Information page for written examinations at Linköping University

Examination date	2019-03-23
Room (5)	G32(35) G34(6) TER3(28) TER4(104) TERE(4)
Time	14-18
Edu. code	TDDB68
Module	TEN1
Edu. code name **Module name**	Concurrent Programming and Operating Systems (Processprogrammering och operativsystem) Examination (Tentamen)
Department	IDA
Number of questions in the examination	6
Teacher responsible/contact person during the exam time	Mikael Asplund
Contact number during the exam time	0700 895 827
Visit to the examination room approximately	15-16
Name and contact details to the course administrator (name + phone nr + mail)	Veronica Kindeland Gunnarsson 013-28 56 34 adm-gu@ida.liu.se
Equipment permitted	Engelsk ordbok / english dictionary, Miniräknare / pocket calculator
Other important information	
Number of exams in the bag	

Information page for written examinations at Linköping University

Examination date	2019-03-23
Room (5)	G32(35) G34(6) TER3(28) TER4(104) TERE(4)
Time	14-18
Edu. code	TDDB68
Module	TEN1
Edu. code name **Module name**	Concurrent Programming and Operating Systems (Processprogrammering och operativsystem) Examination (Tentamen)
Department	IDA
Number of questions in the examination	6
Teacher responsible/contact person during the exam time	Mikael Asplund
Contact number during the exam time	0700 895 827
Visit to the examination room approximately	15-16
Name and contact details to the course administrator (name + phone nr + mail)	Veronica Kindeland Gunnarsson 013-28 56 34 adm-gu@ida.liu.se
Equipment permitted	Engelsk ordbok / english dictionary, Miniräknare / pocket calculator
Other important information	
Number of exams in the bag	

Information page for written examinations at Linköping University

Examination date	2019-03-23
Room (5)	G32(35) G34(6) TER3(28) TER4(104) TERE(4)
Time	14-18
Edu. code	TDDB68
Module	TEN1
Edu. code name **Module name**	Concurrent Programming and Operating Systems (Processprogrammering och operativsystem) Examination (Tentamen)
Department	IDA
Number of questions in the examination	6
Teacher responsible/contact person during the exam time	Mikael Asplund
Contact number during the exam time	0700 895 827
Visit to the examination room approximately	15-16
Name and contact details to the course administrator (name + phone nr + mail)	Veronica Kindeland Gunnarsson 013-28 56 34 adm-gu@ida.liu.se
Equipment permitted	Engelsk ordbok / english dictionary, Miniräknare / pocket calculator
Other important information	
Number of exams in the bag	

Information page for written examinations at Linköping University

Examination date	2019-03-23
Room (5)	G32(35) G34(6) TER3(28) TER4(104) TERE(4)
Time	14-18
Edu. code	TDDB68
Module	TEN1
Edu. code name **Module name**	Concurrent Programming and Operating Systems (Processprogrammering och operativsystem) Examination (Tentamen)
Department	IDA
Number of questions in the examination	6
Teacher responsible/contact person during the exam time	Mikael Asplund
Contact number during the exam time	0700 895 827
Visit to the examination room approximately	15-16
Name and contact details to the course administrator (name + phone nr + mail)	Veronica Kindeland Gunnarsson 013-28 56 34 adm-gu@ida.liu.se
Equipment permitted	Engelsk ordbok / english dictionary, Miniräknare / pocket calculator
Other important information	
Number of exams in the bag	

Information page for written examinations at Linköping University

Examination date	2019-03-23
Room (5)	G32(35) G34(6) TER3(28) TER4(104) TERE(4)
Time	14-18
Edu. code	TDDB68
Module	TEN1
Edu. code name **Module name**	Concurrent Programming and Operating Systems (Processprogrammering och operativsystem) Examination (Tentamen)
Department	IDA
Number of questions in the examination	6
Teacher responsible/contact person during the exam time	Mikael Asplund
Contact number during the exam time	0700 895 827
Visit to the examination room approximately	15-16
Name and contact details to the course administrator (name + phone nr + mail)	Veronica Kindeland Gunnarsson 013-28 56 34 adm-gu@ida.liu.se
Equipment permitted	Engelsk ordbok / english dictionary, Miniräknare / pocket calculator
Other important information	
Number of exams in the bag	

TENTAMEN / *EXAM*

TDDB68

Processprogrammering och operativsystem /
Concurrent programming and operating systems

2019-03-23

Examiner: Mikael Asplund (0700895827)

Hjälpmedel / *Admitted material:*

– Engelsk ordbok / *Dictionary from English to your native language*;
– Miniräknare / *Pocket calculator*

General instructions

- This exam has 6 assignments and 9 pages, including this one.
  Read all assignments carefully and completely before you begin.
- Please **use a new sheet of paper for each assignment**, because they will be corrected by different persons.
  Sort the pages by assignments and number them consecutively.
- You may answer in either English or Swedish. English is preferred because not all correcting assistants understand Swedish.
- Write clearly. Unreadable text will be ignored.
- Be precise in your statements. Unprecise formulations may lead to a reduction of points.
- **Motivate clearly** all statements and reasoning.
- Explain calculations and solution procedures.
- The assignments are *not* ordered according to difficulty.
- The exam is designed for 40 points. You may thus plan about 5 minutes per point.
- In order to improve this and other courses on concurrency, we will anonymously record the frequency of different types of answers to exercise 2 while correcting the exam.
- Preliminary grading thresholds:
  - 3: at least 20p
  - 4: at least 27p
  - 5: at least 34p

Good luck!

1. **Multiple choice quesions** (10p)

Below are 10 multiple choice questions. Please answer them by removing the last page of the exam, fill the appropriate boxes, and hand it in together with the rest of your answers. Please note that there might be more than one correct option. Each question below can give 0 or 1 point(s), and to get 1 point you must have identified exactly the right set of choices.

(a) Which of the following information would typically be stored in the Process Control Block (PCB) for a process? (1p)

A) The process state
B) The process file
C) The program counter
D) The process ID

(b) Which of the following memory management tasks can be performed by the MMU: (1p)

A) Memory protection
B) Page table lookup
C) Page replacement
D) TLB lookup

(c) How many times will the following program print out "PID: 0"? (1p)

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main() {
    pid_t pid = fork();
    printf("PID: %d\n", pid);
    pid = fork();
    printf("PID: %d\n", pid);
    return 0;
}
```

A) 1
B) 2
C) 3
D) 4

(d) Which of the following are classical scheduling algorithms (as described in the course literature): (1p)

A) Shortest-job first scheduling
B) Burst-first scheduling
C) Round-robin scheduling
D) Priority scheduling

(e) Which of the following can be said to constitute vulnerabilities (1p)

A) Lack of input validation in a program
B) An email with a virus attached
C) The Android operating system
D) Clear-text storage of passwords

(f) Given a virtual memory system with 4 page frames, how many page faults occur with the *Least-Recently Used* replacement strategy when pages are accessed in the following order: (1p)

2, 3, 1, 4, 5, 1, 3, 2, 2, 3, 1, 5, 4.

A) 7
B) 9
C) 11
D) 13

(g) Which of the following properties are true for semaphores? (1p)

A) The wait and signal operations are atomic.
B) They guarantee freedom from deadlock.
C) They can only be implemented on a system with special hardware.
D) They can be used to solve the critical section problem.

(h) Which of the following allocation methods for blocks on a disk can guarantee access to the middle of a file using at most two disk operations (not considering potential cache functionality)? You can assume that the directory information is already loaded in memory. (1p)

A) Contiguous allocation
B) Linked allocation
C) Indexed allocation
D) Multi-level indexed allocation

(i) Which of the following are possible states for a system process (as described in the course literature): (1p)

A) Waiting
B) Ready
C) Forking
D) Controlling

(j) Which of the following are security *attributes*? (1p)

A) Integrity

B) Authentication

C) Protection

D) Availability

2. **Synchronization** (11p)

As a teacher, you are constantly on the hunt for good ideas for exam exercises. The main problem, however, is that it is easy to forget the good ideas before they are actually used to produce a good question. To solve this problem, one teacher implemented a data structure to keep track of them. The implementation of the data structure is shown in Listing 1 on page 6. It has the following operations:

- `idea_init`: Initializes the idea buffer.
- `idea_add`: Adds an idea (a string) to the buffer. If the buffer is full and the idea could not be added `false` should be returned, otherwise `true` should be returned.
- `idea_get`: Randomly selects and returns an idea from the buffer. The idea is also removed to ensure it is not used for another exam. If no ideas are present, `idea_get` shall wait until a new idea is added with `idea_add`.

During the exam periods, `idea_add` and `idea_get` are used frequently by many teachers. Therefore, it is important that they are usable from multiple threads simultaneously as far as possible.

(a) Is *busy-wait* used somewhere in the implementation? If so, where? (1p)

(b) Use suitable synchronization primitives from listing 2 on page 7 to eliminate any occurrences of *busy-wait* you found. (2p)

(c) After using the data structure for a while, some users notice that the same idea has been used multiple times (i.e. multiple calls to `idea_get` returned the same idea). Furthermore, ideas sometimes disappear from the buffer, even though `idea_add` indicates success by returning `true`. (2p)

Explain with an example what could have happened when...

A) ...the same idea was used multiple times.

B) ...the buffer "lost" one or more ideas.

(d) Mark any critical sections present in the functions `idea_add` and `idea_get`. Also note the resource(s) that need protection. (2p)

(e) Use suitable synchronization primitives from listing 2 on page 7 to synchronize the code based on the critical sections you found. (4p)

Note: Strive for a solution that allows maximum theoretical parallellism, even though that solution may perform worse in practice due to synchronization overheads (please note if you think this is the case).

Note: Points may be deducted for excessive locking

3. **Deadlocks** (6p)

(a) Explain the principle of deadlock avoidance, including the concept of safe states. (2p)

(b) Consider the following resource allocation problem in a system with 3 resources (R1-R3), and 4 processes (P1-P4). The table indicates the currently allocated resources and in parenthesis the maximum possible demand. (4p)

	R1	R2	R3
P1	1 (1)	0 (3)	1 (1)
P2	0 (2)	0 (1)	1 (4)
P3	1 (1)	0 (0)	1 (1)
P4	0 (1)	0 (3)	0 (1)

The currently available resources are: [0, 3, 5]. Use Banker's algorithm to determine if the request [0, 1, 0] from Process P4 should be granted.

4. **Processes and scheduling** (3p)

(a) Given a single-CPU system and the following set of processes with arrival times (in milliseconds), expected maximum execution time (ms), and priority (1 is **lowest**, 5 is **highest** priority). (3p)

Process	Arrival time	Execution time	Priority (as applicable)
P_1	0	3	1
P_2	1	2	3
P_3	2	5	2
P_4	4	1	4
P_5	6	3	5

For each of the following scheduling algorithms, create a Gantt chart (time bar diagram, starting at $t = 0$) that shows when the processes will execute on the CPU. Where applicable, the time quantum will be 2 ms. Assume that a task will be eligible for scheduling immediately on arrival. If you need to make further assumptions, state them carefully and explain your solution.

(i) Round-robin;

(ii) Shortest Job First *with* preemption;

(iii) Priority Scheduling *without* preemption.

5. **Memory management** (6p)

(a) Compare paging with segmentation with respect to fragmentation and how much memory the address translation structures require to convert virtual addresses to physical addresses. (4p)

(b) Explain how the possibility of sharing memory pages among multiple processes can help to speed up the start-up of a child process at a `fork` system call. (2p)

6. **File systems** (4p)

(a) What is/are the main (technical) purpose(s) of *opening* a file? What kernel data structures does the `open()` system call manipulate and how, and what does it return? (2p)

(b) Consider a file `hello.txt` which is owned by the (non-administrator) user Mallory. In a normal system, Mallory would not be able to edit the indode of `hello.txt`, but the system which she is using has a weakness that allows her to edit the inode. Using (2p) terminology from the security domain, what is such a weakness called, and what harm can be done by Mallory being able to modify the inode contents?

```c
#define BUFFER_SIZE 32

struct idea_buffer {
  // All ideas in the buffer. Empty elements are set to NULL.
  const char *ideas[BUFFER_SIZE];
};

// Initialize the buffer.
void idea_init(struct idea_buffer *buffer) {
  for (int i = 0; i < BUFFER_SIZE; i++)
    buffer->ideas[i] = NULL;
}
// Add a new idea to an empty location in the buffer. Returns
// 'false' if the buffer is full.
bool idea_add(struct idea_buffer *buffer, const char *idea) {
  // Find an empty location.
  int found = BUFFER_SIZE;
  for (int i = 0; i < BUFFER_SIZE; i++) {
    if (buffer->ideas[i] == NULL) {
      //Empty location found, put an idea in it an return success
      found = i;
      buffer->ideas[found] = idea;
      return true;
    }
  }
  //Buffer full
  return false;
}

// Get and remove a random element from the buffer. If no elements
// are present, the function waits for an element to be added.
const char *idea_get(struct idea_buffer *buffer) {
  // Find an element. Start from a random index, and look through
  // the array until we find a non-empty element.
  int pos = rand() % BUFFER_SIZE;
  while (buffer->ideas[pos] == NULL) {
    pos = (pos + 1) % BUFFER_SIZE;
  }
  // Remove it.
  const char *result = buffer->ideas[pos];
  buffer->ideas[pos] = NULL;

  return result;
}
```

Listing 1: Implementation of the idea buffer

Available synchronization primitives

```c
struct semaphore;
void sema_init(struct semaphore *sema, unsigned value);
void sema_signal(struct semaphore *sema);
void sema_wait(struct semaphore *sema);

struct lock;
void lock_init(struct lock *lock);
void lock_acquire(struct lock *lock);
void lock_release(struct lock *lock);
```

Listing 2: Syncronization primitives

Intentionally empty page.

Multiple choice form for answering question 1. Please put X:s in the appropriate cells:

	A	B	C	D
1 a)				
1 b)				
1 c)				
1 d)				
1 e)				
1 f)				
1 g)				
1 h)				
1 i)				
1 j)				

Optional: if you feel you need to clarify your interpretation of the question you can do so here. This is not needed if your answer is correct.

1 a)	
1 b)	
1 c)	
1 d)	
1 e)	
1 f)	
1 g)	
1 h)	
1 i)	
1 j)	