------------------------------------------------------------
 */
umsg32  recvclr(void)
{
        intmask mask;                   /* Saved interrupt mask         */
        struct  procent *prptr;         /* Ptr to process's table entry */
        umsg32  msg;                    /* Message to return            */

        mask = disable();
        prptr = &proctab[currpid];
        if (prptr->prhasmsg == TRUE) {
                msg = prptr->prmsg;     /* Retrieve message             */
                prptr->prhasmsg = FALSE;/* Reset message flag           */
        } else {
                msg = OK;
        }
        restore(mask);
        return msg;
}
```

Summary Of Message Passing

- Message passing offers an inter-process communication system
- The interface can be synchronous or asynchronous
- A synchronous interface is the easiest to use
- Xinu uses synchronous reception and asynchronous transmission
- An asynchronous operation allows a process to clear any existing message without blocking
- The Xinu message passing system only allows one outstanding message per process, and uses first-message semantics

Questions?