System and Device- Programming
Standard Exam
13.01.2025

Ex 1 (1.0 points)

Consider the following program. Select the correct statement(s) from the options provided below. Note that incorrect answers imply a penalty in the final score.

```
class C {
public:
    C() { cout << "[C]"; }
    ~C() { cout << "[D]"; }
    C(const C&) { cout << "[CC]"; }
    C& operator=(const C&) { cout << "[CAO]"; return *this; }
    C(C&&) noexcept { cout << "[MC]"; }
    C& operator=(C&&) noexcept { cout << "[MAO]"; return *this; }
};

void f1(C c) {
    cout << "{f1}";
}

void f2(C& c) {
    cout << "{f2}";
}

C f3() {
    cout << "{f3}";
    C local_c;
    return local_c;
}

int main() {
    cout << "{1}"; C c1;
    cout << "{2}"; f1(c1);
    cout << "{3}"; f2(c1);
    cout << "{4}"; C c2 = f3();
    cout << "{5}"; c1 = f3();
    cout << "{6}"; return 0;
}
```

Choose one or more options

1. ⬜ Line number 1 includes one copy constructor.
2. ⬜ Line number 2 includes one standard constructor.
3. ⬜ Function f1 includes one destructor.
4. ⬜ Line number 3 includes one copy constructor.
5. ⬜ Function f2 includes one destructor.
6. ⬜ Line number 4 includes one standard constructor.
7. ⬜ Line number 5 includes one standard constructor and possibly one move assignment operator.
8. ⬜ Line number 6 includes two destructors.
9. ⬜ Line number 6 includes three destructors.

Ex 2 (1.0 points)

Consider the following program written in pseudo-code and including three threads T1, T2, and T3. The semaphores sem_a, sem_b, and sem_c are initialized as reported in the example. Select the correct statement(s) from the options provided below. Note that incorrect answers imply a penalty in the final score.

```
Initialization:
init(sem_a, 1); init(sem_b, 1); init(sem_c, 1);

Threads:
T1                  T2                  T3
wait(sem_a);        wait(sem_b);        wait(sem_c);
printf("A");        printf("C");        printf("E");
wait(sem_b);        wait(sem_c);        wait(sem_a);
printf("B");        printf("D");        printf("F");
signal(sem_b);      signal(sem_c);      signal(sem_a);
signal(sem_a);      signal(sem_b);      signal(sem_c);
```

Choose one or more options

1. ⬜ The sequence A C D E B F is an admissible output.
2. ⬜ The sequence A C E B D F is an admissible output.
3. ⬜ The sequence C E F A D B is an admissible output.
4. ⬜ The sequence A B C D E F is an admissible output.
5. ⬜ The sequence E A B C D F is an admissible output.
6. ⬜ The sequence E C A F D B is an admissible output.
7. ⬜ The sequence A E C is an admissible output.
8. ⬜ The sequence C E A D F B is an admissible output.
9. ⬜ The sequence E C A is an admissible output.
10. ⬜ The program is subject to deadlock.

Ex 3 (1.0 points)

The following C++ and CUDA program computes the sum of two arrays v1 and v2, in a third array v3. Reports two specific values:

1. The number of threads that are globally run
2. Which would be the element manipulated by the thread with threadIdx.x = 2, threadIdx.y = 3, blockIdx.x = 1, blockIdx.y = 2.

Reports the two integer values one after the other, separated by a single space.

```
__global__ void addVectors2D(const int* v1, const int* v2, int* v3, int width) {
    int col = blockIdx.x * blockDim.x + threadIdx.x;
    int row = blockIdx.y * blockDim.y + threadIdx.y;

    int index = row * width + col;
    if (col < width && row < width) {
        v3[index] = v1[index] + v2[index];
    }
}

int main() {
    int width = 64;
    int numElements = width * width;
    size_t size = numElements * sizeof(int);

    std::vector<int> h_v1(numElements), h_v2(numElements), h_v3(numElements);
    int *d_v1 = nullptr, *d_v2 = nullptr, *d_v3 = nullptr;
```

```
    for (int i=0; i<numElements; ++i) {
        h_v1[i] = i;
        h_v2[i] = 2;
    }

    cudaMalloc(&d_v1, size);
    cudaMalloc(&d_v2, size);
    cudaMalloc(&d_v3, size);

    cudaMemcpy(d_v1, h_v1.data(), size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_v2, h_v2.data(), size, cudaMemcpyHostToDevice);

    dim3 threadsPerBlock (4, 4, 1);
    dim3 numBlocks (width/threadsPerBlock.x, width/threadsPerBlock.y, 1);

    addVectors2D<<<numBlocks, threadsPerBlock>>>(d_v1, d_v2, d_v3, width);

    cudaMemcpy(h_v3.data(), d_v3, size, cudaMemcpyHostToDevice);

    cudaFree(d_v1);
    cudaFree(d_v2);
    cudaFree(d_v3);

    return 0;
}
```

Ex 4 (2.5 points)

Specify what templates are in C++ and why they are helpful. Then write the following code using them.
A `findMax` template function is designed to be a generic function that can accept any two values of the same type and return the larger of the two.
A `KeyValuePair` template class is a generic container designed to hold a pair of associated values: a key (of type K) and a value (of type V), which allows the key and the value to be of completely different data types. The class constructor initializes the key-value pair, and the `getKey` and `getValue` methods provide access to it.
Write a short program to show how to compare two values of different types using the first function, and how to create instances of `KeyValuePair` with different type combinations.

Ex 5 (3.5 points)

Implement in C++ (using barriers) the following program that simulates a multi-stage scientific computation.
The main program runs N threads. Each thread goes through three distinct stages:

1. In the first stage, each thread randomly initializes a vector of random size with random integer values.
2. In the second stage, each thread sorts the data in the vector randomly using an ascending or descending order.
3. In the third stage, each thread verifies that the vector has really been sorted as desired during the second stage.

Notice that no thread can start stage 3 until all threads have successfully finished stage 2, and no threads can start stage 2 until all threads have ended stage 1.

Ex 6 (2.5 points)

Explain the strategy of multiplexing IO using the POSIX system call `select`. Show an example in which it is possible to write to three and read from two channels.

Ex 7 (3.0 or 4.0 points)

Given two large vectors (arrays) of numbers, `v1` and `v2`, of the same size `n`, calculate their dot product.

The dot product is defined as the sum of the products of their corresponding elements:
`result = v1[0]*v2[0] + v1[1]*v2[1] + ... + v1[n-1]*v2[n-1]`

Each candidate is required to deliver one (**and only one**) of the following two different implementations:

- [Max 3.0 points] A C++ one using promises and futures in which:
    - In stage one, each thread computes a different product mult[i] = v1[i] * v2[i] and stores it in an intermediate array in global memory.
    - In stage two, a sequential iteration (on the host) performs the sum of all elements mult[i].
- [Max 4.0 points] A CUDA one in which:
    - In stage one, a simple kernel runs n threads to calculate the product mult[i] = v1[i] * v2[i] and stores it in an intermediate array in global memory.
    - In stage two, a sequential iteration (on the host) performs the sum of all elements mult[i].