EURECOM
Sophia Antipolis

Exam
Operating Systems - OS

Ludovic Apvrille
ludovic.apvrille@telecom-paristech.fr

February, 8th, 2013

Authorized documents: Nothing! The grading takes into account the fact that you don't have any document with you.
A grade is provided for every question (beware: do organize your time). 1 additional point is given as a general appreciation, including written skills and readability.

1 Understanding of the course (5 points, ∼30 minutes)

1. What is the main purpose of swapping? Can a process be run by an Operating System if some of its pages are swapped out? [2.5 points]

2. What are the two techniques that are commonly used by device drivers to exchange information with devices? Explain the two, and explain in which situations they are efficient, or not. [2.5 points]

2 Memory allocation (6 points, ∼40 minutes)

Memory allocated by Operating Systems is usually a multiple of a given memory page size. Operating Systems commonly store references to allocated pages in linked lists. Yet, programmers like to allocate a memory chunk whose size is not necessarily a multiple of memory pages handled by the Operating System. To do so, user-level libraries manages more fine-grained chunks of memory. For example, *malloc()* is a user-level library function which handles random sizes of memory allocations. That is, that library function allocates necessary pages using system calls (e.g., *mmap()*, *brk()*), and manages allocations within pages using its own data structures (e.g., linked lists).

1. Why isn't it the Operating System directly handling fine-grained allocations? [1 point]

2. Let's now consider the following code:

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

int main() {
  char *str;

  str = (char *)malloc(sizeof(char)*4);
  strcpy(str, "12345678\n");
  printf("str=%s", str);
  free(str);
}
```

   a. At execution, an error may occur when executing *strcpy*: could you explain it precisely, i.e., what is the cause of the error, and how it is detected at run time? In particular, you should explain why sometimes no error occurs. [2 points]

   b. Again at execution, an error may also occur when executing *free*: could you explain it precisely, i.e., what is the cause of the error, and how it is detected at run time? [3 points]

3 POSIX programming (8 points, ∼50 minutes)

Let's consider the following code.
(Memo: pthread_yield() causes the calling thread to relinquish the CPU.)

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

pthread_mutex_t m;
pthread_t a, b;

void *f(void *param) {
  while (1)
    {
      pthread_mutex_lock(&m);
      printf("%s", param);
      pthread_mutex_unlock(&m);
      pthread_yield();
    }
}

int main() {
  pthread_mutex_init(&m, NULL);
  pthread_create(&a, NULL, f, "Hello ");
  pthread_create(&b, NULL, f, "World\n");

  pthread_join(a, NULL);
  pthread_join(b, NULL);
}
```

1. Give two possible traces of execution. [1 point]

2. Modify this code so as to have "Hello World" printed on each line. [2 points]

3. Enhance the code so as to print "Hello World" exactly 10 times. [1 point]

4. Now, we would like to have two threads being able to print "Hello" and two printing "World". Synchronize those 4 threads so as to print exactly 10 times "Hello World". [4 points]