Laborator 11 – Standard Template Library

Tema: Bitdefender – Sistem de monitorizare a proceselor

Autor: ing. Eduard Donea

Poveste

Bitdefender monitorizează în timp real zeci de procese care rulează în sistemul de operare: consum de CPU, utilizare memorie, procese suspecte, alerte, comportamente neobișnuite și evenimente critice.

Pentru a gestiona acest flux continuu se utilizează un modul scris în C++, bazat intens pe **STL**, capabil să ordoneze procesele, să filtreze alertele, să recomande procesul optim pentru închidere și să gestioneze cozi și stive de evenimente critice.

Studenții vor implementa un sistem complet în două fișiere: `Bitdefender.h` și `Bitdefender.cpp`, plus un `main.cpp` pentru testare.

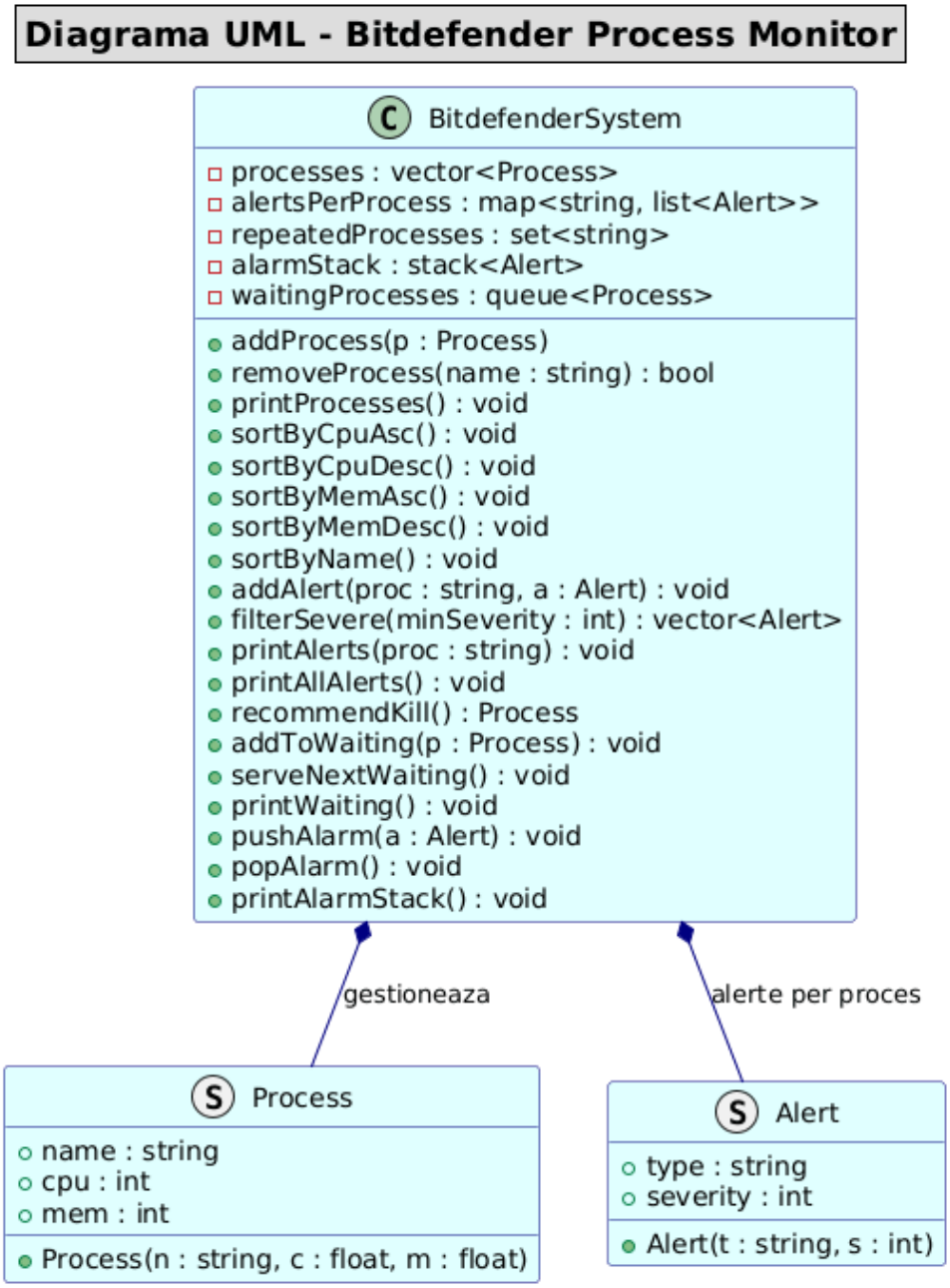
Scopul laboratorului

- folosirea containerelor STL: `vector`, `list`, `map`, `set`, `stack`, `queue`;
- sortări cu **lambda functions**: după CPU, RAM, nume;
- filtrarea și procesarea alertelor cu `copy_if`;
- recomandarea procesului optim pentru închidere (lambda + max logic);
- folosirea unui **struct** pentru date simple (Process, Alert);
- iteratoare directe și inverse.

Structura proiectului

- **Bitdefender.h – Process** – struct pentru procese (nume, CPU, RAM);
- **Bitdefender.h – Alert** – struct pentru alerte (tip, severitate);
- **Bitdefender.h – BitdefenderSystem** – clasa principală cu containerele STL;
- **main.cpp** – fișierul de testare.

Figura 1 – Diagrama UML a sistemului Bitdefender



Fișierul `Bitdefender.cpp`

```cpp
#include "Bitdefender.h"

// =========================
// STRUCTS IMPLEMENTATION
// =========================

// TODO: Implementati constructorii structurilor Process si Alert.

// =========================
// BITDEFENDER SYSTEM
// =========================

// TODO: Implementati constructorul clasei.

// ==============================
//      GESTIONARE PROCESE
// ==============================
void BitdefenderSystem::addProcess(const Process& p) {
    // TODO:
    // - Adaugati procesul in vectorul processes.
}

bool BitdefenderSystem::removeProcess(const std::string& name) {
    // TODO:
    // - Folositi remove_if + lambda
    // - Stergeti procesele eliminate cu erase()
    // - Returnati true daca s-a sters ceva
}

// ==============================
//   SORTARI CU LAMBDA
// ==============================
void BitdefenderSystem::sortByCpuAsc() {
    // TODO: sort + lambda (cpu crescator)
}

void BitdefenderSystem::sortByCpuDesc() {
    // TODO: sort + lambda (cpu descrescator)
}

void BitdefenderSystem::sortByMemAsc() {
    // TODO: sort + lambda (mem crescator)
}

void BitdefenderSystem::sortByMemDesc() {
    // TODO: sort + lambda (mem descrescator)
}
```

```cpp
void BitdefenderSystem::sortByName() {
    // TODO: sort + lambda alfabetic
}

// ==============================
//          ALERTE
// ==============================
void BitdefenderSystem::addAlert(const std::string& proc,
                                 const Alert& a) {
    // TODO:
    // - alertsPerProcess[proc].push_back(a)
    // - Daca severitate > un prag -> repeatedProcesses.insert(proc)
}

void BitdefenderSystem::printAlerts(const std::string& proc) const {
    // TODO: Parcurgeti lista alertsPerProcess[proc]
}

void BitdefenderSystem::printAllAlerts() const {
    // TODO: Parcurgeti entire map-ul
}

std::vector<Alert> BitdefenderSystem::filterSevere(int minSev) const {
    // TODO:
    // - Creati vector result
    // - copy_if cu lambda: a.severity >= minSev
    // - Returnati result
}

// ==============================
//  RECOMANDARE PROCES DE TERMINAT
// ==============================
Process BitdefenderSystem::recommendKill() const {
    // TODO:
    // - Alegeti procesul cu cpu*mem maxim
    // - max_element + lambda
}

// ==============================
//     QUEUE + STACK
// ==============================
void BitdefenderSystem::addWaiting(const Process& p) {
    // TODO: waitingQueue.push(p)
}

void BitdefenderSystem::serveNextWaiting() {
    // TODO:
    // - Afisati si scoateti front(), daca exista
}
```

```
void BitdefenderSystem::pushAlarm(const Alert& a) {
    // TODO: alarmStack.push(a)
}

void BitdefenderSystem::popAlarm() {
    // TODO: Afisati si scoateti top(), daca exista
}

// ==============================
//           PRINT PROCESE
// ==============================
void BitdefenderSystem::printProcesses() const {
    // TODO: Parcurgeti processes cu iteratori
}
```

Structurile `Process` și `Alert`

Structurile `Process` și `Alert` sunt modele simple de date folosite în toate containerele STL.
Process: nume, CPU, RAM **Alert**: tip, severitate
În UML sunt tratate identic cu clasele deoarece stochează informație esențială pentru sistemul de securitate.

Clasa `BitdefenderSystem`

Clasa gestionează întregul flux de monitorizare:

- `vector<Process>` – procesele active;
- `map<string, list<Alert>>` – alerte asociate fiecărui proces;
- `set<string>` – procese care au primit alerte severe repetate;
- `queue<Process>` – procese care așteaptă scanarea;
- `stack<Alert>` – stivă de alarme critice.

Sunt folosiți algoritmi STL precum `std::sort`, `std::max_element`, `std::copy_if`, iteratori direcți și inversi și funcții `lambda` pentru comparatori.

Fișierul `main.cpp`

```cpp
#include "Bitdefender.h"

int main() {

    BitdefenderSystem sys;

    // ============================
    // TODO A: Adaugati procese
    // ============================
    // Exemplu: sys.addProcess({"chrome.exe", 37, 1200});

    // TODO B: Afisati toate procesele

    // TODO C: Sortati dupa CPU, RAM si nume

    // TODO D: Adaugati alerte pentru unele procese

    // TODO E: Filtrati alertele severe >= un prag

    // TODO F: Recomandati procesul optim de terminat

    // TODO G: Folositi queue pentru procese in asteptare

    // TODO H: Folositi stack pentru stiva de alarme

    return 0;

}
```

Cerințe (10p)

1. (2p) Implementați toate funcțiile din `Bitdefender.cpp`.
2. (2p) Sortări cu lambda (CPU, RAM, nume).
3. (2p) Filtrare alerte severe cu `copy_if`.
4. (2p) Recomandare proces optim (lambda + max_element).
5. (2p) Testare completă în `main.cpp`.

Compilare și rulare

```
g++ -std=c++17 -g *.cpp -o lab11
valgrind ./lab11
```

Dacă programul nu compilează, se scad 4p.

Bonus GitHub Classroom

Studentii pot obține până la **3p bonus** dacă încarcă toate laboratoarele până la finalul semestrului sau **1p** dacă trimit cel puțin jumătate.

1. Accesați cursul de pe OCW – Programare Orientată pe Obiecte.
2. Mergeți la secțiunea **Bonus GitHub laborator**.
3. Acceptați tema pentru Laborator 11.

Atenție: Repository-urile sunt monitorizate automat. Codul copiat între studenți anulează bonusul pentru toți participanții.