Försättsblad till skriftlig tentamen vid Linköpings universitet

Datum för tentamen	2019-06-11
Sal (1)	TER2(15)
Tid	14-18
Utb. kod	TDIU11
Modul	TEN1
Utb. kodnamn/benämning **Modulnamn/benämning**	Operativsystem Skriftlig tentamen
Institution	IDA
Antal uppgifter som ingår i tentamen	5
Jour/Kursansvarig Ange vem som besöker salen	Ahmed Rezine
Telefon under skrivtiden	013 - 28 19 38
Besöker salen ca klockan	
Kursadministratör/kontaktperson (namn + tfnr + mailaddress)	Anna Grabska Eklund 013-28 23 62 anna.grabska.eklund@liu.se
Tillåtna hjälpmedel	Ordbok, engelska till valfritt språk.
Övrigt	
Antal exemplar i påsen	

TDIU11 Exam

Ahmed Rezine

2019-06-11 14-18

Admitted material

Dictionary from English to your chosen language.

Jour

Ahmed Rezine (013-281938).

Instructions

- Fill in the exam wrapper and read the instructions on it before you start.
  Read instructions and all assignments carefully and completely before you begin.
- You may answer in either English or Swedish.
- State your interpretation of the question and all assumptions you make.
- Write clearly. **Unreadable text will be ignored.**
- Be precise in your statements. **Prove your point when possible.**
  Ambiguous formulations will lead to reduction of points.
- **Motivate clearly and in depth all statements and reasoning.**
  Explain calculations and solution procedures.
- The exam is 40 points and graded U, 3, 4, 5 (**preliminary** limits: 21p, 31p, 36p).
  Points are given for motivations, explanations, and reasoning.

Definitions

Unless you are more specific, the correcting team will interpret the following terms as follow:

memory Volatile random access memory (DRAM), about 100ns access time.

disk Permanent storage, about 10ms access time.

page A fix size region of virtual memory, possibly on disk.

frame A fix size region of physical memory (DRAM).

block A data block located on disk.

Problem 1 (12p)

Part A. Assume a paged virtual memory with 16 bits for both logical addresses and physical addresses. Pages are 512 bytes (2^9B) in size.

1. How many pages can a process address? (2p)
2. Assume one level paging. How large (in bytes) is a page table if we assume 2 bytes for each page entry? (1p)
3. Does it make sense to have two or more paging levels? Justify. (1p)
4. Suppose the first entries of the page table of some process associate pages to frames according to table (1). What is the physical address (in binary) corresponding to the logical address $(0000\ 1000\ 0000\ 0111)_{\texttt{binary}}$? (2p)

Part B. Assume now a paged virtual memory with 32 bits logical addresses and 4KiB (i.e., 2^{12}B) pages.

1. How many pages can a process address? (2p)
2. Assume one level paging. What is the size of a process page table assuming 4 bytes page entries? (1p)
3. Assume two levels paging. How many bits of the logical address should you allocate to each level? Justify (2p)
4. Assume your two level-solution. How many bits of a first level page entry can be used for information other than the frame of the second level page table? (1p)

frame bits	valid bit	other bits	
$(001\ 0000)_{\texttt{binary}}$	$(1)_{\texttt{binary}}$	$(\texttt{XXXX XXXX})_{\texttt{binary}}$	0^{th} page table entry
$(000\ 1000)_{\texttt{binary}}$	$(0)_{\texttt{binary}}$	$(\texttt{XXXX XXXX})_{\texttt{binary}}$	1^{st} page table entry
$(000\ 1001)_{\texttt{binary}}$	$(1)_{\texttt{binary}}$	$(\texttt{XXXX XXXX})_{\texttt{binary}}$	2^{nd} page table entry
$(000\ 0000)_{\texttt{binary}}$	$(1)_{\texttt{binary}}$	$(\texttt{XXXX XXXX})_{\texttt{binary}}$	3^{rd} page table entry
$(000\ 0011)_{\texttt{binary}}$	$(1)_{\texttt{binary}}$	$(\texttt{XXXX XXXX})_{\texttt{binary}}$	4^{th} page table entry
$(000\ 0010)_{\texttt{binary}}$	$(0)_{\texttt{binary}}$	$(\texttt{XXXX XXXX})_{\texttt{binary}}$	5^{th} page table entry
...	...	...	
$(\texttt{XXX XXXX})_{\texttt{binary}}$	$(\texttt{X})_{\texttt{binary}}$	$(\texttt{XXXX XXXX})_{\texttt{binary}}$	n^{th} page table entry

Table 1: Some entries of a process page table. The symbol X is used to mean a bit value that is not relevant for the question.

Problem 2 (4p)

1. Describe, in no more than half a page, a buffer overflow security attack allowing an attacker to execute arbitrary code on the victim machine. Do not hesitate to make use of a simple sketch to help your description. (2p)
2. How can segmentation and the underlying hardware help making such attacks more difficult? (2p)

Job	Arrival time	Execution time
J_1	0	6
J_2	1	4
J_3	2	2
J_4	5	1

Table 2: A workload to be scheduled

Problem 3 (10p)

Assume an idle system. Consider the workload depicted in Table (2).
Assume a one level scheduler using a Round Robin queue with a 4 time units time quantum.

1. Draw a Gantt diagram for processes' execution and queue contents. (2p)
2. What is the waiting time of each job? (1p)
3. Is the scheduler preemptive? Justify (1p)
4. What needs to be saved in memory in order for a process to resume execution in case it needs to execute for longer than a time quantum? (1p)

Assume a one level scheduler using Preemptive Shortest Job First (SJF).

1. Draw a Gantt diagram for processes' execution and queue contents. (2p)
2. What is the waiting time of each job? (1p)
3. General question: give an advantage of using SJF instead of a round Robin scheduler, and an advantage of using round Robing instead of SJF (2p).

Problem 4 (10p)

Consider a file system on a disk with 512 bytes physical blocks (i.e., $physical\_block\_size = 512\ bytes$) and logical blocks that are concatenations of a number of physical blocks. Assume indexed allocation with 12 direct pointers, one indirect pointer, one double indirect pointer and one triply indirect pointer.

1. What is an inode? what information should it contain? (1p)
2. Assume block pointers are 4 bytes (32 bits) long.
   a) Assume the disk is 4 TiB (2^{42} bytes) large. Let n be the number of physical blocks in a logical block. Here, each logical block is $n \times physical\_block\_size$ large. What is the smallest value of n so the block pointers can point to any portion of the disk? (1p)
   b) Assume the disk is 16 TiB (2^{44} bytes) instead. Let m be the number of physical blocks in a logical block. Here, each logical block is $m \times physical\_block\_size$ large. What is the smallest value of m so the block pointers can point to any portion of the disk? (1p)
3. Discuss advantages and disadvantages of the two logical block sizes. Consider at least two different aspects. (2p)

4. Assume block pointers are still 4 bytes long. What is the maximum size of a file given the adopted indexed allocation scheme described above with $n \times physical\_block\_size$ logical blocks? what about $m \times physical\_block\_size$ logical blocks? (2p)

5. Suppose only whole blocks may be read to / written from memory, that operations on blocks are carried in memory, that only direct pointers are used for indexed allocation, that needed inodes are already in main memory, that you do not need to account for writing inodes back to disk, and that free blocks are available with known disk positions in memory (no I/O needed to find them). We would like to add one full logical block of data between the 4^{th} and the 5^{th} logical data blocks of a file that takes up 8 logical blocks. How many I/O operations on logical blocks are required for each of: (a) contiguous allocation, (b) indexed allocation, and (c) linked allocation. (3p)

Problem 5 (4p)

A replay attack consists in an attacker listening and repeating valid data (e.g., passwords) in order to gain access to secret data. One mechanism to resist replay attacks in password authentication is to use one-time passwords. The idea is that the password is unusable by the time the attacker learned it by repeating it. For this, a list of n passwords is prepared and shared between the system and the user in a secure way. Passwords can only be used once.
Instead of sharing a big list of n passwords, a system and a user program can instead share a single "main password" pw. They can change pw after each n authentications (the mechanism for changing the main password in a secure manner is not relevant for this problem).
Describe a mechanism that allows a user program and a system to have the same list of n passwords while sharing only one single "main password" pw. Explain how your mechanism can derive the i^{th} password from the current "main password" pw. The mechanism can be public (everyone, including the attacker) knows how the mechanism works, but no attacker should be able to deduce the passwords unless it knows the main password. (hint: one way hash functions like those used to save hash functions of salted passwords)