Operating Systems (INFR09047)

2019/2020 Semester 2

Review #2

abarbala@inf.ed.ac.uk

Notes

Final Exam Reminder

- It covers all material presented in the course
  - All course slides and related book chapters
  - Review #1 notes
  - Review #2 notes (this slide deck)
- When solving it, **justify each step** in your solution

Review #2 Content

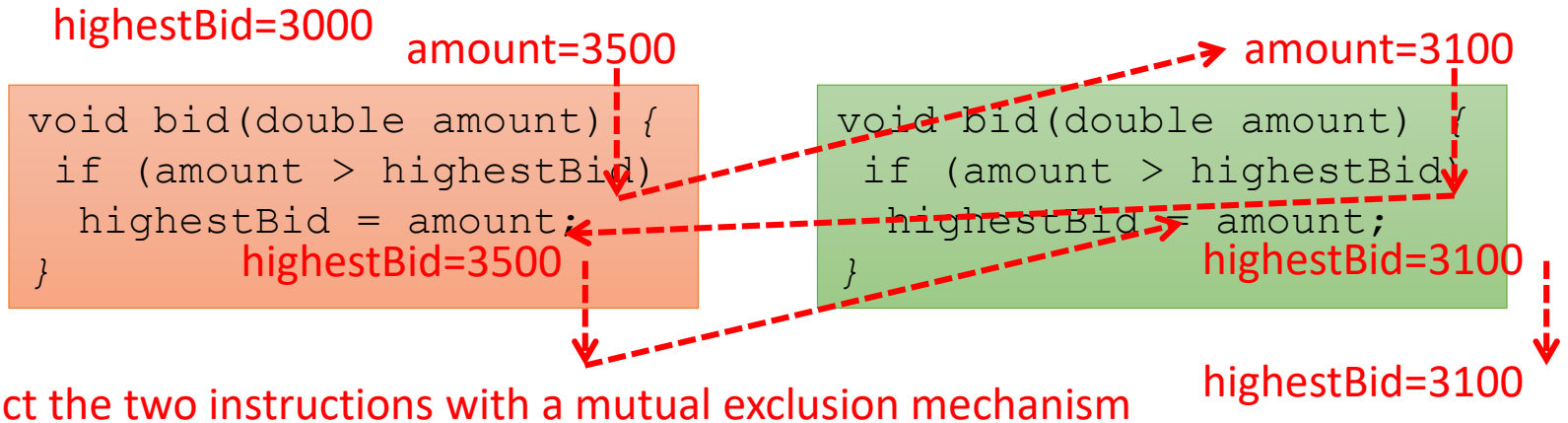
- Synchronization
- Deadlocks
- Memory
- Virtual Memory
- Disk
- File System

Synchronization (Chapter 6) #1

- **6.8** Race conditions are possible in many computer systems. Consider an online auction system where the current highest bid for each item must be maintained. A person who wishes to bid on an item calls the bid(amount) function, which compares the amount being bid to the current highest bid. If the amount exceeds the current highest bid, the highest bid is set to the new amount. This is illustrated below:

```
void bid(double amount) {
 if (amount > highestBid)
  highestBid = amount;
}
```

Describe how a race condition is possible in this situation and what might be done to prevent the race condition from occurring.



Synchronization (Chapter 6) #2

- **6.18** The implementation of mutex locks provided in Section 6.5 suffers from busy waiting. Describe what changes would be necessary so that a process waiting to acquire a mutex lock would be blocked and placed into a waiting queue until the lock became available.

The definition of acquire() is as follows:

```
acquire() {
  while (!available)
    ; /* busy wait */
  available = false;
}
```

The definition of release() is as follows:

```
release() {
  available = true;
}
```

Check if it is available
If not, put yourself in a waiting queue and sleep

Check if anyone is in waiting queue
If yes, wake at least one waiter

Idea: be inspired by semaphores and monitors

Synchronization (Chapter 6) #3

- **6.24** In Section 6.7, we use the following illustration as an incorrect use of semaphores to solve the critical-section problem:

  wait(mutex);

  ...

  critical section

  ...

  wait(mutex);

  Explain why this is an example of a liveness failure.

Liveness refers to a set of properties that a system must satisfy to ensure that processes make progress during their execution life cycle.

wait(mutex); <-------- We assume everywhere else the use of mutex is correct (no liveness issues)

...

critical section — We correctly enter the critical section

...

wait(mutex); <-------- We are in the critical section and we are trying to enter the critical section again -- deadlock

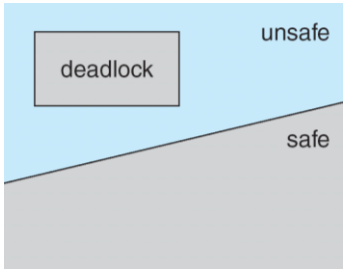
Synchronization (Chapter 6) Equivalent APIs

- To **enter** critical section
  - down()
  - lock()
  - acquire()
  - wait()
- To **exit** critical section
  - up()
  - unlock()
  - release()
  - signal()

Deadlocks (Chapter 8) #1

- **8.2** Suppose that a system is in an unsafe state. Show that it is possible for the threads to complete their execution without entering a deadlocked state.

Safe State: If the system **can allocate resources** to each thread/process (up to its maximum) **in some order** and still **avoid deadlocks**

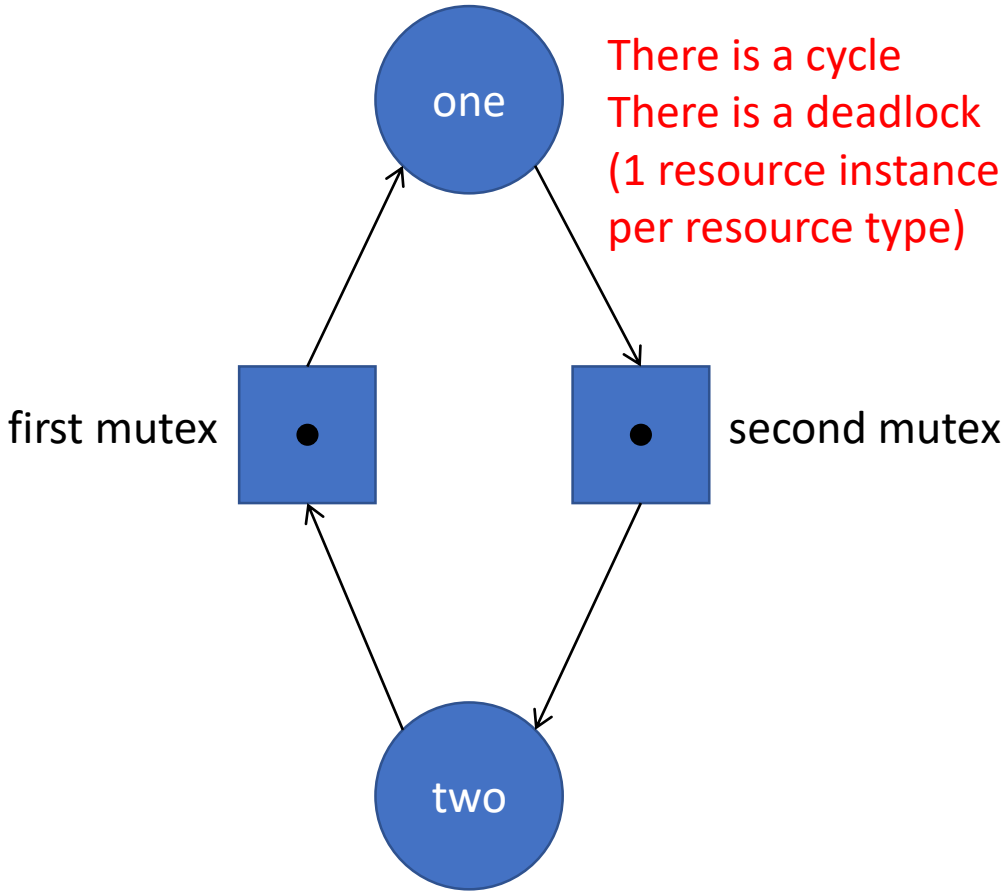


Idea: find a sequence that doesn't allocate up to its maximum

Deadlocks (Chapter 8) #2

- **8.13** Draw the resource-allocation graph that illustrates deadlock from the program example shown in Figure 8.1 in Section 8.2.

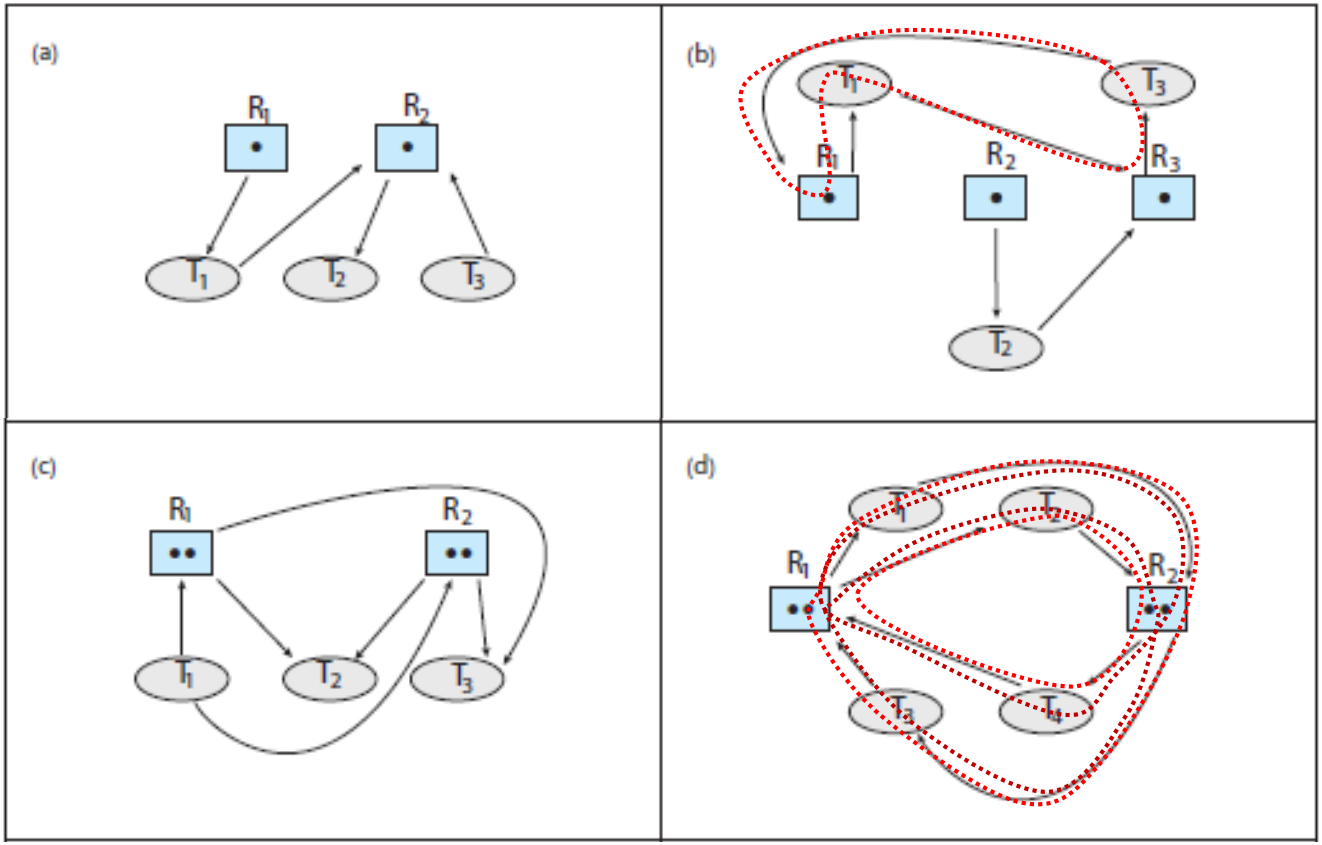
```
/* thread one runs in this function */
void *do work one(void *param)
{
pthread mutex lock(&first mutex);
pthread mutex lock(&second mutex);
/**
* Do some work
*/
pthread mutex unlock(& second mutex);
pthread mutex unlock(& first mutex);
pthread exit(0);
}
/* thread two runs in this function */
void *do work two(void *param)
{
pthread mutex lock(&second mutex);
pthread mutex lock(&first mutex);
/**
* Do some work
*/
pthread mutex unlock(&first mutex);
pthread mutex unlock(&second mutex);
pthread exit(0);
}
```



Deadlocks (Chapter 8) #3

- **8.18** Which of the six resource-allocation graphs shown in Figure 8.12 illustrate deadlock? For those situations that are deadlocked, provide the cycle of threads and resources. Where there is not a deadlock situation, illustrate the order in which the threads may complete execution.

No deadlock (no cycle) T2>T3>T1



Deadlock (cycle)

No deadlock (no cycle) T2>T3>T1

Deadlock (cycles)

Memory (Chapter 9) #1

- **9.4** Consider a logical address space of 64 pages of 1,024 words each, mapped onto a physical memory of 32 frames.
  - a. How many bits are there in the logical address?
  - b. How many bits are there in the physical address?

Assume 1 word == 4 bytes
64 pages * 1024 words/page * 4 bytes/word = 262144 bytes (256kB)
32 frames * 1024 words/frame * 4 bytes/word = 131072 byts (128kB)

a. log2(262144) = 18
b. log2(131072) = 17

Memory (Chapter 9) #2

- **9.7** Assuming a 1-KB page size, what are the page numbers and offsets for the following address references (provided as decimal numbers):
  - a. 3085
  - b. 42095
  - c. 215201
  - d. 650000
  - e. 2000001

1kB page size == 1024 bytes

a. Page number is 3085/1024 = 3 – offset is 3085 - (1024*3) = 13
b. Page number is 42095/1024 = 41 – offset is 42095 – (1024*41) = 111
c. …

Memory (Chapter 9) #3

- **9.16** On a system with paging, a process cannot access memory that it does not own. Why? How could the operating system allow access to additional memory? Why should it or should it not?

Paging is used to limit each process to access its own memory only – each process accesses a different set of memory frames.

The OS may let multiple processes to access a set of page frames.

This is necessary to let two different processes communicate via shared memory.

Virtual Memory (Chapter 10) #1

- **10.2** Assume that you have a page-reference string for a process with *m* frames (initially all empty). The page-reference string has length *p*, and *n* distinct page numbers occur in it. Answer these questions for any page-replacement algorithms:
  - a. What is a lower bound on the number of page faults?
  - b. What is an upper bound on the number of page faults?

a. The lower bound is a function of the number of available frames.

b. The upper bound is the page-reference string length (fault for each reference).

Virtual Memory (Chapter 10) #2

- **10.5** Consider the page table for a system with 12-bit virtual and physical addresses and 256-byte pages.
  The list of free page frames is *D*, *E*, *F* (that is, *D* is at the head of the list, *E* is second, and *F* is last). A dash for a page frame indicates that the page is not in memory. Convert the following virtual addresses to their equivalent physical addresses in hexadecimal. All numbers are given in hexadecimal.
  - 9EF
  - 111
  - 700
  - 0FF

Page	Page Frame
0	–
1	2
2	C
3	A
4	–
5	4
6	3
7	–
8	B
9	0

>>> 0x9EF = 2543
Address at page 2543/256 = 9; offset 2543 – (256*9) = 239
Physical address = 0*256 + 239 = 239
>>> 0x111 = 273
Address at page 273/256 = 1; offset 273 – (256*1) = 17
Physical address = 2*256 + 17 = 529
…

Virtual Memory (Chapter 10) #3

- **10.9** Consider the following page reference string:
  7, 2, 3, 1, 2, 5, 3, 4, 6, 7, 7, 1, 0, 5, 4, 6, 2, 3, 0 , 1.
  Assuming demand paging with three frames, how many page faults would occur for the following replacement algorithms?
  - LRU replacement
  - FIFO replacement
  - Optimal replacement

LRU

fr	7	2	3	1	2	5	3	4	6	7	7	1	0	5	4	6	2	3	0	1
0	7	7	7	1		1	3	3	3	7		7	7	5	5	5	2	2	2	1
1		2	2	2		2	2	4	4	4		1	1	1	4	4	4	3	3	3
2			3	3		5	5	5	6	6		6	0	0	0	6	6	6	0	0
	f	f	f	f		f	f	f	f	f		f	f	f	f	f	f	f	f	f

18 page faults

…

Disk (Chapter 11) #1

- **11.1** Is disk scheduling, other than FCFS scheduling, useful in a single-user environment? Explain your answer.

Idea: in a single-user environment all requests are generated by a user; hence, in principle, there is no needed for an algorithm that is more fair.

Disk (Chapter 11) #2

- **11.12** Explain why NVM devices often use an FCFS disk-scheduling algorithm.

Idea: NVM devices have no moving parts, each block can be accessed at the same cost.

Disk (Chapter 11) #3

- **11.13** Suppose that a disk drive has 5,000 cylinders, numbered 0 to 4,999. The drive is currently serving a request at cylinder 2,150, and the previous request was at cylinder 1,805. The queue of pending requests, in FIFO order, is:
  2,069; 1,212; 2,296; 2,800; 544; 1,618; 356; 1,523; 4,965; 3,681
  Starting from the current head position, what is the total distance (in cylinders) that the disk arm moves to satisfy all the pending requests for each of the following disk-scheduling algorithms?
  - a. FCFS
  - b. SCAN
  - c. C-SCAN

a. FCFS **order** is 1805 > 2150 > 2069 > 1212 > 2296 > 2800 > 544 > 1618 > 356 > 1523 > 4965 > 3681
   the total **distance** is (2150-1805) + (2150-2069) + (2069-1212) + (2296-1212) + (2800-2296) + (2800-544) + (1618-544) + (1618-356) + (1523-356) + (4965-1523) + (4965-3681) = 13356

b. SCAN **order** is 1805 > 2150 > 2296 > 2800 > 3681 > 4965 > 2069 > 1618 > 1523 > 1212 > 544 > 356
   …

File System (Chapter 13) #1

- **13.4** Could you simulate a multilevel directory structure with a single-level directory structure in which arbitrarily long names can be used? If your answer is yes, explain how you can do so, and contrast this scheme with the multilevel directory scheme. If your answer is no, explain what prevents your simulation's success. How would your answer change if file names were limited to seven characters?

Something like this **/what/a/wonderful/day/isToday.txt** can be converted to **/what_a_wonderful_day_isToday.txt** – thus, the answer is yes. One character must be used for the simulation.

If the number of chars are limited, the simulation is limited to what 7 chars can represent.

File System (Chapter 13) #2

- **13.7** Consider a system that supports 5,000 users. Suppose that you want to allow 4,990 of these users to be able to access one file.
  - a. How would you specify this protection scheme in UNIX?
  - b. Can you suggest another protection scheme that can be used more effectively for this purpose than the scheme provided by UNIX?

Remember: Unix uses ACL, for each file, there is a list of users and their permissions. However, users are organized in groups, hence each file has permissions specified, for the file owner, group owner, and others.

Solution.a: put 4990 into the same group and allow the group to access the file.

Solution.b: the orthogonal solution is to provide each user with the list of files that he/she can access, in this case this solution may be better.

File System (Chapter 13) #3

- **13.8** Researchers have suggested that, instead of having an access-control list associated with each file (specifying which users can access the file, and how), we should have a **user control list** associated with each user (specifying which files a user can access, and how). Discuss the relative merits of these two schemes.

Check previous problem.

File System (Chapter 14) #1

- **14.8** Contrast the performance of the three techniques for allocating disk blocks (contiguous, linked, and indexed) for both sequential and random file access.

Contiguous: random access is very efficient – knowing the start address of the file you just need to add an offset.

Linked: it is not efficient – it is necessary to walk the linked list, in the worst case, the entire linked list should be walked.

Indexed (direct ptr only): quite efficient – the index node must be read first, then the specific block can be fetched.

File System (Chapter 14) #2

- **14.9** What are the advantages of the variant of linked allocation that uses a FAT to chain together the blocks of a file?

Idea: FAT keeps the per-file "linked chain" of blocks id in memory, this speeds up random access.

File System (Chapter 14) #3

- **14.15** Consider a file system that uses inodes to represent files. Disk blocks are 8 KB in size, and a pointer to a disk block requires 4 bytes. This file system has 12 direct disk blocks, as well as single, double, and triple indirect disk blocks. What is the maximum size of a file that can be stored in this file system?

(check next slide before proceeding)

12 direct, 1 single indirect, 1 double indirect, 1 triple indirect

12 direct for a total size of 12 * 8kB = 96kB
1 single indirect for a total of (8kB / 4B) * 8kB = 16MB
1 double indirect for a total of (8kB / 4B) * (8kB / 4B) * 8kB = 32GB
1 triple indirect for a total of (8kB / 4B) * (8kB / 4B) * (8kB / 4B) * 8kB = 64TB
Total is 96kB + 16MB + 32GB + 64TB = **~64TB**

(However, this exercise is wrong because you cannot address 8G with 4bytes)

File System (Chapter 14)
inode structure