Module 21: Problem and Solution

Lecture 42: Multi-core ComputingInter-process Communication

The Lecture Contains:

Previous Next

Semaphore Types

- Binary or counting
- Binary: Semaphore value can be T or F only
- Binary semaphores provide mutual exclusion
    - Sometimes known as mutex locks
- Counting semaphores
    - Can be used when the given resource has finite number of instances (n).
    - Initialize semaphore.value to n.

Synchronization Among Processes

- Two Processes P1 and P2.
- How do we make sure that P1 executes S1 first before P2 executes S2?

semaphore synch;

P1:	P2:
S1;	synch.wait();
synch.signal();	S2

Previous Next

OS Implementations

- Busy wait is not tolerated in an OS!!
- Bounded wait is to be ensured.
- Solution:
    - Use sleep and wakeup to move processes from running to waiting state and waiting to ready state.
    - Maintain a queue of processes waiting and wake up processes in that order.

```
class semaphore {
private:
int value;
list<PCB> wait_list;
public:
semaphore() {
value = 0;
wait_list = list<PCB>();
}
semaphore(int a) {
value = a;
wait_list = list<PCB>();
}
```

```
void wait(void) {
value--;
if (value < 0) { wait_list.push_front(
current);
sleep();
}
}
void signal(void) {
PCB p;
value++;
if (value <=0) {
p = *(wait_list.end()); wait_list.pop_back();
wakeup(p);
}
}
}
```

Wait also known as *P*
Signal also known as *V*

Previous Next

Linux API for Semaphores

- Semget
  - To get an array of semaphores
- Semctl
  - Semaphore controls. For example initial value
- Semop
  - Semaphore Operations (lock, signal etc.)

Pthread Mutex Locks

- Data Types: pthread_mutex_t
- Creation: pthread_mutex_init
- Lock: pthread_mutex_lock
- Unlock: pthread_mutex_unlock

  pthread_mutex_t mutex;
  pthread_mutex_init(&mutex, NULL);
  pthread_mutex_lock(&mutex);
  pthread_mutex_unlock(&mutex);

Win32 Mutex Locks and Semaphores

- Declaration
  - HANDLE mutex; (or HANDLE semaphore);
- Creation
  - CreateMutex: To create a mutex lock
  - CreateSemaphore: To create a semaphore
    - semaphore = CreateSemaphore(NULL, 1, 5, NULL);
- Wait for mutex or semaphore
  - WaitForSingleObject(HANDLE, WAITTYPE)
    - WaitForSingleObject(Sem, INFINITE);
- Releasing
  - ReleaseMutex(mutex)
  - ReleaseSemaphore(sem, 1, NULL)

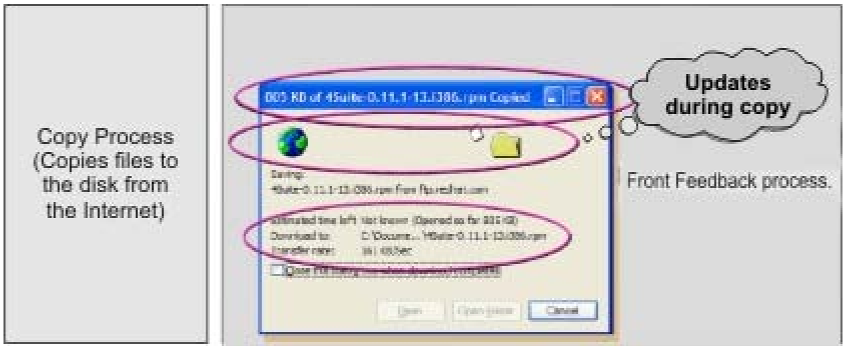
Previous Next

Multi-core ComputingInter-process Communication

Processes in a Group

- A process can be independent
    - Is not directly affected by other processes.
    - Does not affect other processes.
    - Example: /bin/ls and the shell
        - Are they related?
- Processes may be cooperating
    - Information Sharing
    - Speed up of execution
    - Modularity and convenience

Cooperating Processes (example)



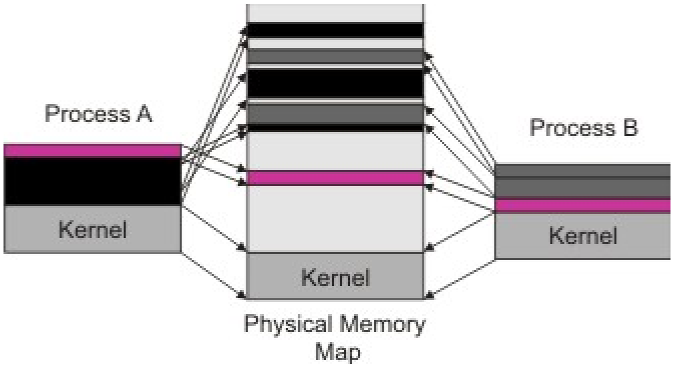
- Not really an example of “processes” but “threads”.
- The issues are the similar though.

Previous Next

Cooperating Processes

- Require
    - Inter process communication
        - Shared memory between processes
        - Message passing
            - Sender makes a call to the OS to send a message
            - Receiver makes a call to read message from the OS
    - Producer consumer relationship
        - A process produces data to be consumed by other process.

Shared Memory System

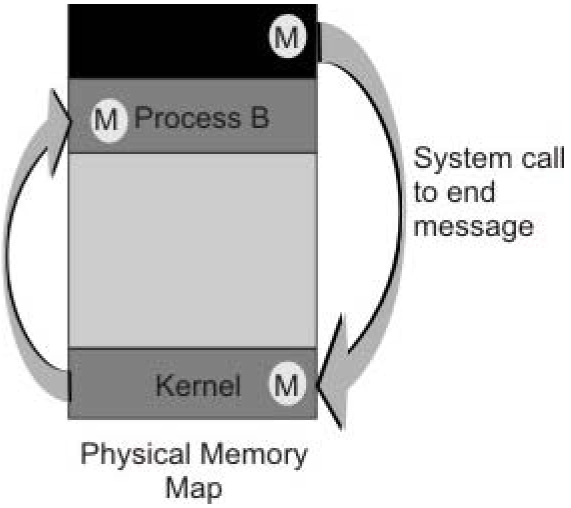


Previous Next

Module 21: Problem and Solution

Lecture 42: Multi-core ComputingInter-process Communication

Message Passing (Send)



Message Passing (Receive)

- Receive can be blocking
    - A process makes a system call to receive a message.
        - If message is not available, the process is made to sleep (wait) and woken up when message is received.
- Receive can be non-blocking
    - Process makes a system call to receive a message.
        - Return value from the system call determines whether a message is ready or not.

Previous Next

Lecture 42: Multi-core ComputingInter-process Communication

Cooperating Processes

- Inter process communication
  - Shared memory between processes
  - Message passing
- Producer consumer relationship

IPC: Shared Memory

- Shared buffer between processes

```
#define BUF_SZ 1024
typedef struct {
...
} BUF_Data;
struct {
BUF_Data items[BUF_SZ];
int inptr, outptr; /* Global variables */
} buffer;/* Must be shared between
/* two processes */
```

Producer and Consumer Code

```
void produce(BUF_Data item) {
while ((buffer.inptr+1)%BUF_SZ == buffer.outptr) ;
buffer.items[buffer.inptr] = item;
buffer.inptr = (buffer.inptr +1)%BUF_SZ;
}
BUF_Data consume(void) {
BUF_Data item;
while (buffer.outptr == buffer.inptr) ;
item = buffer.items[buffer.outptr];
buffer.outptr = (buffer.outptr +1)%BUF_SZ;
return (item);
}
```

Previous Next