UNIVERSITETET I BERGEN

KANDIDAT

221

PRØVE

INF113 0 Innføring i operativsystem

Emnekode	INF113
Vurderingsform	Skriftlig eksamen
Starttid	30.11.2022 14:00
Sluttid	30.11.2022 17:00
Sensurfrist	--
PDF opprettet	03.05.2024 11:01

Information

Oppgave	Tittel	Oppgavetype
ℹ	Contact Information	Informasjon eller ressurser
ℹ	General info about the exam	Informasjon eller ressurser

Simple Level

Oppgave	Tittel	Oppgavetype
1	E1	Flervalg (flere svar)
2	E2	Flervalg (flere svar)
3	E3	Sant/usant
4	E4	Sant/usant
5	E5	Fyll inn tekst
6	E6	Sant/usant
7	E7	Fyll inn tekst
8	E8	Flervalg
9	E9	Flervalg (flere svar)
10	E10	Fyll inn tekst
11	E11	Flervalg (flere svar)
12	E12	Flervalg

Middle Level

Oppgave	Tittel	Oppgavetype
13	M1	Flervalg

14	M2	Flervalg
15	M3	Flervalg
16	M4	Tekstfelt
17	M5	Tekstfelt
18	M6	Tekstfelt

Advanced Level

Oppgave	Tittel	Oppgavetype
19	A1	Langsvar
20	A2	Langsvar

1 E1

Which statements are true?

Select one or more alternatives:

- [x] The OS allows many programs to run at the same time.
- [x] The OS allows application software to access hardware resources without needing to know their technical details.
- [x] The OS takes physical CPU(s) and memory and abstracts them into a more general and easy-to-use virtual form.
- [ ] The OS is responsible for generating assembly code for application programs.

2 E2

Among the options below, which two abstractions help the users to find the storage location of their data?

Select one or more alternatives:

- [x] directory
- [ ] cache
- [ ] disk
- [ ] register
- [x] file
- [ ] memory

3 E3

Hardware drivers schedule all access to hardware to avoid conflicts if multiple programs try to access the same device or resource simultaneously.

Select one alternative:

- (•) False
- ( ) True

4 E4

Operating system utilities are designed to analyse, configure, optimize or maintain a computer. For example, the vmstat (Virtual Memory Statistics) command returns virtual memory status information, including process states and paging activity.

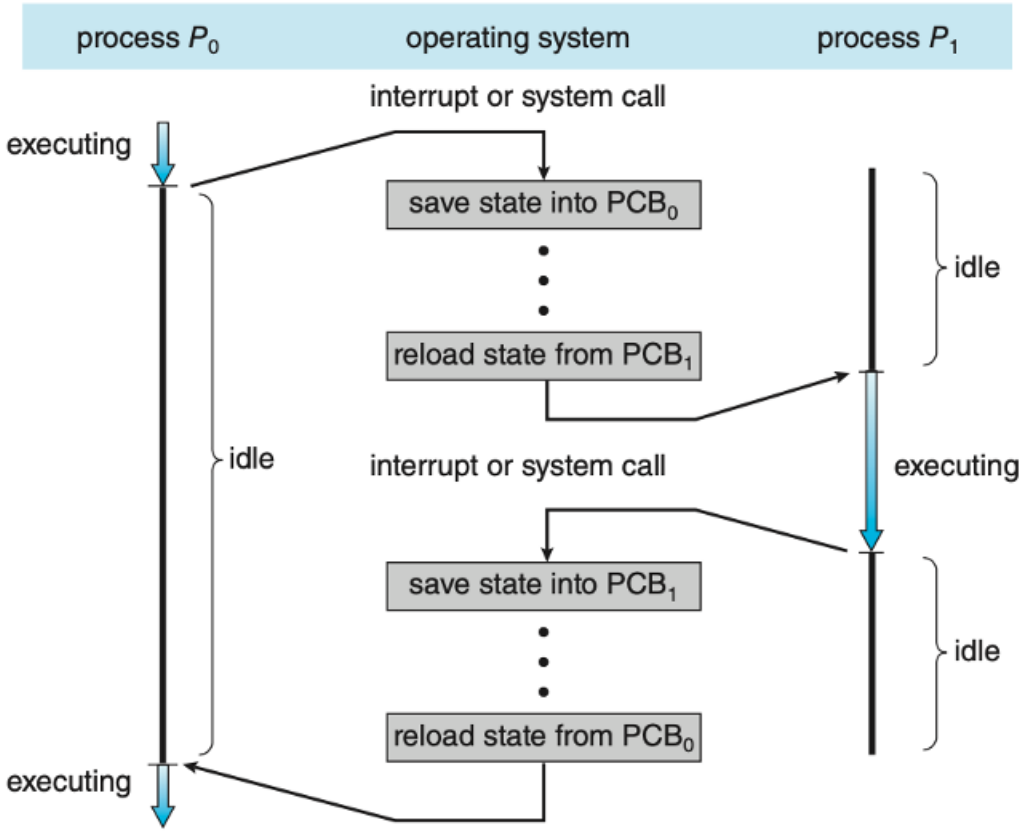
Select one alternative:

- (●) True
- ( ) False

5 E5

Which action of the OS does the picture below illustrates?

the os is swapping from one process to another, and needs to save the information about the p0 process and load information about the new p1 process. for example the program counter, and variables etc..



6 E6

The sequence of page frames in the physical memory matches the sequence of pages in the address space.

Select one alternative:

- True
- False

7 E7

What is the value of a binary semaphore when it is available?

1

8 E8

A non-terminating recursive function call will cause which of the following problem?

Select one alternative:

- stack overflow
- deadlock
- buffer overflow
- memory leak

9 E9

Compare Hard Disk Drive (HDD) with Solid State Drive (SSD). Which statements are correct?
Select one or more alternatives:

- [x] SSDs provide a huge performance advantage over HDDs— they’re faster to start up, faster to shut down, and faster to transfer data.
- [x] HDDs are much cheaper, per gigabyte, than SSDs.
- [x] HDDs contain mechanical moving parts which make them more noisy then SSDs.
- [ ] SSDs are more sensitive to physical vibration than HDDs.

10 E10

The command line "*gcc -o swap swap.c*" compiles the swap.*c* file and generates the executable file *swap*. Based on the given Makefile below, write down a simpler alternative command line which produces the same result as "*gcc -o swap swap.c*", and nothing else.

```
-------------------------------Makefile-----------------------------
all: process swap uppercase prime fib

clean:
    rm -f process swap uppercase prime fib

process: process.c
    gcc -o process process.c

swap: swap.c
    gcc -o swap swap.c

uppercase: uppercase.c
    gcc -o uppercase uppercase.c

prime: prime.c
    gcc -o prime prime.c

fib: fib.c
   gcc -o fib fib.c
--------------------------------------------------------------------------
```

make swap

11 E11

Which of the choices below are possible outputs from the execution of the following program?

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

int x = 10;

int main(){
   int rc = fork();
   if (rc == 0){
      for( ; ; ){
         sleep(1);
         printf("I am the child, x = %d\n", x);
         x++;
      }
   }else if(rc > 0){
      for( ; ; ){
         sleep(3);
         printf("I am the parent, x = %d\n", x);
         x = 12;
      }
   }
}
```

Select one or more alternatives:

- [x] I am the parent, x = 11
- [ ] I am the parent, x = 12
- [x] I am the child, x = 10
- [x] I am the parent, x = 10
- [x] I am the child, x = 12
- [ ] I am the child, x = 11

12 # E12

What does this shell script below do?

```
#! /bin/bash

for x in *
do
    if [ -f $x ]
    then
        cp  $x  $x.v2
    fi
done
```

Select one alternative:

- Rename all the files within the current directory
- Move all the files to a new directory
- Backup all the files within the current directory
- Copy all the directories to a new directory

13 **M1**

Assume that it takes one time unit for CPU to issue an I/O instruction, three time units for the I/O device to actually perform the task, and then one more time unit for CPU to complete the I/O instruction. So the total time from issuing an I/O instruction to finish the I/O instruction is 1+3+1=5 time units, in which 2 time units are spent on the CPU.

For running two processes, i.e., one process executes 1 I/O instruction and the other process executes 5 CPU instructions, compute the percentage of time that CPU is in use in the following two systems, respectively:

1. The system does NOT switch to another process while one is doing I/O, instead waiting until the process is completely finished.
2. The system switches to another process whenever one is WAITING for I/O.

Select one alternative:

- 60%, 90%
- 70%, 100%
- 65%, 95%
- 50%, 80%

14 **M2**

Compute the average response time when running five jobs of different lengths: 3, 10, 5, 2, 20 with the Round Robin (RR) scheduler and a time-slice of 5. Assume 5 jobs arrive in the system with this order at roughly the same time.

What is the average response time of using Round Robin?

Select one alternative:

- 5.2
- 6.3
- 7.8
- 4.9

15 **M3**

When the physical memory space is not enough, the OS will swap pages out of memory to the swap space and swap pages into memory from the swap space. Assume a physical memory can contain at most 3 pages and the memory is empty initially. Track the memory state changes when accessing pages 1,0,2,1,4,0,2,0,1,3 one-by-one in a sequence using the Least-Recently-Used (LRU) swapping policy. What is the hit rate?

Select one alternative:

- 50%
- 30%
- 20%
- 80%

16 **M4**

Valgrind is an instrumentation framework for building dynamic analysis tools that check C and C++ programs for memory errors. Below is an output example.

```
...
==3256==  HEAP SUMMARY:
==3256==   in use at exit: 44 bytes in 1 blocks
==3256==   total heap usage: 3 allocs, 2 frees, 2,092 bytes allocated
==3256==
==3256==  LEAK SUMMARY:
==3256==    definitely lost: 44 bytes in 1 blocks
==3256==    indirectly lost: 0 bytes in 0 blocks
==3256==    possibly lost: 0 bytes in 0 blocks
==3256==    still reachable: 0 bytes in 0 blocks
==3256==    suppressed: 0 bytes in 0 blocks
...
```

(1) What kind of memory error is reported by Valgrind? (2 points)

(2) How will you fix this memory problem in the C source code? Just mention an approach is good enough. (3 points)

(The length of the answer: approximately 2 sentences)

Fill in your answer here

1: this is a memory leak
2: memory was requested with malloc, but was never freed. so to solve this problem you need to use the free() function on the pointer that was allocated

17 M5

Duel-mode operation

(1) When does CPU switch from user mode to kernel mode, and vice versa. (2 points)

(2) Point out one benefit of using this dual-mode operation. (3 points)

(The length of the answer: approximately 3~5 sentences)

Fill in your answer here

1: it switches from user mode to kernal mode when the program makes an api call that needs kernal mode access. for example malloc. it also switches to kernal mode when there is an error. for example a page fault when a program tries to access a memory address outside its address space.

2: one benefit of dual mode is security, when a program wants to do something potentialy unsafe, they have to make an api call that the os handles for the program instead of the program directly doing the action. this is helpfull both if the program is malicius, but also if it is buggy.

18 M6

A program can *change* its behaviour over time. For example, a CPU-bound program may transition to an interactive program, or vice versa.

Explain how a multi-level feedback queue (MLFQ) scheduler adjusts the priority of a program which has been **CPU-bound** for a while but now becomes **an interactive program.** (5 points)

(The length of the answer: approximately 5 sentences)

Fill in your answer here

an MLFQ has multiple queues with different priorities, a process in a higher priority queue will be run first, and the ones in the lower queue will be run if there is enough time. processes in the same priority will share cpu time via round robin.
a process that is cpu-bound will drop down to a lower priority queue to allow other programs to also get access to cpu time. but all processes will periodically be bumpt up to a higher queue. and when the process switches from cpu-bound to be more interactive, it is less likely that it will be bumped down. so it will recive a higher priority

19 # A1

Below is an ×86 assembly code with AT&T syntax. Assume variable x is located at memory address 2500 and variable y is located at memory address 3000. The initial value of x is 10 and the initial value of y is 20.

Thread 1	Thread 2
mov 2500, %ax	mov 3000, %bx
mov 3000, %bx	mov 2500, %ax.
add %bx, %ax	sub %ax, %bx
mov %ax, 2500	mov %bx, 3000

(1) Write down the original source code in C for each thread seperately. (2 points each)

(2) What are the possible final values of x and y after both threads finish the execution? Write down the possible pairs of x and y - one for each case. For example, (x=2, y=4) and (x=2, y=7) are two different cases. (6 points for the correct pair of numbers, 4 points for the explanation of how you derive the result)

Fill in your answer here

1:

```
int x = 10;
int y = 20;

main(){
   fork(thread1())
   fork(thread2())
}

thread1() {
   x = x + y
}

thread2() {
   y = y - x
}
```

2:

(x=30, y=10) both threads moved the original variables into the registers before either thread updated any variables.

(x=30, y = -10) race condition where thread 1 updated x before thread 2 read x.

(x=20 , y=10) race condition where thread 2 updated y before thread 1 could read y

Ord: 80

20 # A2

Assume the physical memory size is 64KB, the address space size for the process we are looking at is 16KB, and the page size is 2KB. The format of the page table is described as the following: The high-order (left-most) bit of the page table entry is the VALID bit. If the bit is 1, the rest of the entry is the page frame number (PFN). If the bit is 0, the page is not valid. A table from hexadecimal to binary can be found in the attached file.

Page Table (from entry 0 down to the max size)
[0] 0x8018
[1] 0x0000
[2] 0x0000
[3] 0x800c
[4] 0x8009
[5] 0x0000
[6] 0x801d
[7] 0x8013

Given the page table above, answer each of the following questions to decide which of these 3 virtual addresses are valid and which are not?

Virtual Addresses :
VA 0x1008 --- PA or invalid ?
VA 0x33da --- PA or invalid ?
VA 0x3a39 --- PA or invalid ?

(1) How many pages can the address space contain? (1 point)
(2) How many bits are required in a virtual address (VA) in order to address all of the address space? (1 point)
(3) How many top-bits of a virtual address (VA) in this case are used to indicate the page number of the page that the VA locates? (1 point)
(4) How many bits in a virtual address (VA) indicate the offset of VA in its page? (1 point)
(5) How many bits are required in a physical address (PA) in order to address all of the physical memory? (1 point)
(6) For each virtual address above, answer the following questions:
(6-1) Calculate the Page Number. (1 point each)
(6-2) Is the virtual address valid or not?
(6-2-a) If the virtual address is invalid, explain why it is invalid. (2 points each)
(6-2-b) If the virtual address is valid, what is the corresponding page frame number? (1 point each)
(6-3) Write down the corresponding physical address (PA) in the hexadecimal format.
Hint: PA is formed by the concatenation of the page frame number and the offset in the binary format. (2 points each)

Fill in your answer here

1:
2: 14 bits
3: we need 2 bits from the virtual address to find the page number. the first letter/number in the virtual address is converted to binary, and the two first bits (from the left) are discarded.

the next two bits are used to look up the page number.

4: 12 bits

5: 14 bits

6:

-VA 0x1008 the first number 1 translates to 0001 we remove the first 2 bits, and are left with 01 giving us the 2 page in the table (page [1]). the first number in the page is 0 which translates to 0000 in binary. since the first bit is not 1 the address is invalid

the physical address in binary is 0000 0000 0000 1000, which in hex becomes 0x0008

-VA 0x33da the first number 3 translates to 0011 which means the page number is [3]

the first number in that page is 8 which translates to 1000, since the first bit is 1, the page is valid.

the last number is c which in binary is 1100. we remove the first two bits giving us 00 which is the page frame number.

the physical address becomes 0000 0011 1101 1010 which in hex is 0x03da

-VA 0x3a39 the first number 3 translates to 0011 which means the page number is [3]

the first number in that page is 8 which translates to 1000, since the first bit is 1, the page is valid.

the last number is c which in binary is 1100. we remove the first two bits giving us 00 which is the page frame number.

the physical address becomes 0000 1010 0011 1001 which in hex is 0x0a39

Ord: 280