IPC

Hui Chen [a]

[a]CUNY Brooklyn College

March 11, 2021

Outline

1. Motivation
2. IPC
   - Shared Memory
   - Message Passing
3. Producer and Consumer Problem
4. UNIX (POSIX) and Windows IPC
5. Sharing Data among Threads and Processes

Outline

1. Motivation
2. IPC
   - Shared Memory
   - Message Passing
3. Producer and Consumer Problem
4. UNIX (POSIX) and Windows IPC
5. Sharing Data among Threads and Processes

Indepdent or Cooperating Processes

Processes within a system may be independent or cooperating.

- *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
  - Information sharing
  - Computation speed-up
  - Modularity

Multiprocess Architecture

Taking advantatage of *indepedent* or/and cooperativing processes, design multiprocess architecture applications

Example Applications

- The Chromimum projects
- The instructor's Monte Carlo simulation program to estimate π
- Shell scripts

What benefits do we get by using the multiprocess architecture?

Outline

1 Motivation

2 IPC
- Shared Memory
- Message Passing

3 Producer and Consumer Problem

4 UNIX (POSIX) and Windows IPC

5 Sharing Data among Threads and Processes

Interprocess Communication

- Cooperative processes communicate with each other to share data.
- There are two communication *models*
  - Shared memory
  - Message passing

Shared Memory

- OS must provide a system call to create a shared memory region.
- OS must attach the shared memory region to communicating processes' address spaces.
- OS must removes the restriction that normally one process is prevented from accessing another process's memory.
- All accesses to the shared memory region are treated as routine memory accesses, and no assistance from the kernel is required.
- The processes are also responsible for ensuring that they are not writing to the same location simultaneously.

Message Passing

- Processes exchange messages. There is no conflict needed to be avoided.
- IPC facility provides two operations:
  - `send(message)`
  - `receive(message)`
- Processes establish a communication link between them and exchange messages via `send/receive`

Design Message Passing

- Physical communication link can be shared memory, hardware bus, or network.
- Logically, the communication be
  - direction or indirect communication (like mailbox)
  - Blocking or non-blocking (synchronous or asynchronous)
  - explicit buffering or implicit (automatic) buffering

Outline

1. Motivation
2. IPC
   - Shared Memory
   - Message Passing
3. Producer and Consumer Problem
4. UNIX (POSIX) and Windows IPC
5. Sharing Data among Threads and Processes

Producer and Consumer Problem

The producer procudes information while the consumer consumes information

Bounded Buffer via Shared Memory

Shared Buffer

```
1 #define BUFFER_SIZE 10
2 typedef struct { } item;
3
4 // The following are shared among cooperating processes
5 item buffer[BUFFER_SIZE];
6 int in = 0;
7 int out = 0;
8
```

Producer

```
1 while (true) {/* produce an item in next
        produced */
2   while (counter == BUFFER_SIZE)
3     ; /* do nothing */
4   buffer[in] = next_produced;
5   in = (in + 1) % BUFFER_SIZE;
6   counter++;
7 }
8
```

Consumer

```
1 while (true) {
2   while (counter == 0)
3     ; /* do nothing */
4   next_consumed = buffer[out];
5   out = (out + 1) % BUFFER_SIZE;
6   counter--;
7   /* consume the item in next consumed
        */
8 }
9
```

Process Synchronization

Both producer and consumer may read and write to the shared memory concurrently ...

Producer and Consumer via Blocking Message Passing

Producer

```
1 message next_produced;
2 while (true) {
3   /* produce an item in next_produced */
4   send(next_produced); /* blocking */
5 }
6
7
```

Consumer

```
1 message next_consumed;
2 while (true) {
3   receive(next_consumed); /* blocking */
4   /* consume the item in next_consumed
        */
5 }
6
```

How aboub non-blocking message passing?

Outline

1. Motivation
2. IPC
   - Shared Memory
   - Message Passing
3. Producer and Consumer Problem
4. UNIX (POSIX) and Windows IPC
5. Sharing Data among Threads and Processes

UNIX IPC

Examine the example programs

- POSIX ordinary and named pipes
- POSIX shared memory
- POSIX message passing
- Berkeley Sockets

Windows IPC

Examine the example programs

- Windows anonymous and named pipes
- Windows mail slots
- Windows shared memory

Outline

1. Motivation
2. IPC
   - Shared Memory
   - Message Passing
3. Producer and Consumer Problem
4. UNIX (POSIX) and Windows IPC
5. Sharing Data among Threads and Processes

Which data sharing or IPC mechanism to use?

- Processes, or threads, or both?
- How do processes share data?
- How do threads share data?