Operating Systems ID1206 and ID2206

Exam TENA 6 hp

2018-04-03 14:00-18:00

Instruction

- You are, besides writing material, only allowed to bring one self hand written 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

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 *[p*]*, 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 24 basic points only at most 22 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: 12 basic points
- E: 13 basic points
- D: 16 basic points
- C: 20 basic points
- B: 22 basic points and 6 higher points
- A: 22 basic points and 10 higher points

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

Name:____________________________ Persnr:____________________________

1 Processer

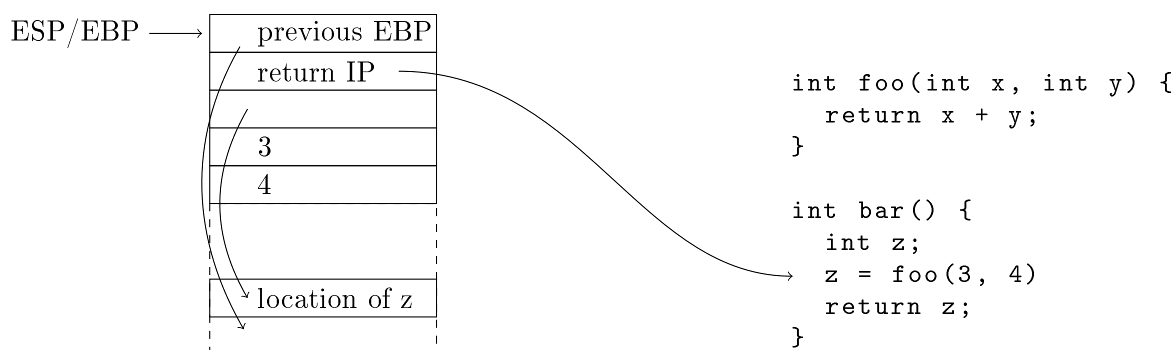
1.1 fork() [2p]

When you create a process using `fork()` the two processes will **share** some structures. Which, if any, of the following will the two processes share.

- Stack **Answer:** no
- Heap **Answer:** no
- Global memmory **Answer:** no
- Code area **Answer:** no
- Open files **Answer:** yes

1.2 what is on the stack [2p]

Assume we do a procedure call to `foo()` as in the example below and that all information is placed on the stack, i.e. we will not use the registers for arguments and result, what is then on the stack when we enter `foo`? Draw a stack and describe what it contains. Also show the stack pointer and base pointer.



Answer:

Name:___________________________ Persnr:___________________________

1.3 spot the error [2p]

In the program below we have a procedure `print_fibs/0` that with the help of `some_fibs/1` will print a sequence of Fibonacci numbers from 0 to 46. Everything works fine, or .... what have we forgotten, what consequences could it have?

Answer: We allocate an array on the heap but then fail to `free()` this space. The result is a memory leak - each time we call `print_fibs()` we will increase the size of the heap. In the end we will run out of space.

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

#define MAX 46

int *some_fibs(int n) {

  if (n <= MAX) {
    int *buffer =  malloc((n + 1) * sizeof(int));
    buffer[0] = 0;
    buffer[1] = 1;
    for(int i = 2; i <= n; i++) {
      buffer[i] = buffer[i-1] + buffer[i-2];
    }
    return buffer;
  } else {
    return NULL;
  }
}

void print_fibs() {

  int *fibs;
  int n = 46;

  fibs = some_fibs(n);
  for(int i = 0; i <= n; i++) {
    printf("fib(%d) = %d\n", i, fibs[i]);
  }
}
```

1.4 why malloc [2p]

When you prgram in C you must explicitly allocate some data structures on the heap by using malloc. In languages that have automatic memory manegement, as for example Java, Erlang and Python, you do not have to think about where structures are allokated. How do these languages work? Is everything allocated on the stack, is there no heap? If there is a heap must one not remove structures that are not used? Explain how these langauges handle the memory.

Answer:

1.5 same-same [2p*]

When you implement malloc/free one alternative is to only hand out even exponents of two. You would have a lower limit and possibly hand out blocks of size: 16 bytes, 32 bytes, 64 bytes, 128 bytes etc. This has some advantages but also some drawbacks. Describe how the system could be implemented and its pros and cons.

Answer:

1.6 yield() or futex_wait() [2p*]

If we implement a spin-lock we can call either `sched_yield()` or `futex_-wait()` as in the example below, if the lock is not taken at the first try. Why do we want to do that and what is the difference between the two solutions.

```
int try(int *lock) {
  __sync_val_compare_and_swap(lock, 0, 1);
}

int futex_wait(volatile int *futexp) {
  return syscall(SYS_futex, futexp, FUTEX_WAIT, 1, NULL, NULL, 0);
}

void lock(volatile int *lock) {
  while(try(lock) != 0) {
    sched_yield();
  }
}

void lock(volatile int *lock) {
  while(try(lock) != 0) {
    futex_wait(lock, 1);
  }
}
```

Answer:

Name:____________________________ Persnr:____________________________

2 Communication

2.1 count [2p]

If we execute the prohgram below we will print the final value of the variable `count`. Which possible proint out can we receive and why?

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>

int count = 0;

int main() {

  int *status;

  int pid = fork();

  if( pid == 0) {
    for (int i = 0; i < 10; i++) {
      count += 1;
    }
    return 0;
  } else {
    for (int i = 0; i < 10; i++) {
      count += 1;
    }
    wait(status);
  }
  printf("count = %d\n", count);
  return 0;
}
```

Answer:

2.2 pipes [2p]

Which, if any, statements are true?

- A pipe is duplex channel for communication between processes within the same operating system. **Answer:** false

- When you read from a pipe you will read one message at a time. You will allways read full message, never half a message. **Answer:** false

- When you read from or write to a pipe you will use the regular library procedures that are used for files. **Answer:** true

2.3 types of sockets [2p*]

When we create a socket we declare wich type of socket that we want; below are three types listed. For each of the types describe the properties of the created socket. Note properties as: byte or message oriented, one way or duplex, reliability, order and flow control.

- `SOCK_DGRAM`

  Answer:

- `SOCK_STREAM`

  Answer:

- `SOCK_SEQPACKET`

  Answer:

Answer:

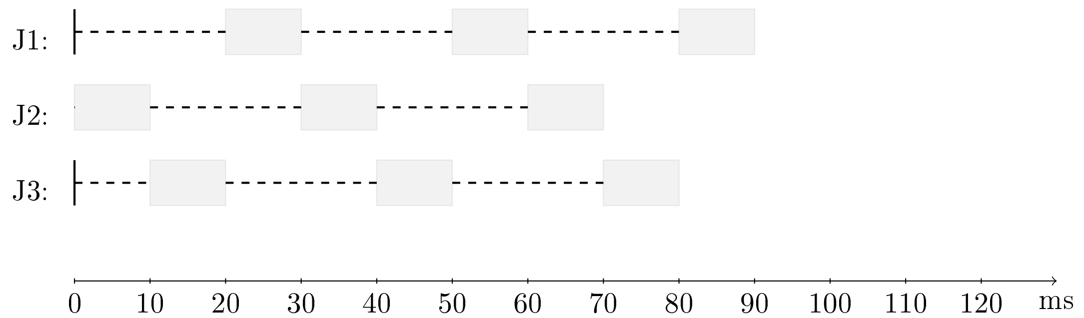
Name:____________________________ Persnr:____________________________

3 Scheduling

3.1 worse turnaround time [2p]

Show in a graph how we can get a worse turnaround time if we instead of running three jobs of 30 ms each afer each other, implement round-robin where each job obtains a 10 ms time slot.

Answer:



3.2 MLFQ - WTF! [2p]

Assume that we implement a MLFQ as a job scheduler with the rules below. What will the problem be and how can we solve it?

- Rule 1: if Priority(A) > Priority(B) then A is scheduled for execution.
- Rule 2: if Priority(A) = Priority(B) then A and B are scheduled in round-robin.
- Rule 3: when a new job is created it starts with the highest priority.
- Rule 4a: a job that has to be preempted (time-slice consumed) is moved to a lower priority.
- Rule 4b: a job that initiates a I/O-operation (or yields) remains on the same level.

Answer:

3.3 a zombie [2p*]

A process can be in a so called zombie state. What does it mean, why is aprocess in this state, when is it removed from the state and then what happens with the process?

Answer:

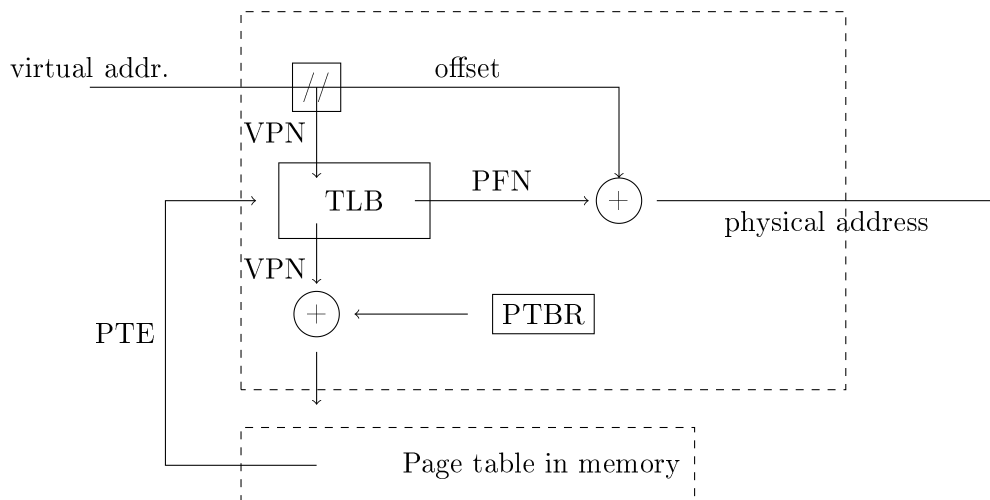
Name:______________________________ Persnr:______________________________

4 Virtual memory

4.1 A paging MMU with TLB [2p]

Below is a picture of a MMU that uses a TLB to translate virtual addresses to physical addresses. Identify the following units and addresses: virtual address, physical address, page table base register (PTBR), offset in page, page number (VPN), frmae number (PFN), TLB, page table, page table entry (PTE).

Answer:



Name:__________________________ Persnr:__________________________

4.2 the hiden code [2p]

When implementing malloc/free one can do a small trick and “hide” information about a block that is handed out. Which information is hiden and how is it hiden.? Explain why this is necessary, how it is done and where in the code below that it is done.

```
void *malloc(size_t size) {
  if( size == 0 ){
    return NULL;
  }
  struct chunk *next = flist;
  struct chunk *prev = NULL;

  while(next != NULL) {
    if (next->size >= size) {
      if(prev != NULL) {
        prev->next = next->next;
      } else {
        flist = next->next;
      }
      return  (void*)(next + 1);
    } else {
      prev = next;
      next = next->next;
    }
  }

  void *memory = sbrk(size + sizeof(struct chunk));
  if(memory == (void *)-1) {
    return NULL;
  } else {
    struct chunk *cnk = (struct chunk*)memory;
    cnk->size = size;
    return  (void*)(cnk + 1);
  }
}
```

Answer:

4.3 inverted page table [2p*]

An inverted page table is a page table that given porcess identifier and page number returns the frame number where the page is found (if it is in memory). The table has one entry per frame in memory and can not be indexed directly by the page number, one has to search for the right entry.

What is the advantage of using an inverted page table if the lookup operation take longer time? Explain the advantages and when it is most advantageus to use an inverted page table.

Answer:

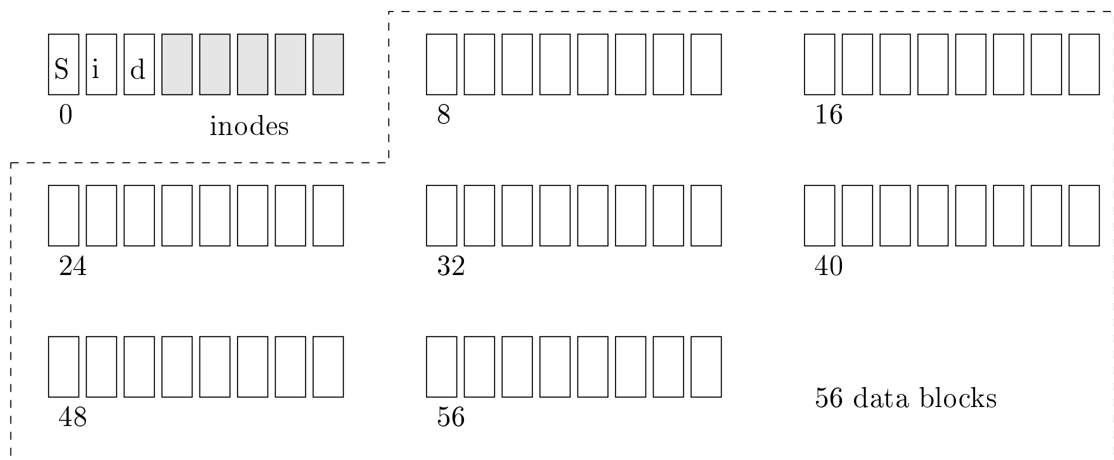
Name:______________________________ Persnr:______________________________

5 File systems and storage

5.1 a simple file system [2p]

Describe how a simple file system could be organized on a hard drive divided into segments of 4 Kbyte. Describe which datstructures are needed and draw how these are arranged on the hard drive (we have a minimal disk as example).

Answer:



Name:____________________________ Persnr:____________________________

5.2 content of a directory [2p]

You have implemented you own program to list the content of a directory. You have tested your program on different directories and thing seams to work. All of a sudden you have a segmentation error, what has happened?

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

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

  if( argc < 2 ) {
    perror("usage: myls <dir>\n");
    return -1;
  }

  char *path = argv[1];

  DIR *dirp = opendir(path);

  struct dirent *entry;

  while((entry = readdir(dirp)) != NULL) {
    printf("\tinode: %8lu",  entry->d_ino);
    printf("\tname: %s\n", entry->d_name);
  }

  return 0;
}
```

Answer:

Name:______________________________ Persnr:______________________________

5.3 log-based fs [2p*]

In a log-based file system the inodes do not have any dedicated location but are written in segments together with the data blocks. The problem then is to find the inodes: how do we keep track of the locations of the inodes? Describe and draw in teh figure below how we find the inodes.

Answer: