

Process Interaction : Key terms

Starvation : When a process that could run never gets selected by the scheduler

Deadlock : When none of a group of processes can run as each is waiting for one of the others to complete something.

Livelock : When a group of processes each changes state in response to changes in one of the group without doing any useful work.

Race condition : Where the resultant value depends on the relative timing of a group of processes or threads.

Process Interaction : Key terms 2

Critical Section : Code within a process where access is made to a shared resource and that must not be executed while another process is accessing the shared resource

Mutual Exclusion : Term given to the requirement that when a process is in its critical section then no other process can be in a critical section that uses the same resources.

Process Interaction : Cooperation

Processes often need to cooperate in order to complete their tasks.

- A process may make a request for a service
 - eg data from disk.

- Processes may be designed to work together
 - `ps -aux | grep ngu | tr -s " " " " | cut -d \ -f 11`
 - requires four processes to work with each other

- Processes have to cooperate via a buffer
 - eg burning a cdrom.
 - aka producers & consumers

Process Interaction : Producers & Consumers

producer P and consumer C share a buffer,
ITEMS represents the current number of items in the buffer.

Assume ITEMS contains the value 1

C can execute the line
items --;

i) C commences execution of this
line

ii) process C gets switched out

v) C completes execution of this
line

P can execute the line
items ++;

iii) P commences execution of
this line

iv) P completes execution of the
line

Process Interaction : Producers & Consumers 2

What is the value of ITEMS now?

What should it be?

Looking at the compiled code (Intel x86) shows

for C

for P

i) MOV EAX, items ; 1

iii) MOV EAX, items ; 1

ii) DEC EAX ; 0

iv) INC EAX ; 2

vi) MOV items, EAX ; 0

v) MOV items, EAX ; 2

ITEMS now has 1 invisible item in it !!

What we need are 'atomic updates'

Mutual Exclusion : Software Solutions

The segment of a processes code that may affect other processes is known as its critical section.

The general structure for a process of this type is

beginning section
entry protocol for critical section
critical section
exit protocol for critical section
remainder section

Mutual Exclusion : Software Requirements

- Must work even if context switch occurs in the critical section.
- Must be independent of process speed.
- Must not be dependent on the ordering of the code
- Process must not terminate between the start of the entry protocol and the end of the exit protocol.

- Must be free from deadlock.
- Must be free from starvation
- Must have minimal overhead in entry and exit.

Software Solution : Algorithm 1

Using a 'guard' variable :

- initialize the guard value to 0

beginning section

WHILE (guard == 1)

 DO nothing

END WHILE

guard = 1

critical section

guard = 0

remainder section

test: MOV EAX, guard

 CMP EAX, #1

 JE test:

 MOV guard,#1

Software Solution : Algorithm 2

Using a 'turn' variable

- initialize the turn variable to 0

Process 0

beginning section

```
WHILE (turn != 0)
  DO nothing
END WHILE
```

critical section

turn = 1

remainder section

Process 1

beginning section

```
WHILE (turn != 1)
  DO nothing
END WHILE
```

critical section

turn = 0

remainder section

Software Solution : Algorithm 3

Using an array of two guards instead of a shared variable.
if flag[0] is 1 then process 0 is in the critical section.
if flag[1] is 1 then process 1 is in the critical section.

Process 0

beginning section

```
WHILE (flag[1] == 1)
  DO nothing
END WHILE
flag[0] = 1
  critical section
```

flag[0] = 0

remainder section

Process 1

beginning section

```
WHILE (flag[0] == 1)
  DO nothing
END WHILE
flag[1] = 1
  critical section
```

flag[1] = 0

remainder section

Software Solution : Algorithm 4

Using an array of two flags, version 2.

A process sets its flag to indicate intent.

A process then waits for the other flag to clear.

Process 0

beginning section

```
flag[0] = 1
WHILE (flag[1] == 1)
  DO nothing
END WHILE
  critical section
```

```
flag[0] = 0
```

remainder section

Process 1

beginning section

```
flag[1] = 1
WHILE (flag[0] == 1)
  DO nothing
END WHILE
  critical section
```

```
flag[1] = 0
```

remainder section

Software Solution : Algorithm 5

Using flags with turns

A process sets its flag to indicate intent.

A process then sets/clears the turn flag

Process 0

beginning section

```
flag[0] = 1
turn = 1
WHILE ((flag[1] == 1) and (turn == 1))
  DO nothing
END WHILE
critical section
```

flag[0] = 0

remainder section

Process 1

beginning section

```
flag[1] = 1
turn = 0
WHILE ((flag[0] == 1) and (turn == 0))
  DO nothing
END WHILE
critical section
```

flag[1] = 0

remainder section

Software Solution : Algorithm 5 cont.

Processes fulfil mutual exclusion

- process 0 blocks when
 - $(\text{flag}[1] == 1) \text{ and } (\text{turn} == 1)$

- process 1 blocks when
 - $(\text{flag}[0] == 1) \text{ and } (\text{turn} == 0)$

No deadlock as one part of the condition must be false for one process.

No strict turn taking.

Works at the machine code level.

Generalizes to any number of processes.

Algorithm 5 : Machine Code

Process 0

```
MOV flag[0], #1  
MOV turn, #1
```

```
test: CMP flag[1], #1  
      JNE enter:  
      CMP turn, #1  
      JE test:  
enter:
```

Process 1

```
MOV flag[1], #1  
MOV turn, #0
```

```
test: CMP flag[0], #1  
      JNE enter:  
      CMP turn, #0  
      JE test:  
enter:
```