EURECOM
Sophia Antipolis

Exam
Operating Systems - OS

Ludovic Apvrille
ludovic.apvrille@telecom-paristech.fr

February, 2017

Authorized documents: Nothing! The grading takes into account the fact that you don't have any documents with you.
A grade is provided for each question (beware: be sure to organize your time with regards to the grading policy). 1 additional point is given for general appreciation, including writing skills and readability.

1 Course knowledge (5 points, ∼20 minutes)

a. If you were to design and program a complex application (e.g., Firefox), would you rather decide to use multiple processes, multiple threads, or both? Discuss the pros and cons of these three alternatives. [1.5 points]

b. What is a deadlock? What are the techniques that are used in general-purpose Operating Systems (Windows, MacOS, Linux) to handle processes that are in a deadlock situation? [1.5 points]

c. First, explain what are swap-out and swap-in operations, and in which case a swap-in operation must be performed. Then, explain why quite frequently that before a swap-in operation can be performed, a swap-out operation must be performed first. [2 points]

2 Page fault (11 points, ∼60 minutes)

For the following questions, using a sample code could facilitate your answer, especially for the last two questions.

a. What is a page fault? Give the main two reasons why it can occur. [1 point]

b. What are the actions taken by an Operating System when a page fault occurs? [2 points]

c. We now assume a system in which the swapping facility has been disabled. Give a C code that will always provoke a page fault, and give another code that may provoke a page fault, but that will not always provoke a page fault. [2 points]

d. Assume two processes sharing code with shared memory. Explain the actions that are executed in the Operating System when:

(a) the shared memory area is created [1 point]

(b) the shared memory area is attached to one of the two processes [1 point]

(c) one of the two processes accesses the shared memory area. [1 point]

e. Assume two threads of two different processes share part of their memory. In which cases do we have to use synchronization mechanisms e.g. *mutex_lock* / *mutex_unlock*? [1.5 points]

f. Continuing the previous question, in which synchronization cases do we have to combine the use of condition variables with *mutex_lock* / *mutex_unlock* ? [1.5 points]

3 Linux kernel (7 points, ∼40 minutes)

a. What is the interest of DMA facilities? In which cases are they used by an Operating System? [1 point]

b. The following code is taken from "dma.c" in the Linux kernel sources. Explain in which context it could be used. Also, explain this code. What are the comments you would add to this code? [1.5 points]

```
1  int request_dma(unsigned int dmanr, const char * device_id)
2  {
3          if (dmanr >= MAX_DMA_CHANNELS)
4                  return -EINVAL;
5
6          if (xchg(&dma_chan_busy[dmanr].lock, 1) != 0)
7                  return -EBUSY;
8
9          dma_chan_busy[dmanr].device_id = device_id;
10
11         /* old flag was 0, now contains 1 to indicate busy */
12         return 0;
13 } /* request_dma */
```

c. Similarly, comment the following function. [1.5 points]

```
1  void free_dma(unsigned int dmanr)
2  {
3          if (dmanr >= MAX_DMA_CHANNELS) {
4                  printk(KERN_WARNING "Trying to free DMA%d\n", dmanr);
```

```
5                   return;
6           }
7
8           if (xchg(&dma_chan_busy[dmanr].lock, 0) == 0) {
9                   printk(KERN_WARNING "Trying to free free DMA%d\n", dmanr);
10                  return;
11          }
12
13  } /* free_dma */
```

d. The following function creates a new message queue in the kernel for IPCs (file msg.c). Comment the main steps of this function. For your information, "RCU" stands for "Read Copy Update" which is a facility of the Linux Kernel that allows concurrent reads and modifications on the same data structure. [3 points]
Note: This question can be considered as a bonus since the exam is graded out of 23.

```
1   static int newque(struct ipc_namespace *ns, struct ipc_params *params)
2   {
3           struct msg_queue *msq;
4           int retval;
5           key_t key = params->key;
6           int msgflg = params->flg;
7
8           msq = kvmalloc(sizeof(*msq), GFP_KERNEL);
9           if (unlikely(!msq))
10                  return -ENOMEM;
11
12          msq->q_perm.mode = msgflg & S_IRWXUGO;
13          msq->q_perm.key = key;
14
15          msq->q_perm.security = NULL;
16          retval = security_msg_queue_alloc(msq);
17          if (retval) {
18                  kvfree(msq);
19                  return retval;
20          }
21
22          msq->q_stime = msq->q_rtime = 0;
23          msq->q_ctime = ktime_get_real_seconds();
24          msq->q_cbytes = msq->q_qnum = 0;
25          msq->q_qbytes = ns->msg_ctlmnb;
26          msq->q_lspid = msq->q_lrpid = 0;
27          INIT_LIST_HEAD(&msq->q_messages);
28          INIT_LIST_HEAD(&msq->q_receivers);
29          INIT_LIST_HEAD(&msq->q_senders);
30
31          /* ipc_addid() locks msq upon success. */
32          retval = ipc_addid(&msg_ids(ns), &msq->q_perm, ns->msg_ctlmni);
33          if (retval < 0) {
34                  call_rcu(&msq->q_perm.rcu, msg_rcu_free);
35                  return retval;
36          }
37
38          ipc_unlock_object(&msq->q_perm);
39          rcu_read_unlock();
40
41          return msq->q_perm.id;
42  }
```