9 Lecture: More Synchronization

Outline:

Announcements
LWP?
More Interprocess Communication
    Further generalization: message passing
Robust Programming Clinic
Classic IPC Problems

9.1 Announcements

- Coming attractions:

Event	Subject	Due Date			Notes
asgn3	dine	Wed	Feb 4	23:59	
lab03	problem set	Mon	Feb 9	23:59	
midterm	stuff	Wed	Feb 11		
lab04	scavenger hunt II	Wed	Feb 18	23:59	
asgn4	/dev/secret	Wed	Feb 25	23:59	
lab05	problem set	Mon	Mar 9	23:59	
asgn5	minget and minls	Wed	Mar 11	23:59	
asgn6	Yes, really	Fri	Mar 13	23:59	
final (sec01)		Fri	Mar 20	10:10	
final (sec03)		Fri	Mar 20	13:10	

  Use your own discretion with respect to timing/due dates.

- Don't just click the little 'X' to stop minix
- function pointers
- schedulers
- context structure
- tryAsgn2

9.2 LWP?

- Assignment news:
  - Yes, really.
- Winding up the stack
  1. Figure out what you want the world to look there (executing the first instruction of threadfun with the universe looking like it'd just been called)
  2. Figure out how you're going to get there. `swap_rfiles()'s` endgame:
     <load context>
     `leave`
     `ret`

Leave is:

```
movq %rbp, %rsp
popq %rbp
```

So, putting it all together, the endgame is:

```
movq %rbp, %rsp   ; copy base pointer to stack pointer
popq %rbp         ; pop the stack into the base pointer
ret               ; pop the stack into the instruction pointer
```

3. work backwards through this to figure out what you want the loaded context to look like so you wind up where you wanted to be in step 1.

You know what you need to know from the description of the normal linkage, even though you're doing something distinctly abnormal.

- Function pointers
- Don't separate {`load,store`}`_state()`.
- tryAsgn2

9.3 More Interprocess Communication

Ok, so, looking at our mechanisms that work

Spin Locks:	Waste Time
Semaphores:	Complicated (easy to get wrong)
Monitors:	Require language support

So what else can we do?

9.3.1 Further generalization: message passing

If you need me, just whistle.

Monitors require language support, a tricky thing in the "real world". Message passing gets much of the same safety using operating system support.

Processes communicate via two primitive functions:

- send(dst,message)
- receive(src,message)

Where do they get sent? (How do we address them?)
Options:

- individual processes
- mailboxes (possibly multiple consumers)

In either case, there may or may not be buffers. If there is a buffer, the OS has to manage this buffer (a meta-producer-consumer problem :). If not, a sender is stopped until there is a receiver: rendezvous

Issues with message passing.

- messages can be lost (acknowledgement)

- messages can be duplicated (sequence numbers)
- messages can be forged (authentication)
- performance:
  - Copying overhead (copy, copy, copy)
  - Transmission latency (You may as well do it yourself)

But it can elegantly solve some problems, as with the producer-consumer solution given in Figure 16. Message passing producer-consumer: send empty messages and send back full ones.

```
#define N 100

void producer(void) {
  message m;
  thing item;
  while(TRUE) {
    produce(&item);
    receive(consumer,&m);
    build_message(&m,item)
    send(consumer,&m);
  }
}
```

```
void consumer(void) {
  message m;
  thing item;
  int i;

  /* send N empties */
  for(i=0;i<N;i++)
    send(producer,&m);

  /* go to work */
  while(TRUE) {
    receive(producer,&m);
    extract_item(&m,item);
    send(producer,&m);
    consume(&item);
  }
}
```

Figure 16: A message-passing solution to the producer-consumer problem.

9.4 Classic IPC Problems

- Producer/Consumer (been there, done that?)
- Dining Philosphers(1965): synchronization rite of passage.

  Competing access for limited resources.

  Issues:
  - deadlock
  - starvation
- Readers/Writers: e.g. a database.

  Issues:

– Many readers can work simultaneously, but only one writer.
– Starvation: If we let all the readers by (no problem), the writer may starve.
– perhaps an aging scheme?