CS 537
Lecture 4
Inter-Process Communication
Michael Swift

Notes
• Homework 1 due today in class, on paper
• Quiz 1 will be next Tuesday at the beginning of section
  • probably 3 questions
  • Covers up until today’s lecture
    – hardware support for OS
    – system calls
    – processes
    – one question will be from book
• Book material in first two reading assignments

Project Questions
• How was project 0?
• What was easy?
• What was hard?
• What can I teach you about?

Questions for this Lecture
• How can multiple processes cooperate?
Interprocess Communication (IPC)

- To cooperate usefully, threads must communicate with each other
- How do processes and threads communicate?
  - Shared Memory
  - Message Passing
  - Signals

Interprocess Communication

- Processes within a system may be independent or cooperating
  - Cooperating process can affect or be affected by other processes, including sharing data
- Reasons for cooperating processes:
  - Information sharing
  - Computation speedup
  - Modularity
  - Convenience
- Cooperating processes need interprocess communication (IPC)
- Two models of IPC
  - Shared memory
  - Message passing

Cooperating Processes

- Independent process cannot affect or be affected by the execution of another process
- Cooperating process can affect or be affected by the execution of another process
- Advantages of process cooperation
  - Information sharing
  - Computation speed-up
  - Modularity
  - Convenience

Communications Models

- Two communication models:
  - Shared memory
  - Message passing
IPC: Shared Memory

- Processes
  - Each process has private address space
  - Explicitly set up shared memory segment within each address space
- Threads
  - Always share address space (use heap for shared data)
- Advantages
  - Fast and easy to share data
- Disadvantages
  - Must synchronize data accesses; error prone
- Synchronization: Topic for end of semester

IPC: Signals

- Signal
  - Software interrupt that notifies a process of an event
  - Examples: SIGFPE, SIGKILL, SIGUSR1, SIGSTOP, SIGCONT
- What happens when a signal is received?
  - Catch: Specify signal handler to be called
  - Ignore: Rely on OS default action
  - Example: Abort, memory dump, suspend or resume process
  - Mask: Block signal so it is not delivered
    - May be temporary (while handling signal of same type)
- Disadvantage
  - Does not specify any data to be exchanged
  - Complex semantics with threads
  - Not implemented in Windows

IPC: Message Passing

- Message passing most commonly used between processes
  - Explicitly pass data between sender (src) + receiver (destination)
  - Example: Unix pipes, Windows LPC
- Advantages:
  - Makes sharing explicit
  - Improves modularity (narrow interface)
  - Does not require trust between sender and receiver
- Disadvantages:
  - Performance overhead to copy messages
- Issues:
  - How to name source and destination?
    - One process, set of processes, or mailbox (port)
  - Does sending process wait (i.e., block) for receiver?
    - Blocking: Slows down sender
    - Non-blocking: Requires buffering between sender and receiver

IPC: Message Passing details

- Mechanism for processes to communicate and to synchronize their actions
- Message system – processes communicate with each other without resorting to shared variables
- If P and Q wish to communicate, they need to:
  - establish a communication link between them
  - exchange messages via send/receive
- Implementation of communication link
  - physical (e.g., shared memory, hardware bus)
  - logical (e.g., logical properties)
Producer-Consumer Problem

- Paradigm for cooperating processes, producer process produces information that is consumed by a consumer process
  - unbounded-buffer places no practical limit on the size of the buffer
  - bounded-buffer assumes that there is a fixed buffer size

Bounded-Buffer – Shared-Memory Solution

- Shared data
  ```c
  #define BUFFER_SIZE 10
  typedef struct {
      ...
  } item;
  
  item buffer[BUFFER_SIZE];
  int in = 0;
  int out = 0;
  
  Solution is correct, but can only use BUFFER_SIZE-1 elements
  ```

Bounded-Buffer – Producer

```c
while (true) {
    /* Produce an item */
    while (((in = (in + 1) % BUFFER_SIZE count) == out));
    /* do nothing -- no free buffers */
    buffer[in] = item;
    in = (in + 1) % BUFFER_SIZE;
}
```

Bounded Buffer – Consumer

```c
while (true) {
    while (in == out);
    // do nothing -- nothing to consume
    // remove an item from the buffer
    item = buffer[out];
    out = (out + 1) % BUFFER_SIZE;
    return item;
}
```
Synchronization

- Message passing may be either blocking or non-blocking
- **Blocking** is considered **synchronous**
  - **Blocking send** has the sender block until the message is received
  - **Blocking receive** has the receiver block until a message is available
- **Non-blocking** is considered **asynchronous**
  - **Non-blocking send** has the sender send the message and continue
  - **Non-blocking receive** has the receiver receive a valid message or null

Buffering

- Queue of messages attached to the link; implemented in one of three ways
  1. Zero capacity – 0 messages
     Sender must wait for receiver (rendezvous)
  2. Bounded capacity – finite length of \( n \) messages
     Sender must wait if link full
  3. Unbounded capacity – infinite length
     Sender never waits