Politecnico di Milano
SCUOLA DI INGEGNERIA INDUSTRIALE E DELL'INFORMAZIONE

Advanced Operating Systems
A.A. 2017-2018 – Exam date: January, 22th 2018

Prof. William FORNACIARI

Surname (readable)....................................... Name (readable)...........................

Matr... Signature......................................

Q1	Q2	TOT

NOTES

It is forbidden to refer to texts or notes of any kind as well as interact with their neighbors. Anyone found in possession of documents relating to the course, although not directly relevant to the subject of the examination will cancel the test. It is not allowed to leave during the first half hour, the task must still be returned, even if it is withdrawn. The presence of the writing (not delivered) implies the renunciation of any previous ratings.

Question Q1 (10 points)

The student is required to describe the general goals and a possible typical architecture of the power supply of an embedded system, also providing details on the benefits, possible problems and architecture of wireless charging.

Question Q2 (13 points)

The following program is a simulator of "triage" in a ER hospital unit. Multiple triage operators are in charge of receiving the patients and queue them in a waiting list, after assigning a priority code, according to the respective emergency level. Codes are red, yellow, green and white in descending order of priority (red is the highest priority). In this ER, one doctor is responsible of "dequeing" the patients and provide assistance.

Given the code already provided, you are asked to write a complete definition and implementation of the classes that are reasonably defined in ***patient.h*** *and* ***synchronized_prio_queue.h,*** *to manage the patients in the proper order.*

```
#include <atomic>
#include <list>
#include <iostream>
#include <memory>
#include <thread>
#include "patient.h"
#include "synchronized_prio_queue.h"
#define NR_TRIAGE_OPERATORS    2

std::atomic<bool> is_open;
std::list<std::shared_ptr<Patient>> arrivals;
SynchronizedPrioQueue<std::shared_ptr<Patient>> waiting_queue;


void queue_patient() {
        /** DETAILS OMITTED  **/
        waiting_queue.put(prio, patient);
        /** ... */
}

void treat_patient() {
        while (is_open) {
                std::shared_ptr<Patient> patient = waiting_queue.get();
                if (patient) {
                        std::cout << "Treating patient: ";
                        patient->print_info();
                }
        }
}

int main() {
        is_open = true;
        arrivals.emplace_back(std::make_shared<Patient>("Francesca Rossi", "MR6700", YELLOW));
        arrivals.emplace_back(std::make_shared<Patient>("Giuseppe Massari", "GM9789", GREEN));
        arrivals.emplace_back(std::make_shared<Patient>("Michele Zanella", "MZ1729", RED));
        arrivals.emplace_back(std::make_shared<Patient>("Federico Terraneo", "FT0000", RED));
        std::list<std::thread> triage_operators;
        for (int i=0; i < NR_TRIAGE_OPERATORS; ++i)
                triage_operators.emplace_back(std::thread(queue_patient));

        waiting_queue.print_status();
        std::thread doctor(treat_patient);

        /** … Wait some time… **/

        is_open = false;
        waiting_queue.set_done();
        for (auto & t: triage_operators) t.join();
        doctor.join();
        return 0;
}
```

Output example:

```
[0]: Michele Zanella, Federico Terraneo,
[1]: Francesca Rossi,
[2]: Giuseppe Massari,
Treating patient: [MZ1729] Michele Zanella, priority: 0
Treating patient: [FT0000] Federico Terraneo, priority: 0
Treating patient: [MR6700] Francesca Rossi, priority: 1
Treating patient: [GM9789] Giuseppe Massari, priority: 2
```

Possible solution

patient.h:

```
#ifndef PATIENT_H
#define PATIENT_H

#include <iostream>
#include <string>

enum priority_code { RED, YELLOW, GREEN, WHITE };


class Patient
{
public:
	Patient():
		name(""), ssn_id(""), priority(WHITE) {}

	Patient(std::string _name, std::string _ssn_id, priority_code _prio = WHITE):
		name(_name), ssn_id(_ssn_id), priority(_prio) {}

	virtual ~Patient() { }

	const std::string get_name() const { return name;  }
	const std::string get_ssn_id() const { return name; }
	void set_priority(priority_code _pc) { priority = _pc; }
	priority_code get_priority() const { return priority; }

	void print_info() {
		std::cout << "[" << ssn_id << "] " << name << ", priority: " << priority << std::endl;
	}

private:
	std::string name;
	std::string ssn_id;
	priority_code priority;
};

#endif /* PATIENT_H */
```

synchronized_prio_queue.h:

```
#ifndef SYNC_PRIO_QUEUE_H_
#define SYNC_PRIO_QUEUE_H_

#include <iostream>
#include <list>
#include <map>
#include <mutex>
#include <condition_variable>
#include <vector>

#define NR_PRIORITIES    4

template<typename T>
class SynchronizedPrioQueue
{
public:

	SynchronizedPrioQueue(int nr_priorities=NR_PRIORITIES) {
		done = false;
		queue.resize(nr_priorities);
	}

	~SynchronizedPrioQueue() {
		std::unique_lock<std::mutex> lck(mux);
		queue.clear();
		done = true;
		cv.notify_all();
	}
```

```
	void put(int prio, const T& data) {
		std::unique_lock<std::mutex> lck(mux);
		if (prio >= queue.size()) {
			std::cout << "Error. Max priority value allowed: "
				<< NR_PRIORITIES-1 << std::endl;
		}
		queue[prio].push_back(data);
		count++;
		cv.notify_all();
	}


	T get()  {
		std::unique_lock<std::mutex> lck(mux);
		while(count == 0 && !done)
			cv.wait(lck);
		T result;
		for (auto & prio_list: queue) {
			T result;
			if (prio_list.empty())
				continue;
			result = prio_list.front();
			prio_list.pop_front();
			count--;
			return result;
		}
		return result;
	}


	void print_status() {
		std::unique_lock<std::mutex> lck(mux);
		int curr_prio = 0;
		for (auto & prio_list: queue) {
			std::cout << "[" << curr_prio++ << "]: ";
			for (auto & data: prio_list) {
				std::cout << data->get_name() << ", ";
			}
			std::cout << std::endl;
		}
	}

	void set_done() {
		done = true;
		cv.notify_all(); // wake up possible waiting threads
	}


private:
	SynchronizedPrioQueue(const SynchronizedPrioQueue&)=delete;
	SynchronizedPrioQueue& operator=(const SynchronizedPrioQueue&)=delete;
	std::vector<std::list<T>> queue;
	std::mutex mux;
	std::condition_variable cv;
	unsigned int count = 0;
	std::atomic<bool> done;
};

#endif //SYNC_PRIO_QUEUE_
```