Concurrent Programming and Synchronization Problems

What Are We Covering?

- OS manages resources
  - CPU scheduling
  - Memory management
  - Device management
    - Disk, terminal, clock
- OS provides file systems
  - File system organization and how to store files on disk
- Support concurrent programming
  - Processes and their states
  - Threads

Concurrent Programming

- What is concurrent programming?
  - Multiple execution units run concurrently
  - E.g., create new processes
  - E.g., create new threads
  - E.g., run two programs "concurrently"
- What is the problem?
  - If these units are independent (they don't interact), then no problem
- Sometimes, these concurrent units do need to interact

How Programs Interact?

- Shared memory model (like a blackboard)
  - Multiple threads look at the same information space for their operations

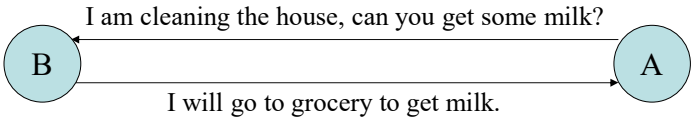
Now: x = 2; y = 3;
A: x = x + 3;
B: y = x – 2;
C: y = y * x;

x = 5; y = 15;

What if A and B and C are executed in parallel?
BAC: x = 5; y = 0;
CAB: x = 5; y = 3;
…

How Programs Interact?

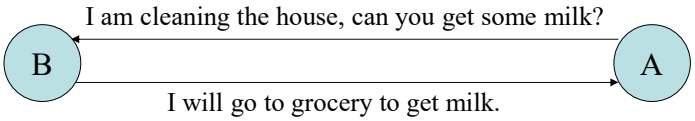
- Message passing model
  - Multiple processes (threads) have their own space
  - They interact by passing messages



What if A and B are not in sync?

How About Synchronization?

- Potential Problem in message passing model



A: Sent the message yesterday morning.
A: Still did not get confirmation in the evening.
A: Went to the grocery and bought milk.
B: Receive the message this morning.
B: Went and bought milk.

Synchronization in Shared Memory

- What are the possible outcomes if the two programs run concurrently?

```
compute-A ()
  { .....
    count = count + 2;
    print (count);
  }
```

```
compute-B ()
  { .....
    count = count – 3;
    print (count);
  }
```

Currently: count = 10

Possible execution orders	Printed results
A1 A2 B1 B2:	12 9
B1 B2 A1 A2:	7 9
A1 B1 A2 B2:	9 9

Any other???

Synchronization in Shared Memory

```
#include <pthread.h>
#include <stdio.h>

int counter;

void *increment (void *arg)
{ int j;   int id = *((int *) arg);

  printf ("Thread %d  started!\n", id);
  for (j = 0; j < 1000000; j++ ) counter++;
  printf ("Thread %d  is ending!\n", id);
}

int main()
{ int i;   pthread_t tid[5];

  for (i = 0; i < 5; i++)
    pthread_create (&tid[i], NULL, increment, (void *) &i);
  for (i = 0; i < 5; i++) pthread_join (tid[i], NULL);

  printf ("Final value of counter = %d\n", counter);
}
```

A bad example of thread programming!!!
How to pass parameters to a thread?
How to protect accesses by multiple threads (counter)?

Synchronization in Shared Memory

- Actually, more outputs can occur

```
compute-A ()
  { .....
    count = count + 2;
    print (count);
  }
```

```
compute-B ()
  { .....
    count = count – 3;
    print (count);
  }
```

⇒

A1 load count
A2 add 2
A3 store count
A4 print (count)

⇒

B1 load count
B2 add –3
B3 store count
B4 print (count)

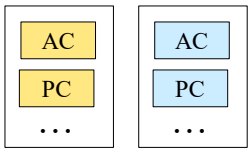
A1 B1 A2 A3 A4 B2 B3 B4: 12 7
A1 B1 A2 B2 A3 B3 A4 B4: 7 7

Synchronization in Shared Memory

- Actually, more outputs can occur

A1 load count
A2 add 2
A3 store count
A4 print (count)

B1 load count
B2 add –3
B3 store count
B4 print (count)



A: yellow B: blue

Memory

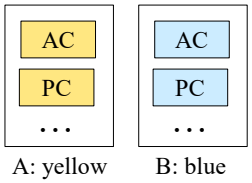
A exec		load	add				store	print	
B exec				load	add	store			print
AC(A)		10	12				12		
AC(B)				10	7	7			
M	10					7	12		
Screen								12	12

Assumption: print(count): print the memory content of count out (hidden load)

Synchronization in Shared Memory

- For the posted thread example

originally count = 0
loop 10 times
  load count
  add 1
  store count



Context switch in AC got 5 previously

	loop 5 times	load					add	store	loop 4 times			
			load	add	store	loop 8 times				load	add	store
AC(A)	0 to 5	5					6	6	6-10			
AC(B)			5	6	6	7 to 14				10	11	11
M	0 to 5				6	7 to 14		6	6-10			11

What to Expect?

- Which way we want the two threads to execute?
  - A first, then B?
  - B first, then A?
  - One at a time, AB or BA both are OK
  - Any execution outcome is fine
- How to achieve the desired execution order?
  - First case: Initialization: Bwait = 1;
    A: execute A code; Bwait = 0;
    B: while (Bwait) do nothing; execute B code;
  - Third case: use a lock box, whoever goes in, lock it
  - Last case: nothing needed

What to Expect?

- Which way we want the two program to execute?
  - One at a time, AB or BA, both are OK
  - This is called **Mutual Exclusion**
- How to achieve mutual exclusion?
  - Use a lock box, whoever goes in, lock it
  - Within the lock box is a **Critical Region**

What to Expect?

- Formalizing the concept
  - **Mutual Exclusion: A requirement for a certain shared object (e.g. code/resources), when satisfied, the system guarantees that no two processes (or other types of agents) will access the object at the same time.**
- How to achieve the desired execution order?
  - **Critical Region: A section of code, or collection of operations in which only one process may be executing at a given time.**
- Identify critical regions in a program
- Use **lock, semaphore, or monitor** to achieve mutual execution for the critical regions

What to Expect?

- Which way we want the two program to execute?
  - A first, then B
  - This is called **synchronization**
- How to achieve synchronization
  - Use **lock** or **semaphore** to achieve synchronization

Readings

- Section 5.1