15-410

"...What could possibly go wrong..."

Grab Bag
Mar. 25, 2005

Dave Eckhardt

Bruce Maggs

Synchronization

Checkpoint 3

- Tonight, see bboard post
- Spending the time to really plan is worthwhile

Final Exam schedule posted

- We need *timely* notice of conflicts

Upcoming

- P3, P4 (but you knew that)
- Book report, another homework
  - Hint: book report was *assigned* well before last week of class...

Outline

When to use if () vs. while ()

Thinking about errors

- **Hmm...**
- **That's not right...**
- **Uh-oh...**

What Could Possibly Go Wrong?

```
void
join0(int *tidp, void **statusp)
{
  mutex_lock(&zl);
  if (!(tp = findzombie())
    cond_wait(&zc, &zm);
    tp = findzombie();
  mutex_unlock(&zm);
  *tidp = ...; *statusp = ...
  zap(tp);
}
```

What Could Possibly Go Wrong?

C usage note

- **Which is better?**

```
mutex_lock(&(objp->m));
mutex_lock(&objp->m);
```

- **What is the type of** `(&objp)->m` **?**

What We Hope For

join0()	*exit()*
`mutex_lock(&zl);`	
`if (!(...))`	
`cond_wait(&zc, &zl);`	
	`mutex_lock(&zl);`
	`append(self, ...)`
	`cond_signal(&zc);`
	`mutex_unlock(&zl);`
`tp = findzombie();`	
`mutex_unlock(&zl);`	
`...status...zap...`	

What Went Wrong?

Nothing!

What Went Wrong?

Nothing!

But what if there is *another* thread?

Not Exactly What We Hope For

join0()	*exit()*	*join0()*
`lock(&zl);`		
`if (!(...))`		
`wait(&zc, &zl);`		
	`lock(&zl);`	
	`append(...)`	
	`signal(&zc);`	
	`unlock(&zl);`	
		`lock(&zl);`
		`tp = fndzmb();`
		`unlock(&zl);`
`tp = fndzmb();`		`.........`
`...deref NULL...`		

Have We Seen This Before?

Dining Philosophers deadlock example

- Deadlock or not depended on scheduler
- Dining Philosophers really does exemplify lots of stuff

What went wrong?

- Protected world state wasn't ready for us
- We went to sleep
- Somebody prepared the world for us to run
- We ran
- We *assumed* nobody else had run
- We *assumed* the world state was still ready for us

To “if()” Or Not To “if()”?

```
void
join0(int *tidp, void **statusp)
{
  mutex_lock(&zl);
  while (!(tp = findzombie())
    cond_wait(&zc, &zm);
  mutex_unlock(&zm);
  *tidp = ...; *statusp = ...
  zap(tp);
}
```

Error Handling

Three kinds of error

- **Hmm...**
- **That's not right...**
- **Uh-oh...**

Important to classify & react appropriately

“New Player” - Take 1

```
// Improve memory locality:
// store players in array
struct player players[MAX];
struct player *new_player(int team, int num)
{
  int i;
  if ((i = emptyslot()) == -1)
    /* OH NO!!! */
    MAGIC_BREAK;
  }
  ...
}
```

“New Player” - Take 2

```
// Improve memory locality:
// store players in array
struct player players[MAX];
struct player *new_player(int team, int num)
{
  int i;
  if ((i = emptyslot()) == -1)
    /* OH NO!!! */
    while(1);
  }
  ...
}
```

What's Going On?

“Out of table slots” - what kind of thing?

- Should really never happen?
- Might happen sometimes?
- Likely to happen once a day?
  - Remember: users always want 110%!

What to do?

- *Resolve* reasonable issues when possible

“New Player” - Take 3

```
struct player *players;
int playerslots;
struct player *new_player(int team, int num)
{
  int i;
  if ((i = emptyslot()) == -1)
    if ((i = grow_table_and_alloc()) == -1)
      /* OH NO!!! */
      while(1);
  }
  ...
}
```

What's Going On?

“Out of heap space” - what kind of thing?

- Should really never happen?
- Might happen sometimes?
- Likely to happen once a day?

What's Going On?

“Out of heap space” - what kind of thing?

- Should really never happen?
- Might happen sometimes?
- Likely to happen once a day?

My suggestion

- “Might happen sometimes”

What to do?

- Hard to say what the right thing is for all clients
  - Is it fatal or not?
- Often: pass the buck

“New Player” - Take 4

```
struct player *players;
int playerslots;
struct player *new_player(int team, int num)
{
  int i;
  if ((i = emptyslot()) == -1)
    if ((i = grow_table_and_alloc()) == -1)
      return (NULL);
  }
  ...
}
```

“Free Player” - Take 1

```
void free_player(struct player *p)
{
  switch(player->role) {
  case CONTENDER:
    free(p->cstate); break;
  case REFEREE:
    free(p->refstate); break;
  }
  free(p->generic);
  mark_slot_available(p - players);
}
```

What's Wrong?

There is a sanity-check missing...

- **Probably somebody will make a mistake eventually**
- **Let's catch it**

“Free Player” - Take 2

```
void free_player(struct player *p)
{
  switch(player->role) {
  case CONTENDER:
    free(p->cstate); break;
  case REFEREE:
    free(p->refstate); break;
  default: return;
  }
  free(p->generic);
  mark_slot_available(p - players);
}
```

All Fixed?

No!

- The program has a bug
  - Maybe the client is passing us random player pointers
  - Maybe we are handing out invalid p->role values
- We happened to catch the bug this time
- We might not catch it every time!
  - A random player pointer might have a “valid” p->role

The program is *broken*

- Hiding the problem isn't our job
- Hiding the problem isn't even *defensible*

Should We “Crash”?

If the program is “broken”, should we “crash”?

- **Often: yes**
  - **Dumping core allows debugger inspection of the problem**
  - **Throwing running program into a debugger is probably nicer**

Summary

if vs. while

- If somebody can revoke your happiness, you'd better check

Three kinds of error

- Hmm...
  - Try to *resolve*
- That's not right...
  - Try to *report*
- Uh-oh...
  - Try to *help the developer* find the problem faster