Operating Systems ID1206
2020 01 07

Instruction

- You are, besides writing material, only allowed to bring one self hand written A4 of notes. The notes are handed in and can not be reused.
- 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 and the hand written page of notes. No additional pages should be handed in.

Grades

The exam is divided into two parts, one basic part consisting of five questions and one part for higher grades also consisting of five questions.

To pass, grade E, the basic part should be completed with the result below. For a higher grade, two or more of the five advanced questions should be answered correctly.

- Fx: 7 points of 10 in the basic part
- E: 8 points of 10 in the basic part

Higher grade is given based on the result off the second part.

- D: two passed
- C: three passed
- B: four passed
- A: all passed

Name:____________________________ Persnr:____________________________

Answer: The provided answers are not necessarily answers that would result in full points, longer explanations could be required.

1 swap a context [2 points]

An interesting experiment is to save the current context and then later install it. This is done using the system calls `getcontext()` and `swapcontext()`.

`The function getcontext(ucontext_t *ucp) initializes the structure pointed at by ucp to the currently active context.`

`The swapcontext(ucontext_t *oucp, const ucontext_t *ucp) function saves the current context in the structure pointed to by oucp, and then activates the context pointed to by ucp.`

What is the result of executing the program below?

```
volatile int done = 0;

int main() {

  ucontext_t one;
  ucontext_t two;

  getcontext(&one);

  printf("done = %d\n", done);

  if(!done) {
    done = 1;
    printf(" - gurka -\n");
    swapcontext(&two, &one);
    printf(" - tomat -\n");
  } else {
    printf(" - salad -\n");
    swapcontext(&one, &two);
    printf(" - morot -\n");
  }

  printf(" - potät -\n");

  return 0;
}
```

Name:________________________ Persnr:________________________

Answer:

```
done = 0
 - gurka -
done = 1
 - salad -
 - tomat -
 - potät -
```

Name:____________________________ Persnr:____________________________

2 fork() [2 points]

Below you find some skeleton code that does a `fork()`. The program continues as two processes that will do different things. Complete the code so that the two sections are executed in the different processes and so that the final code is only executed when the child has terminated.

```
int main() {

  // things we both do

  int pid = fork();

      // executed by child process
      :
      :
      return 0;


      // executed by the parent process
      :
      :


  // executed only when the child has terminated
  :
  :

  return 0;
}
```

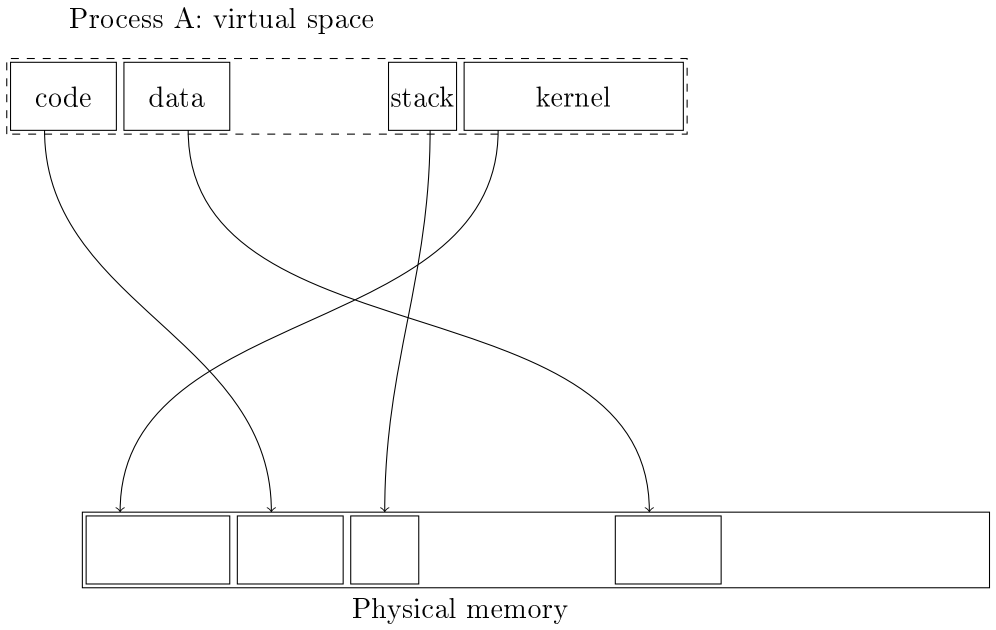
Answer:

```
int main() {
  :
  if(pid == 0) {
    // child
    return 0;
  } else {
    // mother
  }
  wait(NULL);
  // after
}
```

Name:____________________________ Persnr:____________________________

3 Threads and memory [2 points]

Below you find a image of the virtual address space of a process and how it is mapped to the physical memory. If the process creates another thread using a call to `pthread_create()`, the image will change. Draw how the virtual and physical memory will change.



Answer: A new stack is created in the virtual memory and is then mapped to a free segment in memory.

Name:____________________________ Persnr:____________________________

4 a stack, a bottle and.. [2 points]

You have written the program below to examine what is on the stack with the print out as follows.

```
void zot(unsigned long *stop ) {
  unsigned long r = 0x3;
  unsigned long *i;
  for(i = &r;  i <= stop; i++){ printf("%p          0x%lx\n", i, *i);  }
}

void foo(unsigned long *stop ) {
  unsigned long q = 0x2;
  zot(stop);
}

int main() {
  unsigned long p = 0x1;
  foo(&p);
 back:
  printf("  p: %p \n", &p);
  printf("  back: %p \n", &&back);
  return 0;
}
```

```
0x7fff2f14cf58          0x3
0x7fff2f14cf60          0x7fff2f14cf60
0x7fff2f14cf68          0xf3331713173eab00
0x7fff2f14cf70          0x7fff2f14cfa0
0x7fff2f14cf78          0x560ced18173c
0x7fff2f14cf80          0x1
0x7fff2f14cf88          0x7fff2f14cfb0
0x7fff2f14cf90          0x7f0d7750b9a0
0x7fff2f14cf98          0x2
0x7fff2f14cfa0          0x7fff2f14cfc0
0x7fff2f14cfa8          0x560ced18176a
0x7fff2f14cfb0          0x1
  p: 0x7fff2f14cfb0
  back: 0x560ced18176a
```

Point out where the stack base of `zot` is and describe what is found if we add 24 bytes (hex 18) to the base.

Answer: The base is located on `0x7fff2f14cf70` and 24 bytes above this position we find `0x7fff2f14cfb0` which is the adress of the variable `p` in `main()`.

Name:__________________________ Persnr:__________________________

5 a concurrent buffer [2 points]

Below you find the code that removes a value from a buffer that can be accesses by several threads. Implement the corresponding procedure `set(int i)` that adds an element to the buffer.

```
volatile int buffer = 0;
volatile int empty = TRUE;

pthread_cond_t added, removed;

pthread_mutex_t global;

int get() {
  int val;
  pthread_mutex_lock(&global);
  while(empty) {
    pthread_cond_wait(&added, &global);
  }
  val = buffer;
  empty = TRUE;

  pthread_cond_signal(&removed);
  pthread_mutex_unlock(&global);

  return val;
}
```

Answer:

```
void set(int i) {
  pthread_mutex_lock(&global);
  while(!empty) {
    pthread_cond_wait(&removed, &global);
  }
  buffer = i;
  empty = FALSE;

  pthread_cond_signal(&added);
  pthread_mutex_unlock(&global);
}
```

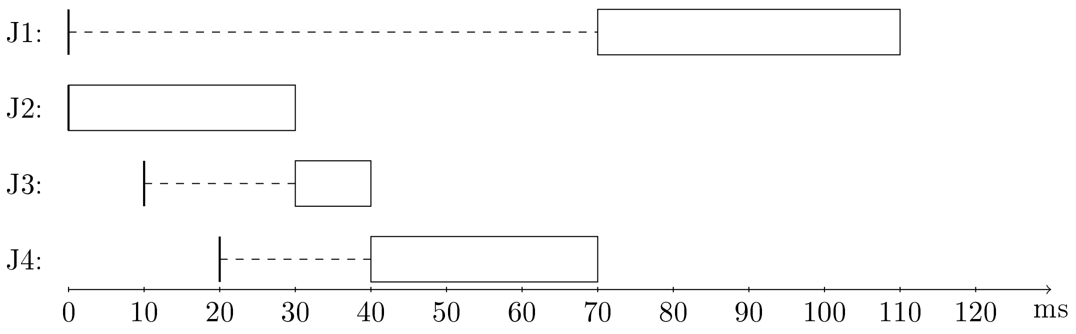
Name:____________________________ Persnr:____________________________

6 Scheduling [P/F]

Assume that we have a scheduler that implements *shortest job first*. We have four jobs described below as ⟨*arrive at*, *execution time* ⟩ in ms. Draw a time diagram and specify the turnaround time for each of the jobs.

Answer:

- J1 : ⟨0,40⟩ 110 ms
- J2 : ⟨0,30⟩ 30 ms
- J3 : ⟨10,10⟩ 30 ms
- J4 : ⟨20,30⟩ 50 ms



Name:____________________________ Persnr:____________________________

7 paged memory with 64 byte pages [P/F]

You have been asked to propose an architecture for a processor that should have a paged virtual memory with the page size as small as 64 byte. The processor is a 16 bit processor and the virtual address space should be 2^{16} bytes.

Propose a scheme that uses a hierarchical page table based on pages of 64 byte and explain how the address translation is done.

Answer: One proposal is to use an offset of 6 bits and then have two levels in the tree with an index of 5 bits on each level. We need 6 bits as offset to address 64 bytes. The 5 bit used as index would mean that we could have 32 elements in a page table. If we encode each entry as two bytes we can encode a table in a page of 64 bytes. Using two bytes as an entry should be sufficient, we can use 10 bits for frame number and 6 bits for flags etc. This would give us a physical space of 64 Ki byte.

Name:____________________________ Persnr:____________________________

8 a simple fs [P/F]

Assume we have a simple file system, built using *inodes* and *data blocks*. Draw how we can represent a folder and two files, `foo.txt` of 5000 bytes and `bar.txt` of 200 bytes using these components. The folder should contain links to the two files. Describe what information is found where and the assumptions you make.

- file size
- file owner
- links from name to files
- number of links to file
- read and write rights of file
- content of file

Answer:

The files are represented by an inode with a pointer to as many blocks as needed. If data blocks are 4 Ki byte, `bar.txt` would need one block and `foo.txt` would need two blocks. The inodes would hold: size, owner, number of links and access rights. The data blocks would of course hold the content of the files.

The map would be represented by an inode holding a reference to one data block. The data block would contain a mapping between the named (bbar.txtänd foo.txt") and the inode numbers.

Name:____________________________ Persnr:____________________________

9 malloc and free [P/F]

Assume that you're looking through an implementation of malloc where you know that the free blocks are kept in a single linked list (`flist` in the code below). The blocks (`chunk`) have one field with its `size` and one pointer to the `next` block in the list. There is a function `after()` that given a block locates the neighboring block in memory. You're looking at the procedure `insert()` below that adds a block to the free-list, but what more does it do?

Explain what the code is doing and point to the relevant sections that do what you describe.

```
void insert(chunk *chnk) {
  chunk *aftr = after(chnk);
  chunk *next = flist;
  chunk *prev = NULL;
  while(next != NULL) {
    if( chnk == after(next) ) {
      next->size = next->size + sizeof(chunk) + chnk->size;
      if(prev != NULL) {
        prev->next = next->next;
      } else {
        flist = next->next;
      }
      chnk = next;
    }
    if(next == aftr) {
      chnk->size = chnk->size + sizeof(chunk) + next->size;
      if(prev != NULL) {
        prev->next = next->next;
      } else {
        flist = next->next;
      }
      next = next->next;
    } else {
      prev = next;
      next = next->next;
    }
  }
  chnk->next = flist;
  flist = chnk;
  return;
}
```

Answer: The prcedure searches through the freelist for free blocks tha are immediate before and/or after (in address order) the block being inserted. If this is the case the blocks are merged into a larger block.

Name:____________________________ Persnr:____________________________

10 Hello from a kernel module [P/F]

You have implemented a small kernel module and added it using `insmod`. The module is written so that one can obtain information, in this case the string "Hello, hello :-)\n" but how do we get this? Describe what you would do to communicate with the module and receive the string as a reply.

```
static int hello_open(struct inode *inode, struct  file *file);

static const struct file_operations hello_fops = {
  .owner = THIS_MODULE,
  .open = hello_open,
  .read = seq_read,
  .llseek = seq_lseek,
  .release = single_release,
};

static int hello_show(struct seq_file *m, void *v) {
  seq_printf(m, "Hello, hello :-)\n");
  return 0;
}

static int hello_open(struct inode *inode, struct file *file) {
  return single_open(file, hello_show, NULL);;
}

static int __init hello_init(void) {
  proc_create("hello", 0, NULL, &hello_fops);
  printk(KERN_INFO "Hello in control\n");
  return 0;
}

static void __exit hello_cleanup(void) {
  remove_proc_entry("hello", NULL);
  printk(KERN_INFO "I'll be back!\n");
}

module_init(hello_init);
module_exit(hello_cleanup);
```

Answer: Simplest way would be to, in a terminal, write:

```
cat /proc/hello
```