Inter-process communication

CS503: Operating systems, Spring 2019

Pedro Fonseca
Department of Computer Science
Purdue University

Edited by Byoungyoung Lee and Pedro Fonseca

Previous lecture

- Conditional variables
- Dining philosopher problem
- Sleeping barber problem
- Deadlocks, livelocks, starvation
- Priority inversion
- XINU semaphores

Inter-process communication (IPC)

- IPC enables and facilitates communication between processes
- Possible approaches:
  - Signal
  - Socket
  - Pipe
  - Shared memory
  - **Message passing** (Xinu)

Pipes in Linux

```c
// int pipe(int pipefd[2]);

int main(int argc, char *argv[]) {
    int pipefd[2];
    pid_t cpid;
    char buf;
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <string>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    if (pipe(pipefd) == -1) {
        perror("pipe");
        exit(EXIT_FAILURE);
    }

    cpid = fork();
    if (cpid == -1) {
        perror("fork");
        exit(EXIT_FAILURE);
    }

    if (cpid == 0) {    /* Child reads from pipe */
        close(pipefd[1]);          /* Close unused write end */

        while (read(pipefd[0], &buf, 1) > 0)
            write(STDOUT_FILENO, &buf, 1);

        write(STDOUT_FILENO, "\n", 1);
        close(pipefd[0]);
        _exit(EXIT_SUCCESS);
    } else {            /* Parent writes argv[1] to pipe */
        close(pipefd[0]);          /* Close unused read end */
        write(pipefd[1], argv[1], strlen(argv[1]));
        close(pipefd[1]);          /* Reader will see EOF */
        wait(NULL);                /* Wait for child */
        exit(EXIT_SUCCESS);
    }
}
```

Inter-process communication

- Used for:
  - Exchange of (nonshared) data
  - Process coordination
- General techniques: **message passing**

Two approaches to message passing

- Approach #1:
  - Message passing is one of many services
  - Messages are separate from I/O and process synchronization services
  - Messages implemented using lower-level mechanisms, such as semaphores

Two approaches to message passing

- Approach #2:
  - The entire operating system is message-based
  - Messages, not function calls, provide the fundamental building block
  - Messages, not semaphores, used for process synchronization
  - Remote Procedural Call (RPC), CORBA, DCOM, SOAP
    - (Un)Marsharling, Interface Description Language (IDL)

Design of a message passing facility

- To understand the issues, we will begin with a trivial message passing facility
- We want to 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 or variable size?
- What is the maximum message size?
- Hoe many messages can be outstanding at a given time?
- Where are messages stored?
- How is a recipient specified?
- How does the receiver know the sender's identity?
- Are replies supported?
- Is the interface synchronous or asynchronous?

Synchronous vs. asynchronous interface?

- Synchronous interface
  - Blocks until the operation is performed
  - Easy to understand / program
  - Extra processes can be used to obtain asynchrony

Synchronous vs. asynchronous interface?

- Asynchronous interface
  - Process starts an operation
  - Initializing process continues the operation
  - Notification:
    - Arrives when the operation completes
    - May entail abnormal control flow (e.g., software interrupt or “callback” mechanism)
  - Pooling can be used to determine status

Why is a message passing mechanism so difficult to design?

- Interacts with:
    - Process coordination subsystem
    - Memory management subsystem
- Affects the user perception of system

Example inter-process message passing design

- Simple, low-level mechanism
- Direct process-to-process communication
- One-word messages
- Messages stored with receiver
- One-message buffer
- Synchronous, buffered reception
- Asynchronous transmission and “reset” operation

Example inter-process message passing design (cont.)

- Three functions:
  - `send(msg, pid)`
  - `msg = receive()`
  - `msg = recvclr()`
- Messages stored in receiver's process table entry
- Send transmits message to specified process
- Receive blocks until message arrives
- Recvclr removes existing message, if one has arrived, but does not block

Example inter-process message passing design (cont.)

- First-message semantics:
  - First message sent to a process is stored until it has been received
  - Subsequent attempts to send fail
- Idiom:

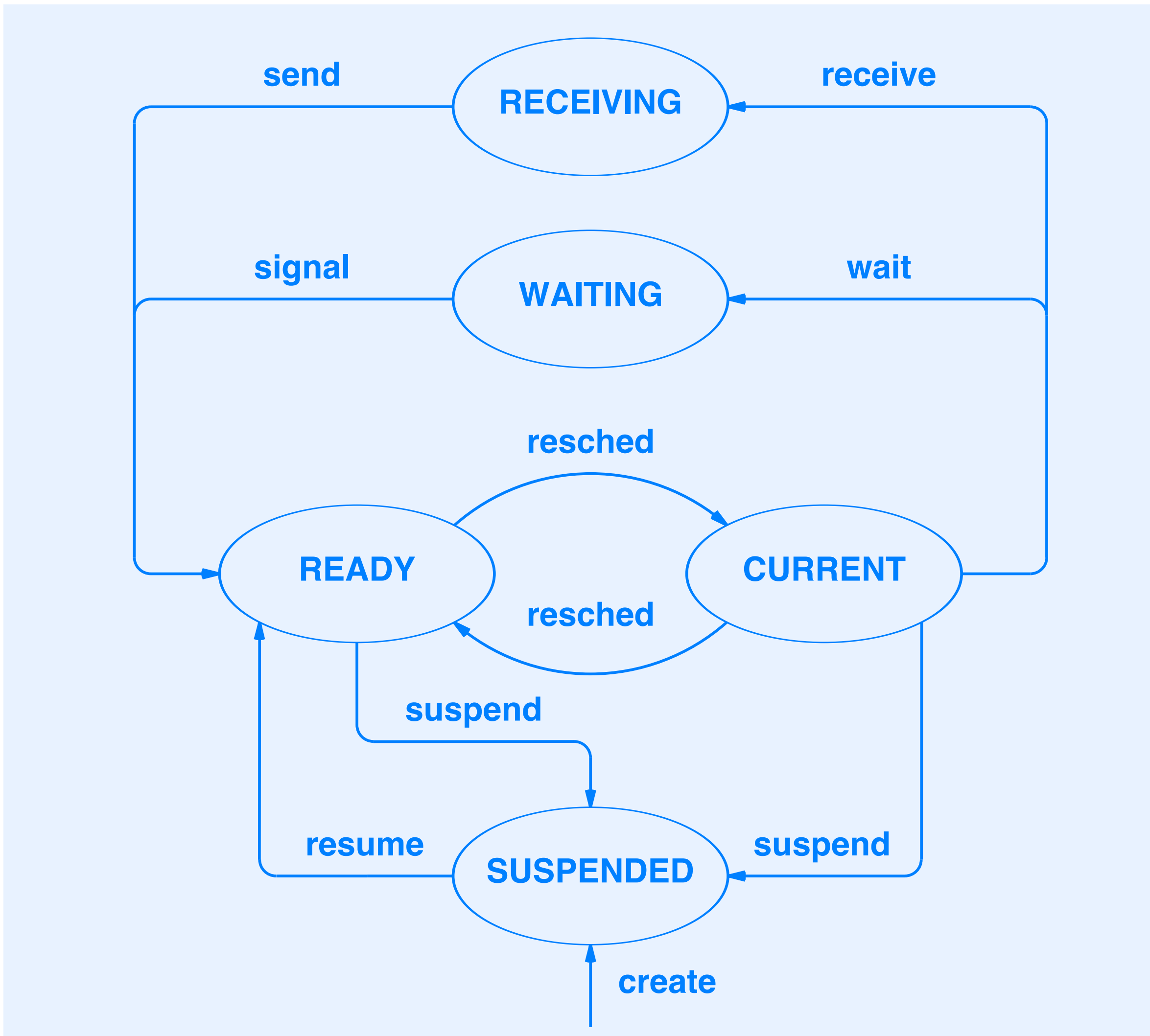
```
recvclr();

/* prepare to receive a message */

... /* allow other processes to send messages */

msg = receive();
```

- Above code returns first message that was sent, even if a high priority process sends later

Process state for message reception

- While receiving a message, a process is not:
  - Executing
  - Ready
  - Suspended
  - Waiting on a semaphore
- New state: RECEIVING
- Entered when received called

State Transitions with Message Passing



XINU code for message passing

- receive() in receive.c
- send() in send.c
- recvclr() in recvclr.c

Summary

- Inter-process communication
- Implemented by message passing
  - Can be synchronous or asynchronous
- Synchronous interface is the simplest
- Xinu uses synchronous reception and asynchronous transmission