Operating Systems ID2206
English version (only for ID2206 HT16)
2017-08-21 8:00-12:00

Instruction

• You are, besides writing material, only allowed to bring one self-handwritten A4 of notes.
• 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.
1 Operating systems

1.1 what happens? [2 points]

If we give the command sequence below, what will the result be?

```
> echo "cd foo grep bar" | grep bar | wc -w
```

Answer: 4

1.2 commands in a shell [2 points]

Give a short description of the commands below.

- `mkdir`
- `cd`
- `cat`
- `ln`

Answer: Have a look using `man`. 
2 Processes

2.1 why on the heap? [2 points]

In the code below we have allocated three arrays where one is on the heap, which array and why is it allocated on the heap and not on the stack?

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

#define MAX 4

int h[MAX];

int *foo(int *a, int *b, int s) {
    int *r = malloc(s * sizeof(int));
    for(int i = 0; i < s; i++) {
        r[i] = a[i]+b[i];
    }
    return r;
}

int main() {
    int f[MAX];
    for(int i = 0; i < MAX; i++) {
        f[i] = i;
        h[i] = i*10;
    }
    int *g = foo(f, h, 4);
    printf("a[2] + b[2] is %d\n", g[2]);
    return 0;
}
```

Answer: 3
2.2 Limited direct execution [2 points*]

In the implementation of an operating system one will use so called limited direct execution, what are the limitations?

Answer:

3 Scheduling

3.1 shortest time-to-completion first [2 points]

Assume that we have a scheduler that implements short time-to-completion first. We have three jobs that arrive at time 0, 10 and 30 ms and take 60, 30 and 10 ms. What is the average turnaround time?

Answer:
3.2 state diagram [2 points]

Here follows a state diagram for scheduling of processes. Enter the marked states and transitions to describe what states mean and when a process is transferred between different states.

Answer:

![State Diagram]

3.3 lottery [2 points*]

There are schedulers that are based on a lottery where you allocate a number of tickets to each process, pick a number and let the winning process execute for a certain amount of time. In these schedulers it not the turnaround time nor the reaction time that you want to minimize, what is it that we want to achieve?

Answer:
4 Virtual memory

4.1 segmenting [2 points]

When we use segmentation to handle physical memory we could have problems with external fragmentation. This is avoided if we instead use paging. How is it that we can avoid external fragmentation using paging?

Answer:

4.2 the memory [2 points]

In Linux, and many other operating systems, a process has a virtual address space. In this space it will hold its own areas: stack, code, global data and heap. The address space also includes the operating system. Draw a schematic image over how these areas are typically arranged. Also include for the stack and heap in which direction they grow.

Answer:

```
\begin{array}{cccccc}
\text{code} & \text{data} & \text{heap} & \rightarrow & \leftarrow & \text{stack} & \text{OS} \\
\end{array}
```
4.3 more memory [2 points*]

If a virtual address is 32 bits we can atmost address a memory of size 4 GiByte. Assume memory is cheap and we can afford a machine that has 64 GiByte of memory. How can we utilize such a machine without changing the size of the virtual address?

Answer:

5 Memory management

5.1 malloc() [2 points]

In Linux (and all Unix dialects) malloc is a library procedure and not a system call. Why is it a library procedure? Would it not be quicker if we called the operating system directly, in the end it will manage the memory any way.

Answer:

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

One strategy to find a suitable memory block is to find the block that best suites our needs (without being too small); this must by all aspects be the a good strategy. Another approach is to simply take the first block that is found even if it is considerably larger than what we need. What would the benefit be for the latter strategy and what is the possible downside?

Answer:

5.3 intern paging [2 points*]

When we implement memory internally for a process (for example in malloc()) we us a form of segmentation. This is why we could have problem with external fragmentation. If it’s better to use paging why do we not use it when we implement internal memory management?

Answer:
6 Concurrent programming

6.1 thing in the heap [2 points]

If we have a multithreaded program the threads can of course read and write global variables and thus work with shared data structures. How is it with data structures that one thread allocates on the heap, can these structures be read/written to by other threads?

Answer:

6.2 deadlock, almost [2 points]

What is the difference between so called deadlock and livelock?

Answer:
6.3 the world is not simple [2 points*]

In the program below we have two procedures that both will update a global variable `count` by 1000 steps. In order to run these procedures in two threads the update is protected by two flags, `a` and `b`. Each process will first set its own flag and then continue only if the other process has not set its flag. The world is not always simple and if we have a processor that only provides “total store order” strange things can happen - explain what could happen.

```c
void *ping(void *arg) {
    int i;
    for (i = 0; i < 1000; i++) {
        while (1){
            a = 1;
            if (b != 1) {
                count ++;
                a = 0;
                break;
            } else {
                a = 0;
            }
        }
    }
}

void *pong(void *arg) {
    int i;
    for (i = 0; i < 1000; i++){
        while (1){
            b = 1;
            if (a != 1) {
                count ++;
                b = 0;
                break;
            } else {
                b = 0;
            }
        }
    }
}
```

Answer:
7  File systems and storage

7.1  remove a file [2 points]

If we us the command `rm` we will nit remove a file rather remove a hard ink to a file. How do we remove the actual file?

Answer:

7.2  two out of three [2 points]

Assume that we have a simple file system without journaling where we write directly to inodes, bit maps and data blocks. Also asuem that we have a crash and the we when creating a file only are able to do two of the three required updates. In each of the cases below, explain the problem we will have.

- inode and bitmaps

  Answer:

- bitmaps and data blocks

  Answer:

- inode and data blocks

  Answer:

7.3  log-based fs [2 points*]

In a log based file system we write all changes to a continuous log without doing any changes to existing blocks of a file. What is the advantage of writing new modified copies of blocks rather than do the small changes we want to do in the original blocks? If it is better, are there any disadvantages?

Answer:
8 Virtualization

8.1 how slow [2 points]

When we run a full operating system in a virtualized environment the execution will take longer time since we then have a virtualization in two levels. How much slower will a computation intensive program run: almost as fast, half speed or typically a factor ten slower? Motivate your answer.

Answer:

8.2 set the IDT [2 points*]

When a hypervisor starts a virtualized operating system, the virtualized system will want to set the register that controls the location of the IDT. What is the problem and how is it solved?

Answer:
9 Implementation

9.1 memory map [2 points]

Below is a, somewhat shortened, printout of a memory mapping of a running process. Briefly describe the role of each segment marked with ???.

> cat /proc/13896/maps

00400000-00401000 r-xp 00000000 08:01 1723260 .../gurka ???
00600000-00601000 r--p 00000000 08:01 1723260 .../gurka ???
00601000-00602000 rw-p 00001000 08:01 1723260 .../gurka ???
022fa000-0231b000 rw-p 00000000 00:00 0 [???
7f6683423000-7f66835e2000 r-xp 00000000 08:01 3149003 .../libc-2.23.so ???
: 7ffd60600000-7ffd60621000 rw-p 00000000 00:00 0 [???
7ffd60648000-7ffd6064a000 r--p 00000000 00:00 0 [vvar]
7ffd6064a000-7ffd6064c000 r-xp 00000000 00:00 0 [vdso]
fffffffffffffff600000-fffffffffffffff601000 r-xp 00000000 00:00 0 [vsyscall]

Answer: The first three segments are: code, read-only data and global data for the running process gurka. Then there is a segment for the heap. The segment marked with libc-2.23.so is a shard library. In the uppermost region we find the segment of the stack.
9.2  fork [2 points]

If we run the program below, what will be printed on the terminal? Make sure that you get the order right.

```c
int x = 0;

int main() {
    int pid;

    pid = fork();

    if (pid == 0) {
        x = x + 10;
        sleep(1);
    } else {
        sleep(1);
        x = x + 2;
        wait(NULL);
    }
    printf("x is %d\n", x);

    return 0;
}
```

Answer: 13
9.3 Boba [2 points]

Assume that we have a program `boba` that 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)

```c
int main() {
    int pid = fork();

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

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

Answer:
9.4 pipes [2 points]

In the code below we have a program that creates a pipe and then use this pipe to send messages (not shown). What would the equivalent skeleton code look like for program that opens the same pipe for reading the messages?

```c
int main() {
    int mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
    mkfifo("sesame", mode);

    int flag = O_WRONLY;
    int fd = open("sesame", flag);
    
}
```

**Answer:**

```c
int main() {
    int flag = O_RDONLY;
    int fd = open("sesame", flag);
    
}
```
9.5 a directory [2 points]

A directory is in Linux represented in the same way as a file i.e. an inode that is pointing to a data block and the data block holds the name and identifiers of the files in the directory. This mean that we use the same system calls when we read a directory as we use when we read a file - true or false? Motivate.

Answer:

9.6 sbrk() and then what [2 points*]

You can use the system call sbrk() to allocate more memory for the heap but how can a process return memory?

Answer:
9.7 an expensive operation [2 points]

Below is a extract from a program that implements Least Recently Used (LRU). The code shows why LRU is expensive to implement and why one probably instead choose to approximate this strategy. What is the code doing and when is it executed?

```c
if (entry->present == 1) {
    if (entry->next != NULL) {
        if (first == entry) {
            first = entry->next;
        } else {
            entry->prev->next = entry->next;
        }
        entry->next->prev = entry->prev;
        entry->prev = last;
        entry->next = NULL;
        last->next = entry;
        last = entry;
    } else {
        :
    }
}
```

**Answer:** The code unlinks an entry and places it last in a list that should be updated with the least used pages first. This operation must be done every time a page is referenced.
9.8 volatile [2 points*]

In the code below you see that count and turn are declared as volatile. Why is it important to declare some variables as volatile when we work with multithreaded code?

```c
int loop = 100000;
volatile int count = 0;
volatile int turn = 0;

void *toggle(void *args) {
    int id = ((struct ids*)args)->id;
    int od = ((struct ids*)args)->od;

    for(int i = 0; i < loop; i++) {
        while(turn != id) {}
        count++;
        turn = od;
    }
}
```

**Answer:**
9.9 character device [2 points*]

We can create so called *character device* and interact with it using *ioctl*. In the code below, describe what `fd`, `JOSHUA_GET_QUOTE` and `buffer` is and how the *device* could work.

```c
if (ioctl(fd, JOSHUA_GET_QUOTE, &buffer) == -1) {
    perror("Hmm, not so good");
} else {
    printf("Quote - %s\n", buffer);
}
```

Answer:
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.

```c
#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];
        }
    }
}
```

Answer:
Answer:

Answer:

Answer:

**Answer:** Problemet är att anrop till tjänsten kan dubbleras och utföras mer än en gång. Man löser det genom att enbart ha s.k. idempotenta operationer dvs operationer som kan processas flera gånger utan att tillståndet förändras mer än vid första anropet.