IPC

Hui Chen [a]

[a]CUNY Brooklyn College

February 27, 2020

Outline

Outline

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

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

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  item next_produced;
2  while (true) {
3    /* 1. produce an item in next produced
         */
4    /* 2. do nothing when buffer is full
         */
5    while (((in + 1) % BUFFER_SIZE) == out
         ) ;
6    /* 3. produce an item and write it to
         current slot */
7    buffer[in] = next_produced;
8    /* 4. advance to next slot next slot
         */
9    in = (in + 1) % BUFFER_SIZE;
10 }
11
```

Consumer

```
1  item next_consumed;
2  while (true) {
3    item next_consumed;
4    /* 1. do nothing when buffer is empty
         */
5    while (true) {while (in == out) ;
6    /* 2. consume the item in current slot
         */
7    next_consumed = buffer[out];
8    /* 3. advance to next slot */
9    out = (out + 1) % BUFFER_SIZE;
10 }
11
```

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

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