TENTAMEN / *EXAM*

TDDB68

Processprogrammering och operativsystem /
Concurrent programming and operating systems

2017-06-07, 14:00–18:00

Jour: Mikael Asplund (0700895827); visiting ca. 15:30

Hjälpmedel / *Admitted material:*

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

General instructions

- This exam has 6 assignments and 10 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.
- **How much to write?** No general policy, but as a rule of thumb: Questions for 0.5p can typically be answered properly in a single line. Correct and concise answers to questions for 1p usually require a few lines. Code and figures should be commented properly.
- Grading: U, 3, 4, 5. The preliminary threshold for passing is 20 points.

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 events will *always* trigger an interrupt (hardware or software).

   A) A user program requesting to read data from a file.

   B) A user program requesting to read data from main memory.

   C) A key being pressed on the keyboard.

   D) A byte being transferred from disk to main memory.

   (b) Consider the program below. How many times will it output the line "Hello"?

```
#include <stdio.h>
#include <unistd.h>
int main()
{
  printf("Hello\n");
  fork();
  fork();
  printf("Hello\n");

  return 0;
}
```

   A) 2 times

   B) 3 times

   C) 5 times

   D) 7 times

(c) Which of the following statements are true about deadlocks:

A) If there is only a single instance of every resource, a cycle in the resource allocation graph means that there is a deadlock.

B) All four Coffman conditions must me met for there to be a deadlock.

C) Banker's algorithm is used to detect and remove deadlocks.

D) Banker's algorithm guarantees freedom from starvation.

(d) Which of of the following events will take place when switching from one process (old) to another (new).

A) Changing the set of currently open files to match with the new process.

B) Copying data from the process control block (PCB) of the new process into the interrupt vector table.

C) Storing CPU register states into the PCB of the old process.

D) Updating registers relating to memory management (e.g., base address pointer).

(e) Which of the following memory management tasks can be performed by the MMU:

A) Memory protection

B) Page table lookup

C) TLB lookup

D) Page replacement

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

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

A) 7

B) 9

C) 11

D) 13

(g) The following is an excerpt from the Wikipedia article on the WannaCry ransomware (accessed 2017-06-01):

> The WannaCry ransomware attack was a worldwide cyberattack by the WannaCry ransomware cryptoworm, which targeted computers running the Microsoft Windows operating system by encrypting data and demanding ransom payments in the Bitcoin cryptocurrency.
> The attack began on Friday, 12 May 2017, and within a day was reported to have infected more than 230,000 computers in over 150 countries. Parts of Britain's National Health Service (NHS), Spain's Telefonica, FedEx and Deutsche Bahn were hit, along with many other countries and companies worldwide. Shortly after the attack began, a web security researcher who blogs as "MalwareTech" discovered an effective kill switch by registering a domain name he found in the code of the ransomware. This greatly slowed the spread of the infection, effectively halting the initial outbreak on Monday, 15 May 2017, but new versions have since been detected that lack the kill switch. Researchers have also found ways to recover data from infected machines under some circumstances.

> WannaCry propagates using EternalBlue, an exploit of Windows' Server Message Block (SMB) protocol. Much of the attention and comment around the event was occasioned by the fact that the U.S. National Security Agency (NSA) had discovered the vulnerability in the past, but used it to create an exploit for its own offensive work, rather than report it to Microsoft. It was only when the existence of this vulnerability was revealed by The Shadow Brokers that Microsoft became aware of the issue, and issued a "critical" security patch on 14 March 2017 to remove the underlying vulnerability on supported versions of Windows. but many organizations had not yet applied it.

Which of the following protection measures could have reduced the impact of this attack?

A) Two-factor authentication
B) Faster updates of critical security patches
C) Proper backup procedures
D) Encryption

(h) Consider the following code snippet (adapted from thegeekstuff.com):

```
#include <stdio.h>
#include <string.h>

int main(void) {
    char buff[15];
    int pass = 0;

    printf("\n Enter password : \n");
    gets(buff);

    if(strcmp(buff, "secret")) {
        printf ("\n Wrong Password \n");
    } else {
        printf ("\n Correct Password \n");
        pass = 1;
    }

    if(pass) {
        printf ("\n Access granted\n");
    }

    return 0;
}
```

What are the *possible* outcomes of entering a string of more than 15 characters into the program?

A) Nothing in particular, the program outputs "Wrong Password".
B) The program outputs "Wrong Password" and then "Access granted".

C) Stack protection mechanisms detects a buffer overflow and terminates the program.

D) The gets function will not accept a string that exceeds the buffer size so it will throw an exception.

(i) Which of the following virtualization configurations are possible when using *paravirtualization*:

A) Host running Ubuntu Linux on a x86-64 machine, Guest running Windows 10 for x86-64

B) Host running Windows 10 on a x86-64 machine, Guest running Ubuntu Linux for x86-64

C) Host running Ubuntu Linux on a x86-64 machine, Guest running Linux for ARM

D) Host running Ubuntu Linux on an IA-32 machine, Guest running Ubuntu Linux for x86-64

(j) Wihch of the following mechanisms reduce the impact of buffer overflow attacks?

A) Marking certain parts of the data as non-executable (e.g., non-executable stack, non-executable pages)

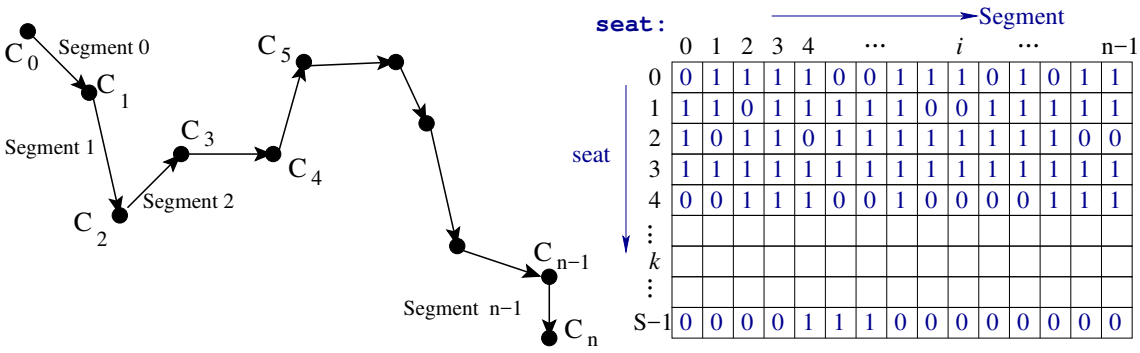
B) Arrays with built-in bound checks

C) Making sure that the source and destination buffers are of appropriate length when copying memory

D) Validate user input before processing it

2. **Synchronization (7.5p)**

A train with S seats goes from city C_0 to city C_n with stops in C_1, C_2, ..., C_{n-1} (see the figure). *Segment* i is the direct connection between cities C_i and C_{i+1}, for $0 \leq i \leq n-1$.



The railway company allows to book seats for any partial contiguous subsequence of segments $i...j$ with $0 \leq i \leq j \leq n$. Booked seats can also be canceled.

For this purpose, a server program had been written. As main data structure representing the S seats over all n segments, a two-dimensional integer array `seat`$[S][n]$ is used, where `seat`$[k][i]$ is 1 if seat k is booked for the segment i, and 0 otherwise (see the figure). Initially, all array entries are set to 0 before any booking requests are accepted for processing.

The following C function implements a first-fit algorithm that searches for a free seat from segment i to j (precondition: $0 \leq i \leq j < n$), and either books and returns the booked seat number if one was available, or returns -1 otherwise:

```
int book ( unsigned int i, unsigned int j )
{
 int k, t;
 for (k=0; k<S; k++) { // try all seats
    int found_k = 1;
    for (t=i; t<=j; t++)
       if (seat[k][t]) {  // if seat k is taken anywhere in i...j,
          found_k = 0;    // we cannot book it.
          break;          // try next k
       }
    if (found_k) {        // seat k was available for i...j:
       for (t=i; t<=j; t++)  // now book it
          seat[k][t] = 1;
       return k;
    }
 }
 // if we arrive here, no seat was available for i...j:
 return -1;
}
```

Canceling a seat for segments $i...j$ (again with precondition $0 \leq i \leq j < n$) is done by the following routine:

```
void cancel ( unsigned int k, unsigned int i, unsigned int j )
{
 int t;
 for (t=i; t<=j; t++)
    seat[k][t] = 0;
}
```

The railway company has now acquired a new multiprocessor server. The entire seat booking program (not shown) is being multithreaded so that it can concurrently process multiple requests for booking or canceling that come in from various clients. The above `seat` data structure is to be held in (shared) memory, and it is your job to make the accesses to it (i.e., functions `book` and `cancel`) thread-safe, which means that concurrent calls do not lead to erroneous behavior such as double bookings of a seat.

(a) Give a concrete example that demonstrates how, without proper synchronization, two concurrent calls to `book` could lead to a double booking of the same seat. (1p)

(b) Identify the critical section(s) in the functions `book` and `cancel`. (1.5p)

(c) Protect the code against race conditions with a single mutex lock. (1p)

(d) Assume that the rate of incoming concurrent calls to `book` and `cancel` is very high. Design a more fine-grained synchronization mechanism for the above data structure (i.e., try to make the critical section(s) short). Show the resulting code.

Explain how this modification can improve system throughput in comparison to a coarse-grained synchronization approach. (2.5p)

(e) What is a *reader-writer lock*?

Under what conditions are reader-writer locks more suitable than ordinary mutual exclusion locks in order to increase system throughput? (1.5p)

3. **Deadlocks (7p)**

(a) **Mutex lock with self-deadlock prevention**

A common synchronization error with mutex locks in concurrent programs is a forgotten lock release (V, signal) operation, i.e., some process/thread does not release a lock and gets deadlocked when it tries to acquire the same lock next time (because it is still holding it itself without "knowing" that).

Design an extended mutex lock data structure that prevents such self-deadlocks. Show the code for the acquire (=wait) and release (=signal) functions, and explain why self-deadlocks are no longer possible. (3p)

(Hint: You will need an extra field in the revised mutex lock data structure, beyond the binary semaphore variable itself. What do you store in that extra field, and when do you store and when do you read it?)

(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.

	R1	R2	R3
P1	0 (4)	0 (1)	0 (1)
P2	3 (3)	3 (3)	4 (6)
P3	0 (3)	0 (0)	1 (2)
P4	0 (0)	0 (0)	0 (1)

The currently available resources are: [1, 0, 2]. Use Banker's algorithm to determine if the request [1, 0, 1] from Process P1 should be granted.

(4p)

4. **Processes and scheduling (5p)**

(a) What is a *process control block* (PCB), what is its purpose, and what information does it contain? (2p)

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

Process	Arrival time	Execution time	Priority (as applicable)
P_1	0	3	1
P_2	2	4	2
P_3	4	2	3
P_4	7	1	4
P_5	9	2	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) FIFO;

(ii) Round-robin;

(iii) Priority Scheduling *with* preemption.

(3p)

5. **Memory Management (5.5p)**

   (a) Consider a page-based virtual memory system with a page size of $2^{10} = 1024$ bytes where virtual memory addresses have 32 bit. If using *multi-level paging*, determine how many levels of paging are required (assuming each table must fit in one page), and describe the structure of the virtual addresses (purpose, position and size of its bit fields). (2p)

   (b) Explain (annotated figure) how the physical address from (a) is calculated by multi-level paging from a virtual address. (2p)

   (c) In what two ways do paging systems incur fragmentation? (1.5p)

6. **File systems (5p)**

   (a) Explain the concept of an allocation method in the context of filesystem implementation. Describe at least two different forms of allocation. (3p)

   (b) 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)

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)				