

From last time



Explain briefly how starvation may occur in process scheduling. (2 marks)

In round-robin scheduling, new processes are typically placed at the end of the ready-state queue rather than at the beginning. Suggest a good reason for this. (2 marks)

A scheduler uses a time-slice of 4.5msec, and a context switch takes 0.5msec. What percentage of CPU time is spent on executing process instructions: (a) if processes use the whole time-slice? (b) if processes only need 0.5msec CPU-bursts?

In general, how would you improve the percentage of CPU time spent on executing process instructions? (3 marks)

COMP25111: Operating Systems
Lecture 8: Process/Thread Synchronisation

Oscar Palomar
School of Computer Science, University of Manchester
Autumn 2016

COMP25111 Lecture 8
Overview & Learning Outcomes

- Process Synchronisation
- Semaphores
- Deadlocks
- Dining Philosophers
- Message-passing
- Everything in this handout about process synchronisation also applies to thread synchronisation

1/28COMP25111 Lecture 8
Problem: Too much milk

Time	Person 1	Person 2	Person 3
8:00	sleeping	sleeping	check fridge - no milk
8:15	wakes up	sleeping	leave for lecture
8:30	check - no milk	sleeping	travelling
8:45	leave for store	wakes up	arrive at University
9:00	arrive at store	check - no milk	go to lecture
9:15	bought milk	leave for store	(lecture)
9:30	return home	arrive at store	(lecture)
9:45		bought milk	go to store
10:00		return home	arrive at store
10:15			bought milk
10:45			return home

COMP25111 Lecture 8
Data inconsistency

```
Concurrent access to shared data

static int i; // shared by A & B
// process or thread A    // process or thread B
i = 0;                    i = 0;
while (i < 100) {          while (i > -100) {
    i++;                    i--;
}
System.out.println("A");  System.out.println("B");

(assume both access & assignment are atomic i.e. indivisible)
```

- Questions:
- which will finish first?
 - will they ever finish?
 - if one finishes, will the other also finish?
 - does it help if one gets a head start?

3/28COMP25111 Lecture 8
Race Condition

Several processes manipulate shared data concurrently & outcome depends on precise order of what happens when

Q: what are a CPU's atomic operations?

e.g.: shared variable `i` in memory, initial value 4

process A: `i++`;
i.e. load Reg from `i`; `Reg = Reg + 1`; store Reg to `i`

process B: `i--`;
i.e. load Reg from `i`; `Reg = Reg - 1`; store Reg to `i`

Question: what can the final value of `i` be?

Definitions

Data inconsistency: disagreement about data values

Synchronisation: using appropriate policies and mechanisms to ensure the correct operation of cooperating processes

Critical section (Critical region): section of code in which shared data is used

Mutual exclusion (mutex): at most 1 process can be in its critical section at once

i.e. if more than one process tries to enter a critical section simultaneously, only one can succeed – others must wait

Semaphores

Dijkstra, 1965: An integer variable (e.g. *S*) accessed via two atomic operations (with Dutch names!):

P(*S*) (“try-to-reduce”, down, wait, aquire, probe, procure)

```
while (S<=0)
    ; /*no action*/
S--;
```

V(*S*) (“increase”, up, signal, release, vacate)

```
S++;
```

Initialise *S* appropriately
= number of processes allowed in critical section at once (usually 1)

Don't loop – yield

In practice, it is silly to busy-wait in *P* ()

P () adds process to a queue & gives up CPU

V () takes process from queue & makes it “ready”

Example – 1 semaphore

Two processes sharing *A*[100]; initialise *S*=1;

...	...
<i>P</i> (<i>S</i>);	<i>P</i> (<i>S</i>);
<i>r</i> 1= <i>A</i> [100];	<i>r</i> 2= <i>A</i> [100]; critical
<i>r</i> 1++;	<i>r</i> 2++;
<i>A</i> [100]= <i>r</i> 1;	<i>A</i> [100]= <i>r</i> 2; section
<i>V</i> (<i>S</i>);	<i>V</i> (<i>S</i>);
...	...

Example – 2 semaphores

Initialise: *S*1=0; *S*2=0; *I*=0;

```
... /* A */  ... /* B */  ... /* C */
P (S1);    P (S2);    for (j=0; j<10; j++)
I=I*2;    I=I+5;      I++;
V (S1);    V (S1);    V (S2);
...        ...        ...
```

Question: What is the the sequence of events?

Deadlock: everyone waiting for everyone else



Deadlock

A set of waiting processes, where each process is waiting for something that can only be provided by another of the processes

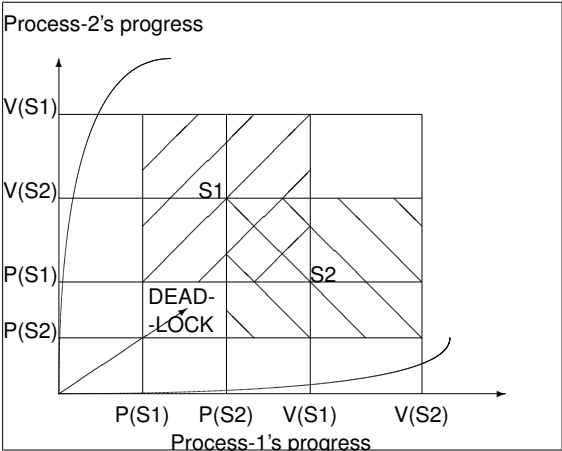
e.g. S1, S2 initialised to 1

```
// A    // B
P(S1); P(S2);
P(S2); P(S1);
...
V(S1); V(S2);
V(S2); V(S1);
```

Detection complicated

Once occurred, almost impossible for OS to solve

Prevention e.g. Deadlock (trajectory) Diagram



Benjamin D. Esham / Wikimedia Commons



The Dining Philosophers

5 philosophers, 5 plates of spaghetti, 5 forks, 1 circular table
Alternately think (alone) or eat with 2 forks (from right and left)

```
Semaphore fork[5]; // initialise to 1

philosopher (int i) {
    while(true) {
        System.out.println (i + " hungry");
        P(fork[i]); P(fork[(i+1)mod 5]);
        System.out.println (i + " eating");
        V(fork[i]); V(fork[(i+1)mod 5]);
        System.out.println (i + " thinking"); // delay?
    }
}
```

This may deadlock

Message-passing

- Shared data → Semaphore(s) control access
- No shared data → send copy to other processes
- Wait for message instead of semaphore
 - from known source(s), or from any source
- Con: slower(?)
- Pro: more general/flexible
 - RPC
 - multi-CPU
 - Distributed/Network

Summary of key points

- Process Synchronisation
- Semaphores
- Deadlocks – may occur in a variety of situations, usually fatal
- Dining Philosophers – many solutions
- Message-passing
- Everything in this handout about process synchronisation also applies to thread synchronisation**
- Next: Threads in Java.

Your Questions

For next time

Explain briefly how a deadlock may occur (2 marks)

shared variables x, y, s; initial values S1=S2=0 x=y=1 s=0

line	Thread A	Thread B	Explain the purpose of the semaphores in:
1.	do{	do{	
2.	V(S1)	P(S1)	– lines 2 & 3
3.	P(S2)	V(S2)	of both threads
4.	x=x+y	P(S1)	– line 4 of B
5.	V(S1)	y=x-y	& line 5 of A
6.	V(S1)	s=s+1	– lines 6 & 7 of A
7.	P(S2)	P(S1)	and 7 & 8 of B
8.	print s,y	V(S2)	
9.	}while(s<7)	}while(s<7)	(1 mark each)

Will A ever terminate? Justify your answer. (1 mark)

What is output by the print statement in line 8 of A? (3 marks)

Exam Questions

Exam Question ctd.

shared variables x1,x2,x3,x4,x5,x6; initially S1=S2=S3=0

	Thread A	Thread B	Thread C
2.	x1= 1	x2= 2	x3= 3
3.	V(S1)	V(S1)	P(S1); P(S1)
4.	P(S2)	P(S3)	V(S2); V(S3)
5.	x4= x2+x3	x5= x1+x3	x6= x1+x2
6.	V(S1)	V(S1)	P(S1); P(S1)
7.	P(S2)	P(S3)	V(S2); V(S3)
8.	x2= x5+x6	x3= x4+x6	x1= x4+x5

continued...

- Is it possible that:
- line 5 of A executes before line 2 of B?
 - line 5 of C executes before line 2 of A?
 - line 5 of B executes before line 5 of C?
 - line 5 of A executes before line 5 of B?
 - line 5 of A executes after line 5 of C?
 - line 8 of A executes before line 5 of B?
 - line 8 of A executes before line 5 of C?
 - line 8 of C executes before line 5 of B?

What are the final values of x1, x2, x3, x4, x5, x6?

If S3 is replaced by S2 throughout, can you find a pattern of execution which gives different answers to the questions above?

Glossary

- Shared data
- Concurrent access
- Data (in)consistency
- Atomic action/operation
- Race condition
- Synchronisation
- Critical section/region
- Mutual exclusion (mutex)
- Semaphore
- P()
- V()
- Busy-wait
- Deadlock
- Deadlock trajectory diagram
- Dining Philosophers
- Message-passing

Reading

- OSC/J: 6.1, 6.2, 6.5 (skim 6.3, 6.4, 6.6)
- older OSC/J: 7.1, 7.2, 7.5 (skim 7.3, 7.4, 7.6)
- MOS: 2.3 (opening paragraphs), 2.3.1, 2.3.2, 2.3.5 (skim 2.3, 2.4, intro of MOS2 ch.3 or MOS3 ch.6)
- Dining philosophers: AOS 7.6.3, MOS2 2.4.1)