Operating Systems ID2206
English version
2017-06-07 8:00-12:00

Instruction

- You are, besides writing material, only allowed to bring one self hand written A4 of notes. Mobiles etc, should be left to the guards.
- All answers should be written in these pages, use the space allocated after each question to write down your answer.
- Answers should be written in Swedish or English.
- You should hand in the whole exam.
- No additional pages should be handed in.

Grades for 6 credits

The exam is divided into a number of questions where some are a bit harder than others. The harder questions are marked with a star *points**, and will give you points for the higher grades. The exam is thus divided into basic points and points for higher grades. First of all make sure that you pass the basic points before engaging with the higher points.

Note that, of the 40 basic points only at most 36 are counted, the points for higher grades will not make up for lack of basic points. The limits for the grades are as follows:

- Fx: 21 basic points
- E: 23 basic points
- D: 28 basic points
- C: 32 basic points
- B: 36 basic points and 12 higher points
- A: 36 basic points and 18 higher points

The limits could be adjusted to lower values but not raised.

Name:_____ Persnr:_____

1 Operating systems

1.1 what happens? [2 points]

If we give the following commands after each other in a *shell*; what will the result be?

```
> mkdir foo
> echo "gurka" > foo/gronsak.txt
> ln -s gronsak.txt foo/check.txt
> mkdir bar
> echo "morot" > bar/gronsak.txt
> mv foo/check.txt bar
> cd bar
> cat check.txt
```

1.2 commands in a shell [2 points]

Give a short description of the commands below.

- `chmod`
- `diff`
- `sed`
- `tail`

Name:_____ Persnr:_____

2 Processes

2.1 what is where? [2 points]

In the code below we have the variables: `x`, `i`, `h` och `c`. In which segments do we find the data stuctuctures they are assigned to: global, stack or heap? What will be printed on stdout?

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

int x = 3;

int foo(int i) {
  int h[] = {1,2,3,4};
  return h[i];
}

int main() {

  int c = foo(x);
  printf("h[%d] = %d \n", x, c);
  return 0;
}
```

2.2 The IDT [2 points*]

In a x86-architecture we have an IDT that is used when we implement among other things system calls. What does the operating system have to add to the IDT in order to make a system call possible?

Name:_____ Persnr:_____

3 Scheduling

3.1 multi-level feedback queue [2 points]

When we implement a scheduler using a *multi-level feedback queue* we can use a strategy to give higher priority to interactive processes. What does this strategy look like?

3.2 lottery [2 points]

In so called lottery based scheduling the focus is not to reduce reaction time nor turnaround time. What is it that we try to acheive and how can implementing a lottery help us?

3.3 multi processor [2 points*]

When implementing a scheduler for a multi processor there is one aditional property to take into account. What is it that must be considereed and which strategy schould be used to handle the problem.

Name:_____ Persnr:_____

4 Virtual memory

4.1 paging [2 points]

Assume we have a virtual address that consist of a 20-bit page number and a 12-bit offset. We have a physical address space of 36 bits which makes our translation from virtual to physical addresses interesting. What will our page table contain and how much memory can a single process address?

4.2 TLB [2 points]

A TLB is important for address translation to be as quick as possible. What is a TLB and how does it improve address translation?

4.3 x86_64 [2 points*]

In x86_64 architecture we have a virtual address space of 48 bits. These are divided in a page number encoded as four segments of 9 bits each and an offset of 12 bits. Why is the page number divided into four parts of 9 bits each? Why not three or two segments, or why simply not use one value of 36 bits?

Name:_____ Persnr:_____

5 Memory management

5.1 intern and extern fragmentation [2 points]

Give an explanation of what is meant by internal and external fragmentation.

5.2 best-fit vs worst-fit [2 points]

What could the advantage be that instead of choosing the resource that best fit a request, choose the one that is as large as possible (worst-fit).

Name:_____ Persnr:_____

5.3 segregated lists [2 points*]

A strategy to implement handling of so called *free lists* in memory management is to let all blocks be of a size equal to a power of 2 (with some smallest value, for example 32 bytes). If a block of a particular size is not available the next largest block is chosen and divided in two. When we free a block we might want to check if adjacent block is free in order to coalesce them into one block and prevent an accumulation of small blocks. How can we easily determine what adjacent block to check? Does the strategy have any limitations?

Name:_____ Persnr:_____

6 Concurrent programming

6.1 count [2 points]

The C code below, with corresponding assembly, hihlights something that we have to take into account when working with shared memory fro threads. What problem is it that the code is an example of?

```
int loop = 10;
int count = 0;

void *hello(void *) {
      :
  for(int i = 0; i < loop; i++) {
      count++;
  }
      :
}
```

```
.L3:
        movl    count(%rip), %eax
        addl    $1, %eax
        movl    %eax, count(%rip)
        addl    $1, -4(%rbp)
        movl    loop(%rip), %eax
        cmpl    %eax, -4(%rbp)
        jl      .L3
```

6.2 Anna Book [2 points]

There is a simple strategy that sometimes can be used to to guarantee that deadlock will not occur. We can avoid the situation all together, not break a deadlock that has occured. Describe this strategy and explain why it avoids deadlocks.

Name:_____ Persnr:_____

6.3 TSFO [2 points*]

You and your colleague Alan Peterson have realized that you need to implement a lock to protect shared data structures. Alan proposes that each thread should have a flag that signals that the thread wants to enter the critical section. If a tread first sets its own flag and then verifies that no other thread has set its flag the problem is solved. If another thread also has the flag set the flag is reset and tries again sometime later. There is a risk for starvation but this is problem that is acceptable.

What will you tell your colleague?

Name:_____ Persnr:_____

7 File systems and storage

7.1 links [2 points]

If one should implement a traversal of a general graph, one needs to be careful not to end up in a circular structure. If you were asked to search a directory and all sub-directories in a Unix system, where there could be both hard and soft links, wich strategy would you use to avoid ending up in a loop?

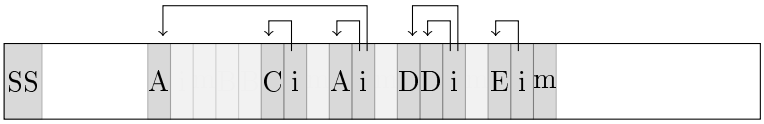
7.2 inodes and data blocks [2 points]

A file system represents fiels and directories by inodes and datablocks. Assume that we do not have anything in any file-cache but have to read anything from disk. Which inodes and datablocks do we have to read to begin reading from the file `/home/alan/gurka.txt`? List the inodes and blocks in the order we will read them.

Name:_____ Persnr:_____

7.3 log based file system [2 points*]

Below you see a schematic image of a log based file system. If the system now has a shortage of space it will try to create some. How can more space be created and what will the system look like when this has been done?



Name:_____ Persnr:_____

8 Virtualization

8.1 kernel mode [2 points]

Assume we have processor that only provides two levels of execution: *user mode* and *kernel mode*. When virtualizing a complete operating system the virtualized operating system needs to run in *kernel mode* - true or false? Motivate your answer.

8.2 containers [2 points*]

So called "Linux containers" is an alternative way of running several operating systems on the same machine. Which limitations does this method compared to full virtualisation?

Name:_____ Persnr:_____

9 Implementation

9.1 danger ahead [2 points]

It is not completely defined what will happen if we run the code below. What is it that we do wrong and what could a possible effect be?

```
int main() {

  char *heap = malloc(20);
  *heap = 0x61;
  printf("heap pointing to: 0x%x\n", *heap);
  free(heap);

  char *foo = malloc(20);
  *foo = 0x62;
  printf("foo pointing to: 0x%x\n", *foo);

  *heap = 0x63;
  printf("foo pointing to: 0x%x\n", *foo);

  return 0;
}
```

Name:_____ Persnr:_____

9.2 dup() [2 points]

Assume that we have a program `boba` trhat writes "Don't get in my way" to `stdout`. What will the result be if we run the program below and why is this the result? (the procedure `dprintf()` takes a file descriptor as argument)

```
int main() {

  int fd = open("quotes.txt", O_RDWR | O_CREAT, S_IRUSR | S_IWUSR);

  int pid = fork();

  if(pid == 0) {
    dup2(fd, 1);
    close(fd);
    execl("boba", "boba", NULL);
  } else {
    dprintf(fd, "Arghhh!");
    close(fd);
  }
  return 0;
}
```

Name:_____ Persnr:_____

9.3 ctrl-c [2 points]

A simple way to kill a program is to hit ctrl-c. If we write a program we might not want to die or we might want to do some last operations before terminating. What mechanisms should we use in our program to handle this?

9.4 random not that bad [2 points]

If we implement a procedure that should evict a page from memory when it is filled we can select the page by random. If we have a memory that consists of r frames and we have a total of n pages that should share the memory one might assume that the the random strategy would give us a hit rate of n/r. In reality the result is often much better, why is this?

Name:_____ Persnr:_____

9.5 list the content of a directory [2 points]

If we want to list the content of a directory we can use the library procedure `opendir()`. Which information can we access directly form the structure pointed to by `entry` in the code below? Whichj information can we not fins and where could this information be found?

```
int main(int argc, char *argv[]) {

  char *path = argv[1];

  DIR *dirp = opendir(path);

  struct dirent *entry;

  while((entry = readdir(dirp)) != NULL) {

    //  what information do we have?

  }
}
```

9.6 read performace [2 points]

Assume that we have an ordinary hard drive that is attached over a SATA connection of 6 Gb/s that gives us a read time of a random 4 KByte block of 12 ms. How is this changed if we replace the disk to one that is attached by a SAS connection of 12 Gb/s? Motivate.

Name:_____ Persnr:_____

9.7 a header and a footer [2 points*]

When implementing a memory allocator it is common to hide a *header* just before the memory area that is allocated. In this head you can for example record the size of the are to make it easier to handle the area when *freed*. One can also use a hidden footer after the area where one can record that the area is used or free and maybe a pointer to its head. What is the advantage of having this information after the area, is it not enough with the header?

9.8 lite bättre [2 points*]

A so called *pipe* is a simple way to send data from one process to another. It does have its limitations and a better way is to use so called *sockets*. If we instead of a pipe use a stream socket between two processes we will have several advantages. Describe two advantages that a steam socket gives us that we will not have if we use a pipe.

Name:_____ Persnr:_____

9.9 name space [2 points*]

Below is code were we open a socket and use the name space `AF_INIT`. We will then be able to address a server using a port number and IP-address. There are other name spaces that we can use when working with sockets. Name one and decribe its advantages and disadvantages it might have.

```
struct sockaddr_in server;
server.sin_family = AF_INET;
server.sin_port = htons(SERVER_PORT);
server.sin_addr.s_addr = inet_addr(SERVER_IP);
```

9.10 AC/DC [2 points*]

If we have allocated a two dimensional array `table` and then want to sum all its values we can do this using the code below. What in the code below will give us an unwanted behaviour and how could we improve the execution time? Motivate.

```
#define ROWS 4000
#define COLS 1000

int table[ROWS][COLS];

int main() {

  :
  long sum = 0;

  for(int c = 0; c < COLS; c++) {
     for(int r = 0; r < ROWS; r++) {
       sum += table[r][c];
     }
  }
  :

}
```