




## Laborator 02

### Setup infrastructură (Făcut în Laborator 01)

- Instalați Windows Subsystems for Linux.
  - Control Panel >> **Windows Features** >> Selectați **Windows Subsystems for Linux** >> **OK**
- Instalați Ubuntu 20.04.
  - Microsoft Store >> Search Ubuntu >> Ubuntu 20.04 >> Install >> Launch
- Asigurați-vă că ați primit cheia (de la asistent pe Teams) și salvați-o.
- Instalați [Putty](#).
  - Host Name: username@20.52.209.189
    - username este cel de pe wiki.mta.ro
  - Connections >> SSH >> Auth >> Browse... pentru a pune cheia.
- Instalați [WinSCP](#).
  - Când instalați de la *User Interface Style* selectați *Commander*.
  - New Site
    - Host name: 20.52.209.189
    - Port number: 22
    - User name: cel de pe wiki.mta.ro, **fără** @wiki.mta.ro
  - Advanced...
    - SSH >> Authentication >> Private Key File [...] >> OK
  - Save >> Login
  - Stânga mergeți în directorul dorit - Dreapta folder-ul de pe server
  - Mergeți în folder-ul labs pe server.
  - La începutul laboratorului copiați de pe server pe local.
    - **Atenție** copiați din folderul labs (folderul 01 conține un folder .vscode care este invizibil).
  - **La sfârșitul laboratorului copiați de pe local pe server.**
- Instalați compilator și make pe Linux.
  - `sudo apt-get install gcc`
  - `sudo apt-get install make`
  - `sudo apt-get install gdb`
- Instalați MPI pe Linux.
  - `sudo apt-get install libopenmpi-dev openmpi-bin`
  - `sudo apt-get install openmpi-doc openmpi-common`
- Instalați [Visual Studio Code](#).
- Instalați extensii Visual Studio Code:
  - C/C++ (IntelliSense) – autor Microsoft (**trebuie instalat în WSL**)
  - Remote-WSL – autor Microsoft
- Setati Visual Studio Code să folosească WSL (Windows Subsystems for Linux).
  - Stânga jos buton verde două săgeți 
  - Remote-WSL: New Window
    - Dacă aveți mai multe distribuții instalate e bine să apăsați Remote-WSL: New Window using Distro... și apoi să o selectați pe cea cu Ubuntu 20.04
  - Open folder...
  - **Scrieți /mnt/ în loc de /root** . Selectați partiția și acum sunteți prezentat cu lista de directoare Windows. Folosiți directorul în care doriți să lucrați.
  - **Trebuie să apară în Visual Studio subfolderul .vscode**



### Exerciții

**Pentru fiecare exercițiu se va scrie în fișierul `_REPORT.txt` rezultatul rulărilor și răspunsul la întrebări.**

1. (**helloWorld.c**) Citiți, compilați și rulați programul helloWorld. În Makefile aveți un exemplu de compilare. Rulare se va face folosind **`mpirun -np 4 ./helloWorld`**.
2. (**send.c**) Implementați un program MPI cu doua procese. Procesul 0 va trimite o valoare procesului doi. După primire acesta o va afișa. Aveți grijă să inițializați variabila doar pe procesul 0.
3. (**send100.c**) Modificați programul anterior în așa fel încât în loc de un element să fie transmis un vector de 100 de elemente o dată, printr-un singur apel. Aveți grijă să inițializați vectorul doar pe procesul 0.
4. (**broadcast.c**) Implementați un program MPI cu 4 procese. Folosind broadcast un programul 2 trimite o valoare tuturor celorlalte. Aveți grijă să inițializați variabila doar pe programul 2. După trimitere afișați variabila de pe toate procesele.
5. (**broadcast100.c**) Modificați programul anterior în așa fel încât în loc de un element să fie transmis un vector de 100 de elemente o dată, printr-un singur apel. Aveți grijă să inițializați vectorul doar pe programul 2. După trimitere afișați vectorul de pe toate procesele.
6. (**scatterGather.c**) Implementați un program MPI cu 4 procese. Procesul 0 inițializează vectorul de 100 de elemente după regula **`v[i]=i`**. Vectorul este împărțit tuturor proceselor. Fiecare din cele 4 procese adună valoarea 42 elementelor din vector de care este responsabil (25 de elemente fiecare). După adunări, vectorul va fi colectat pe procesul 0 și afișat complet.
7. (**circle.c**) Implementați programul MPI cu **N** procese. Procesul 0 trimite procesului următor valoarea **1**. Toate celelalte procese primesc valoarea de la procesul dinaintea lor, adaugă **2** la ea și trimit valoarea mai departe procesului următor. Ultimul proces, după adunare, trimite valoarea procesului 0, formând un cerc. La fiecare send, recv, și adunare se vor face afișări.
8. (**anySource.c**) Scrieți un program MPI cu 4 procese.
  - Primele 3 procese trimit o valoare ultimului.
  - Ultimul proces primește cele trei valori cu **`MPI_ANY_SOURCE`**.
  - Se prindează de pe procesul **3** valoarea și sursa din **`MPI_Status`**.
9. (**anyTag.c**) Scrieți un program MPI cu 2 procese.
  - Procesul **0** trimite 3 valori procesului **1**, fiecare cu alt tag.
  - Procesul **1** va primi valorile folosind **`MPI_ANY_TAG`**.
  - Se prindează valorile de pe procesul **1** și tag-ul din **`MPI_Status`**.

**Exercițiile de la 1 la 9 sunt obligatorii.** Conceptele explorate sunt esențiale pentru obținerea notei **minime** de promovare.



Vă recomandăm, pentru a crește șansele de a obține o notă cât mai mare să explorați și următoarele exerciții:

10. Implementați un program secvențial pentru calculul [PI folosind algoritmul Monte Carlo](#).
  - Se definește un pătrat și un cerc circumscris acestuia.
  - Calculele sunt mai simple dacă pătratul are latura **2**.
  - Sunt generate puncte (două coordonate) aleatoriu înăuntrul acestui pătrat.
  - Se numără câte puncte sunt și în interiorul cercului. Distanța de la centrul cercului la punct este mai mică decât raza.
  - Raportul dintre numărul total de puncte generate în pătrat și numărul de puncte din cerc este egal cu raportul dintre aria pătratului și aria cercului. Folosind această formulă se poate calcula **PI** în funcție de primul raport.
11. Implementați programul anterior distribuit cu MPI.
  - Aveți grijă ca numărul de puncte generate în programul secvențial să fie egal cu numărul de puncte total din programul distribuit.

### Hints:

Dacă aveți problema următoare când rulați cu mpirun:

```
-----  
WARNING: Linux kernel CMA support was requested via the  
btl_vader_single_copy_mechanism MCA variable, but CMA support is  
not available due to restrictive ptrace settings.  
  
The vader shared memory BTL will fall back on another single-copy  
mechanism if one is available. This may result in lower performance.
```

Pentru a rezolva rulați ca root comanda:

```
echo 0 > /proc/sys/kernel/yama/ptrace_scope
```

Dacă aveți o problemă de genul când rulați cu mpirun:

```
-----  
There are not enough slots available in the system to satisfy the 100  
slots that were requested by the application:  
  
./helloWorld  
  
Either request fewer slots for your application, or make more slots  
available for use.  
  
A "slot" is the Open MPI term for an allocatable unit where we can  
launch a process. The number of slots available are defined by the  
environment in which Open MPI processes are run:
```

## Sisteme Tolerante la Defecte

---

Adăugați comenzii mpirun parametrul **--oversubscribe**

