Universitas Semarang
Fakultas Teknologi Informasi dan Komunikasi
Teknik Informatika

Sistem Terdistribusi

Modul Praktikum Mahasiswa

Oleh:
Alauddin Maulana Hirzan, S. Kom., M. Kom
NIDN. 0607069401

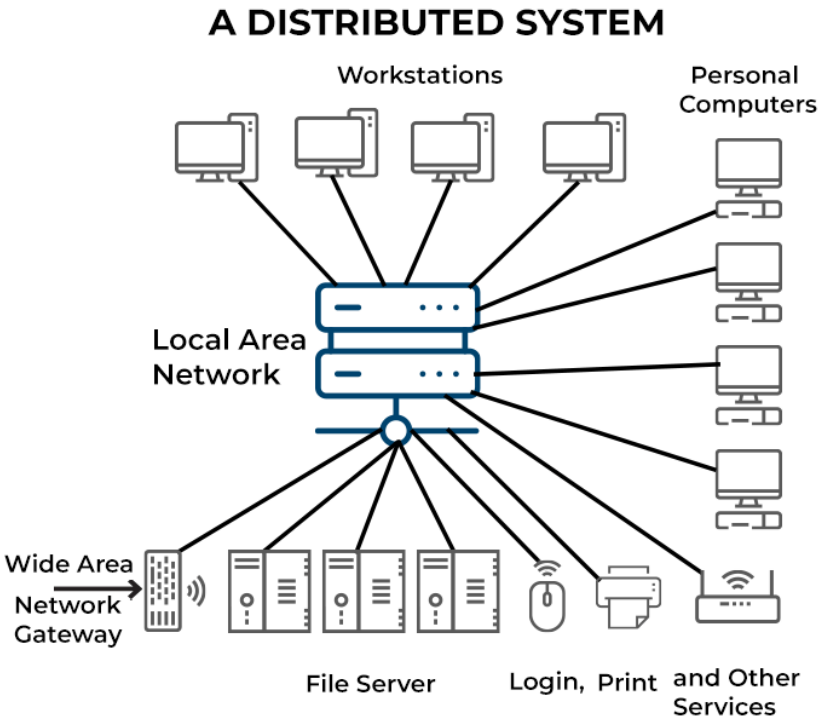
Daftar Isi

Daftar Gambar

Pendahuluan

0.1 Mengenal Sistem Terdistribusi

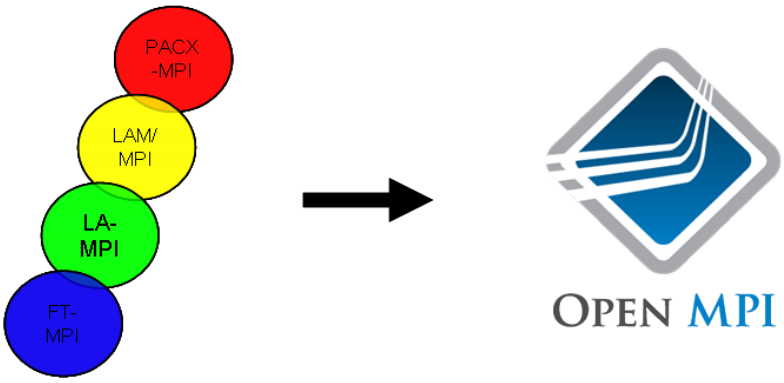
Sistem terdistribusi adalah kumpulan beberapa komputer otonom yang saling terhubung melalui jaringan dan bekerja secara terkoordinasi untuk mencapai tujuan bersama, sehingga bagi pengguna tampak sebagai satu sistem terpadu. Dalam konteks praktikum, sistem ini biasanya melibatkan pembagian tugas (task distribution) ke beberapa node, komunikasi antar proses menggunakan protokol tertentu (misalnya message passing), serta pengelolaan sumber daya secara efisien untuk meningkatkan kinerja, skalabilitas, dan toleransi kesalahan. Konsep utama yang dipelajari meliputi paralelisme, sinkronisasi, konsistensi data, dan mekanisme komunikasi seperti remote procedure call atau message queue, yang menjadi dasar implementasi teknologi modern seperti cloud computing dan cluster computing.



Gambar 1: Sistem Terdistribusi

0.2 Mengenal OpenMPI

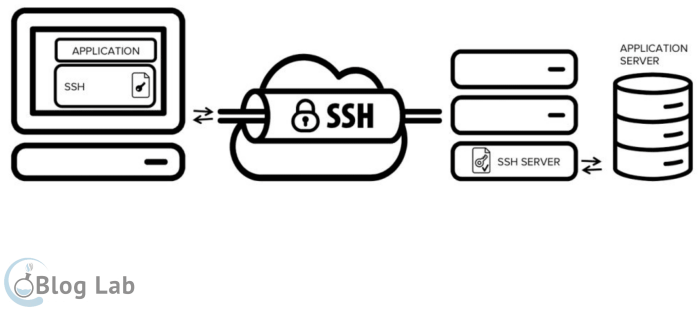
OpenMPI adalah implementasi open-source dari standar Message Passing Interface (MPI) yang digunakan untuk membangun aplikasi komputasi paralel dan terdistribusi, khususnya pada sistem cluster dan high-performance computing (HPC). OpenMPI menyediakan mekanisme komunikasi antar proses yang berjalan pada banyak node, baik dalam satu mesin maupun di jaringan, menggunakan teknik message passing seperti point-to-point dan collective communication. Dalam praktikum, OpenMPI memungkinkan distribusi tugas ke beberapa worker node secara efisien, mengatur sinkronisasi proses, serta mengoptimalkan penggunaan sumber daya komputasi untuk mempercepat eksekusi program paralel.



Gambar 2: OpenMPI

0.3 Mengenal SSH

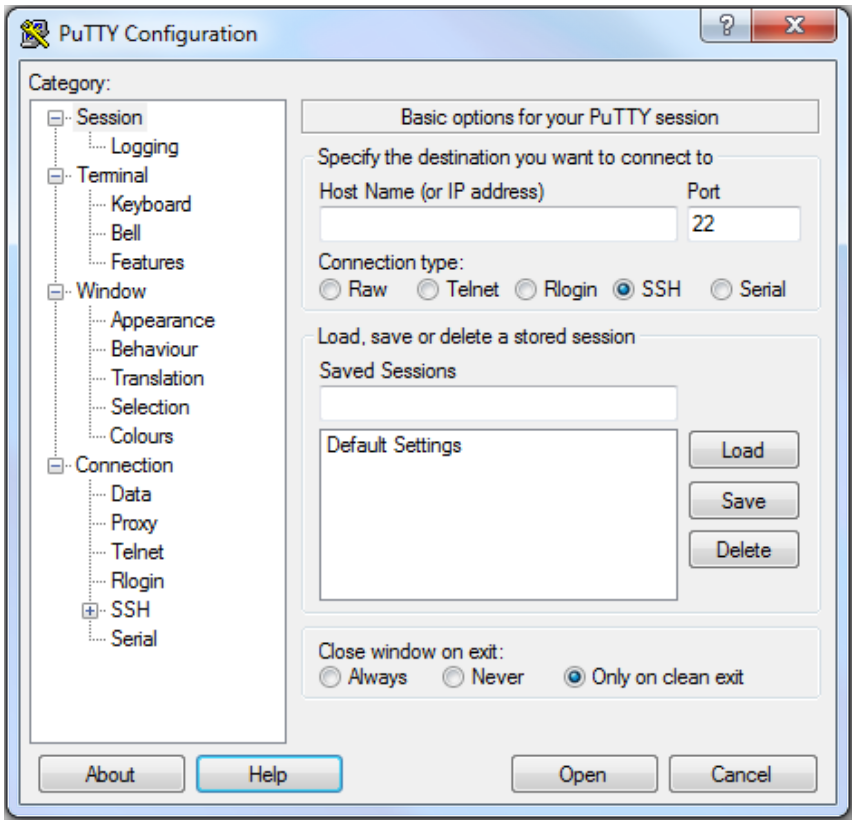
SSH (Secure Shell) adalah protokol jaringan yang digunakan untuk mengakses dan mengelola sistem komputer secara aman melalui koneksi jarak jauh dengan enkripsi. SSH memungkinkan pengguna untuk melakukan login ke server, menjalankan perintah, mentransfer file (melalui SCP atau SFTP), serta mengelola konfigurasi sistem tanpa risiko penyadapan data, karena seluruh komunikasi dilindungi dengan kriptografi. Dalam praktikum sistem terdistribusi, SSH sering digunakan untuk menghubungkan controller node dengan worker node secara remote, termasuk dalam skenario otomatisasi seperti passwordless SSH untuk mendukung eksekusi paralel menggunakan tools seperti OpenMPI.



Gambar 3: SSH

0.4 Mengenal PuTTY

PuTTY adalah aplikasi klien open-source yang digunakan untuk melakukan koneksi jarak jauh ke server melalui protokol seperti SSH, Telnet, dan serial. PuTTY banyak digunakan pada sistem operasi Windows untuk mengakses server berbasis Linux atau Unix karena menyediakan antarmuka terminal yang sederhana namun kuat, serta mendukung fitur keamanan seperti enkripsi pada SSH. Dalam praktikum sistem terdistribusi, PuTTY digunakan untuk menghubungkan controller node ke worker node, menjalankan perintah secara remote, serta mengelola konfigurasi sistem jaringan dan komputasi secara efisien.



Gambar 4: PuTTY

Persiapan Praktikum

Agar praktikum dapat berjalan dengan lancar, mahasiswa diwajibkan memenuhi persyaratan berikut baik dalam bentuk perangkat keras maupun lunak:

0.5 Perangkat Keras

- Prosesor dengan 4 inti
- RAM minimal 8GB
- HDD 10GB

0.6 Perangkat Lunak

Perangkat lunak berikut ini wajib diinstall oleh mahasiswa demi lancarnya praktikum:

- PuTTY
  - Download via https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html

Bab 1

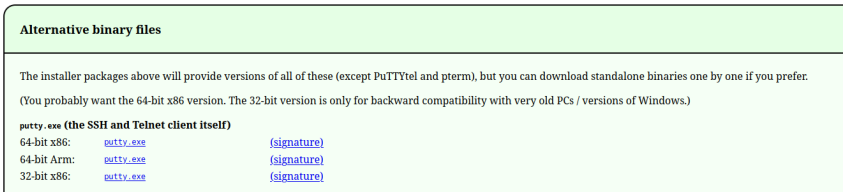
Eksplorasi Lingkungan MPI Cluster

1.1 Pendahuluan

Perkembangan komputasi modern menuntut kemampuan pemrosesan data dalam skala besar dan waktu yang efisien. Salah satu pendekatan yang digunakan untuk memenuhi kebutuhan tersebut adalah komputasi paralel dan terdistribusi, di mana beberapa komputer bekerja secara bersamaan untuk menyelesaikan suatu permasalahan. Konsep ini menjadi dasar dalam berbagai bidang seperti komputasi ilmiah, analisis data besar, kecerdasan buatan, dan simulasi sistem kompleks. Dalam praktikum ini, lingkungan yang digunakan adalah cluster komputasi yang terdiri dari satu controller node dan beberapa worker node yang telah terkonfigurasi dengan OpenMPI serta pustaka mpi4py. Selain itu, koneksi antar node telah diatur menggunakan mekanisme secure shell tanpa password (passwordless SSH), sehingga memungkinkan eksekusi program secara terdistribusi tanpa hambatan autentikasi manual. Pertemuan pertama ini difokuskan pada eksplorasi lingkungan MPI cluster. Mahasiswa akan mempelajari struktur dasar cluster, memahami peran masing-masing node, serta melakukan pengujian konektivitas dan eksekusi program sederhana menggunakan MPI. Kegiatan ini bertujuan untuk memberikan pemahaman awal mengenai bagaimana proses paralel dijalankan dan dikelola dalam lingkungan terdistribusi.

1.2 Tutorial

1. Untuk memulai praktikum ini, mahasiswa diwajibkan menggunakan software *PuTTY* atau *MobaXTerm* atau *Terminal*. PuTTY sendiri dapat diunduh secara gratis melalui website: putty dan unduh versi *64-bit x86*

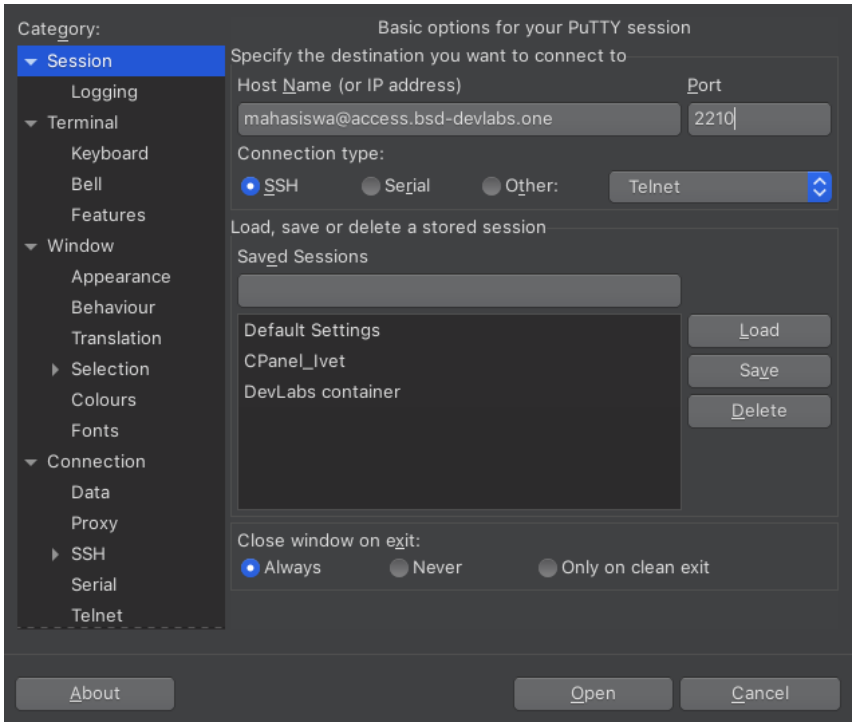


Gambar 1.1: Mengunduh Putty

2. Berikutnya adalah mengakses *node controller* yang sudah disiapkan. Menggunakan

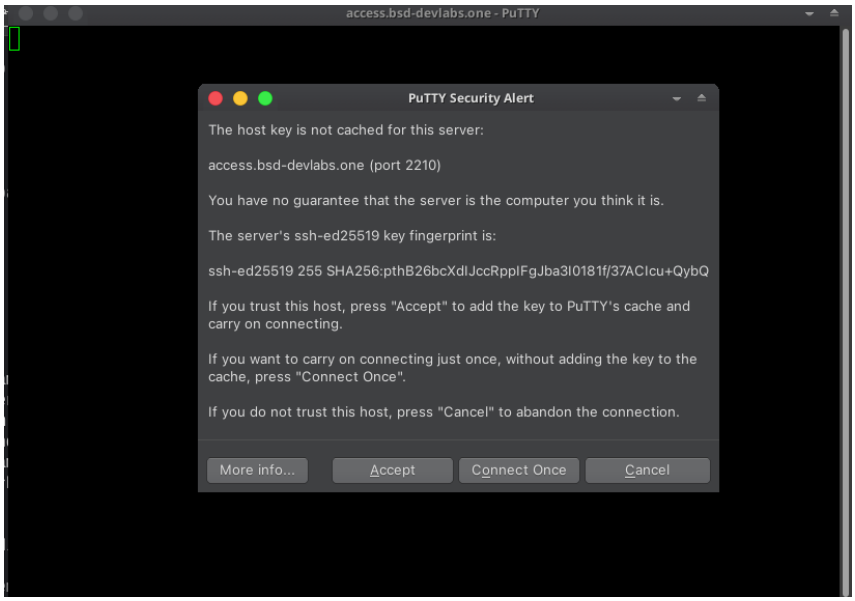
PuTTY, masukkan informasi berikut lalu klik *Open*

- hostname : access.bsd-devlabs.one
- port : 2210
- username : mahasiswa
- password : mahasiswa



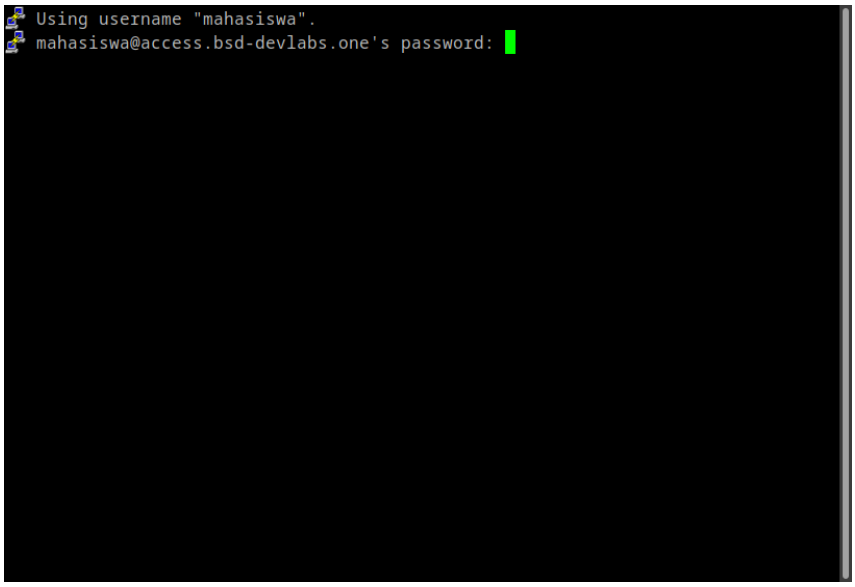
Gambar 1.2: Menggunakan PuTTY

3. PuTTY akan memberikan konfirmasi koneksi. Klik *Accept*



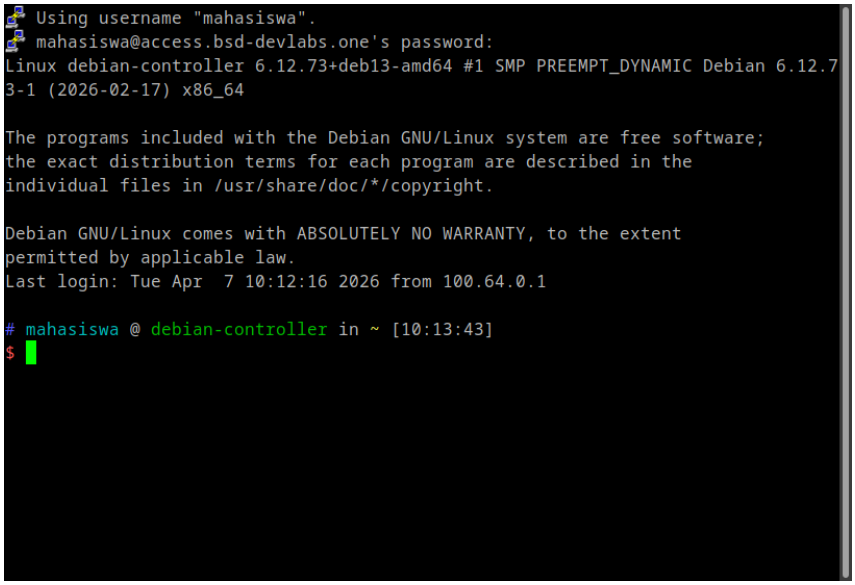
Gambar 1.3: Konfirmasi Koneksi

4. Setelah itu PuTTY akan menanyakan password. Masukkan password lalu *Enter*



Gambar 1.4: Konfirmasi Password

5. Jika berhasil akan muncul prompt shell milik akun mahasiswa

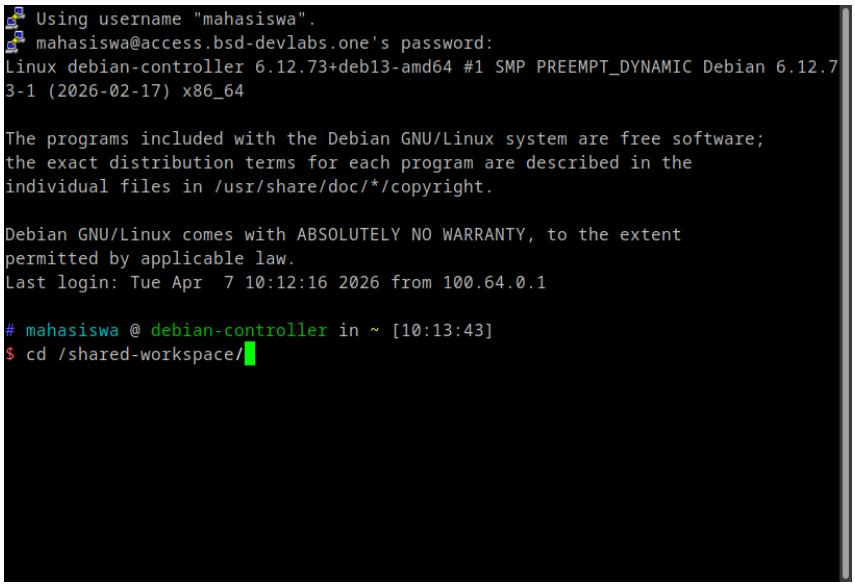


Gambar 1.5: Shell Prompt Mahasiswa

6. Workspace mahasiswa ada di folder `/shared-workspace`. Untuk berpindah ke folder ini gunakan perintah:

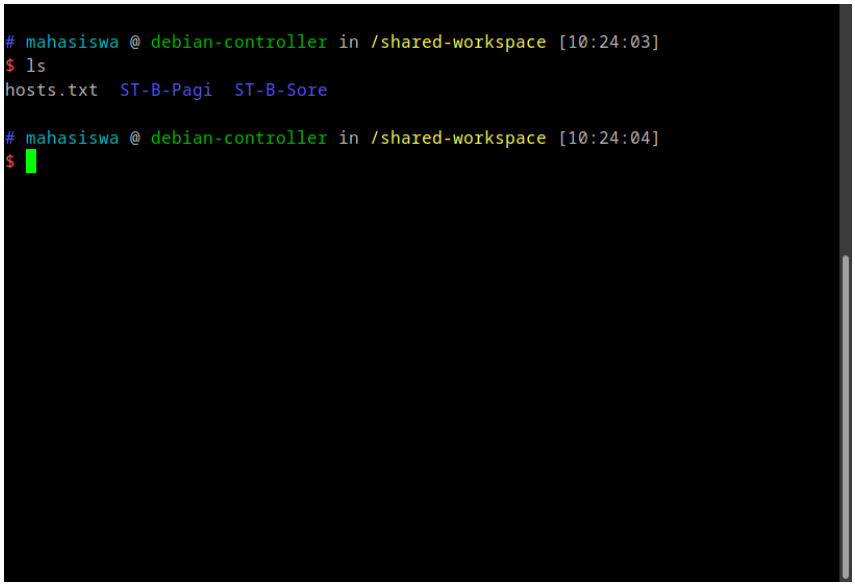
Perintah Terminal

```
cd /shared-workspace
```



Gambar 1.6: Berpindah ke Workspace Mahasiswa

7. Di folder ini ada *1 file* dan *2 folder*. Abaikan file tersebut, dan perhatikan *folder-folder* tersebut:
   - ST-B-Pagi
   - ST-B-Sore



Gambar 1.7: Konten Shared-Workspace

8. Pindah ke folder sesuai kelas masing-masing menggunakan perintah:
   - Kelas Pagi

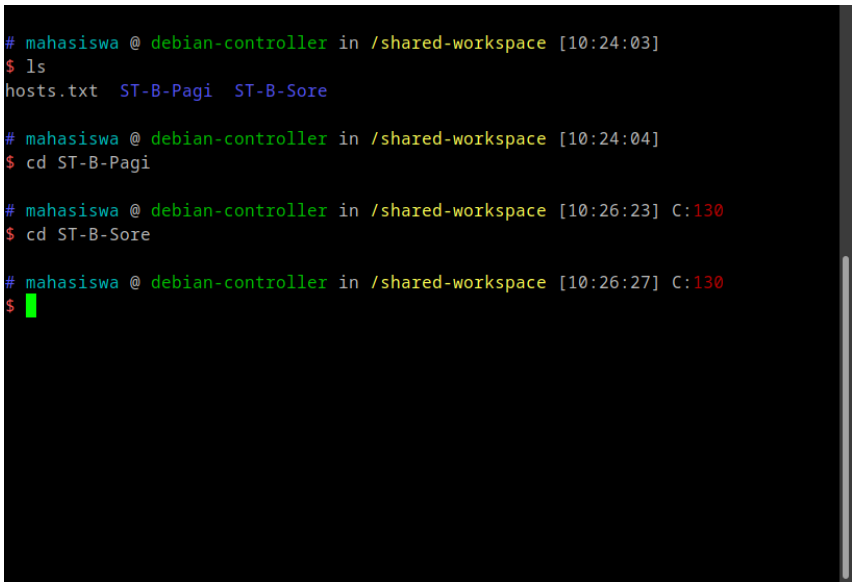
Perintah Terminal

```
cd ST-B-Pagi
```

   - Kelas Sore

Perintah Terminal

```
cd ST-B-Sore
```

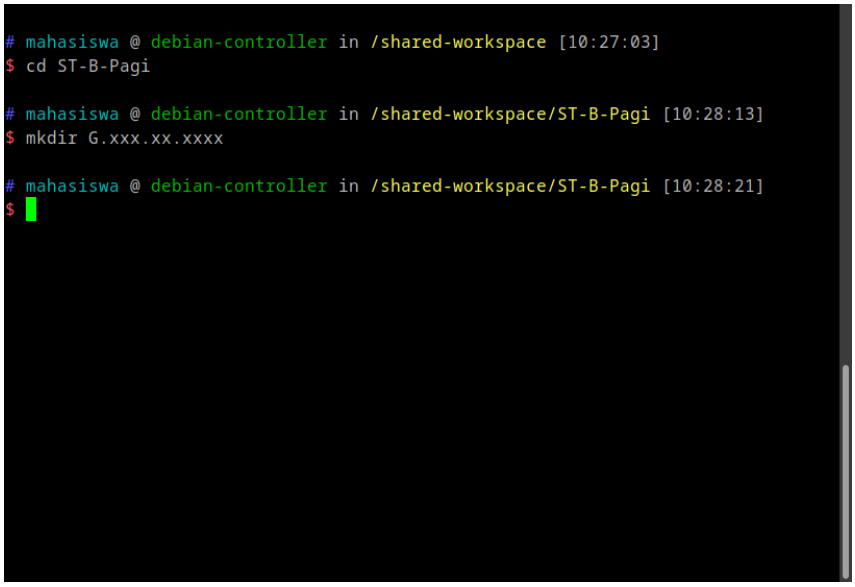


Gambar 1.8: Masuk ke Folder Kelas

9. Setelah masuk ke folder kelas. Buat satu folder dengan format `G.xxx.xx.xxxx` sesuai NIM mahasiswa masing-masing dengan perintah:

Perintah Terminal

```
mkdir G.xxx.xx.xxxx
```

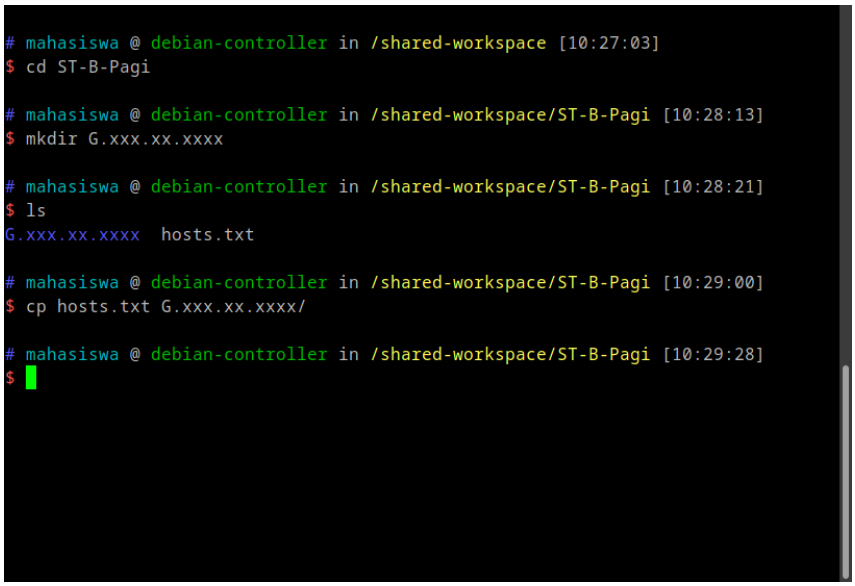


Gambar 1.9: Buat folder mahasiswa sesuai NIM

10. Kopikan file **hosts.txt** ke folder Mahasiswa dengan perintah

Perintah Terminal

```
cp hosts.txt G.xxx.xx.xxxx
```

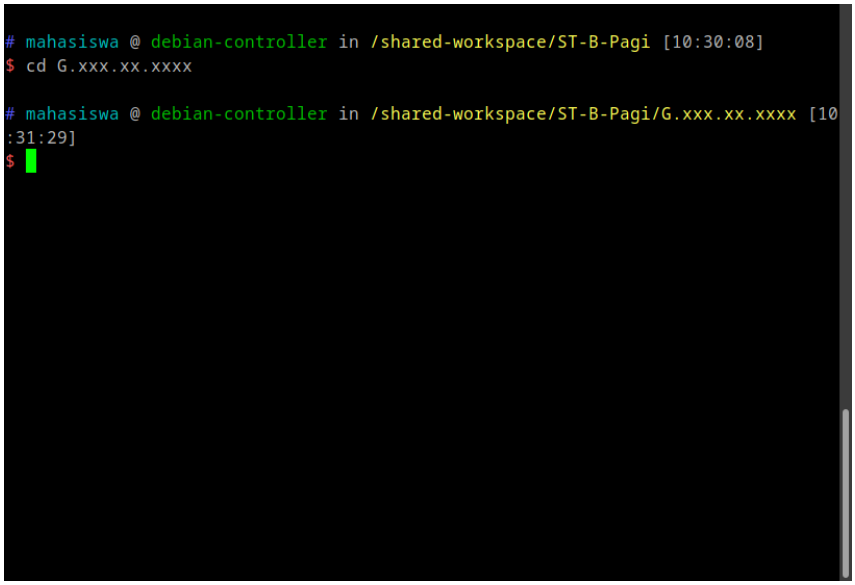


Gambar 1.10: Kopi file hosts.txt

11. Masuk ke folder mahasiswa dengan perintah

Perintah Terminal

```
cd G.xxx.xx.xxxx
```

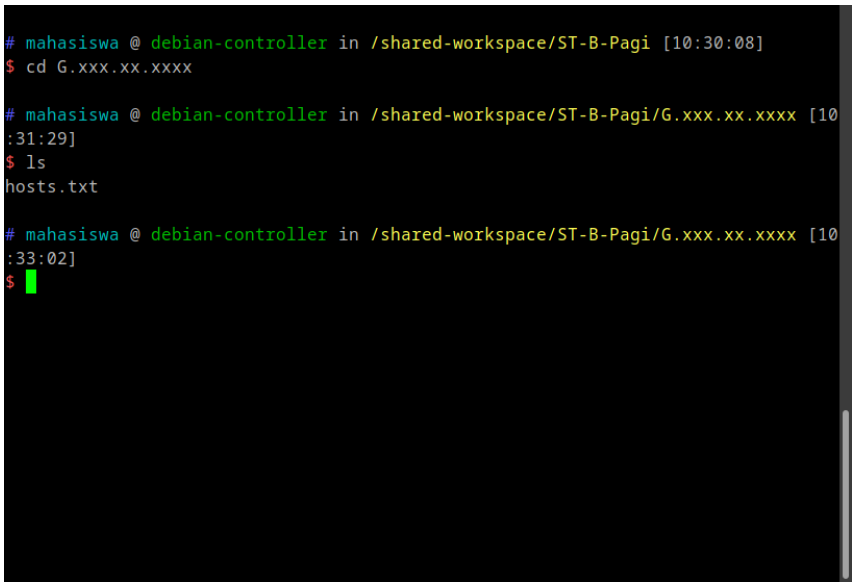


Gambar 1.11: Masuk ke folder mahasiswa

12. Tiap mahasiswa akan punya /working space/ mereka sendiri, sehingga tidak akan mengganggu mahasiswa lain. Pastikan file /hosts.txt/ ada di tiap folder mahasiswa, cek dengan perintah:

Perintah Terminal

```
ls
```



Gambar 1.12: Cek file hosts.txt

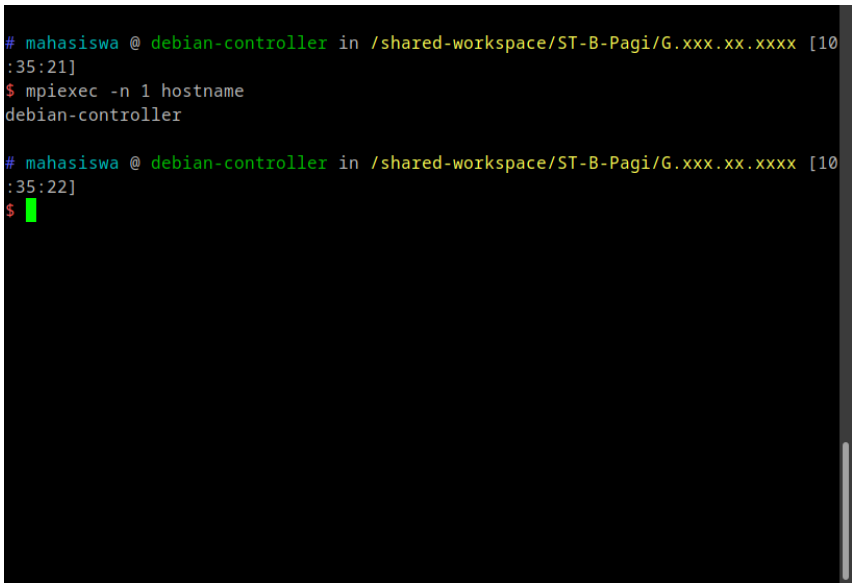
13. Untuk mengetes apakah program *OpenMPI* sudah terinstall dengan baik, gunakan perintah:

Perintah Terminal

```
mpiexec -n 1 hostname
```

Penjelasan

Penjelasan : Jalankan 1 proses dari program hostname



Gambar 1.13: Cek OpenMPI Controller

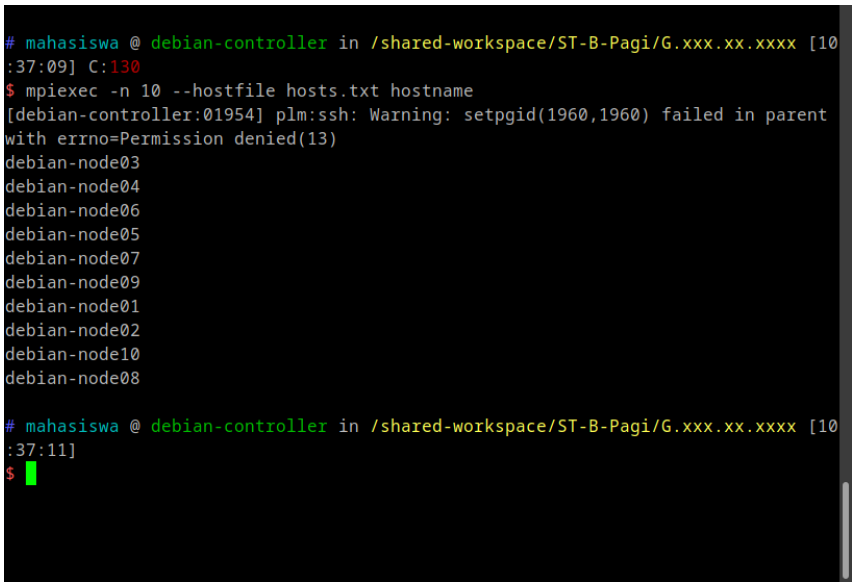
14. Jika `mpiexec` mengembalikan `debian-controller`, maka program bekerja dengan baik untuk lokal. Untuk mengetes semua *worker node* yang ada, gunakan perintah berikut:

Perintah Terminal

```
mpiexec -n 10 --hostfile hosts.txt hostname
```

Penjelasan

Penjelasan : Jalankan 10 proses dari hostname menggunakan node dari hosts.txt



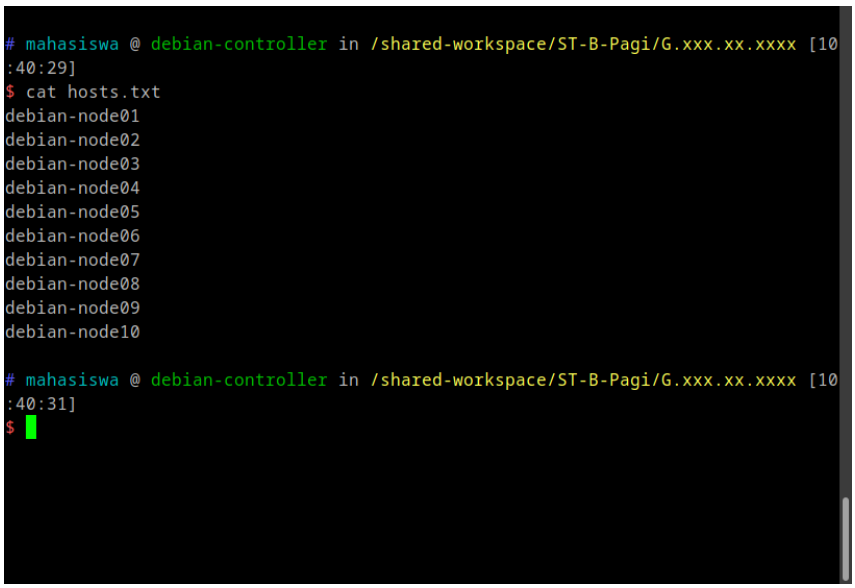
Gambar 1.14: Cek OpenMPI Worker

15. Lama proses tergantung dari sibuk tidaknya masing-masing *worker*, jadi jika lama mohon ditunggu.

16. Hasil 10 hostname yang muncul merupakan refleksi dari jumlah worker yang ada di *hosts.txt*. Untuk melihat daftar *worker node* yang terdaftar, gunakan perintah

Perintah Terminal

```
cat hosts.txt
```

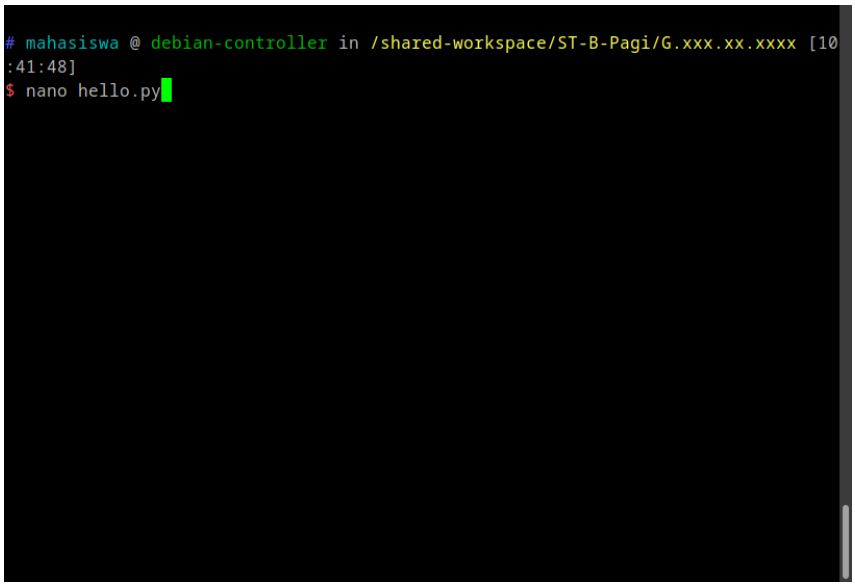


Gambar 1.15: Melihat isi hosts.txt

17. Untuk memahami cara kerja OpenMPI dalam mengirimkan tugas ke masing-masing worker. Buat skrip Python dengan perintah:

Perintah Terminal

```
nano hello.py
```

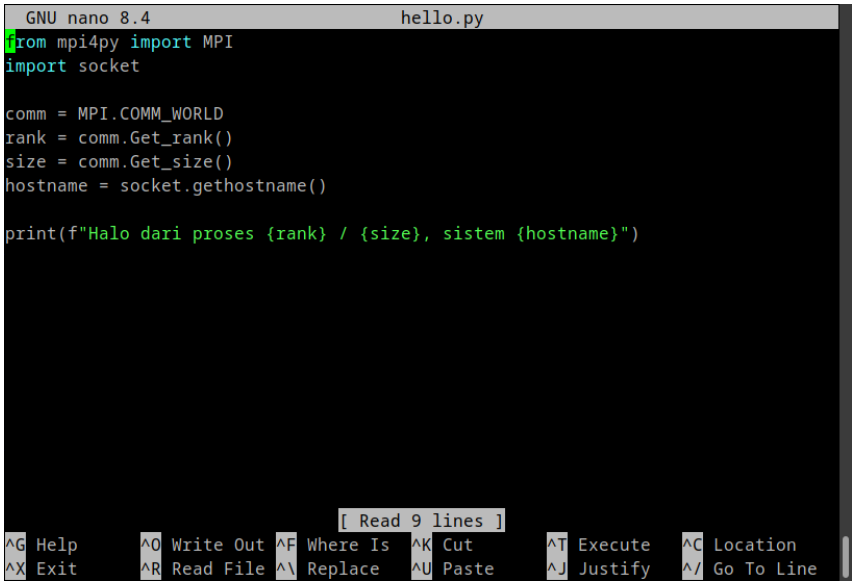


Gambar 1.16: Membuat file hello.py

18. Teks editor akan muncul, lalu masukkan kode berikut:

Perintah Terminal

```
from mpi4py import MPI
import socket

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()
hostname = socket.gethostname()

print(f"Halo dari proses { rank+1 } / { size }, sistem { hostname }")
```



Gambar 1.17: Isi hello.py

19. Untuk menyimpan dan keluar gunakan kombinasi:

(a) `Ctrl + O` dan `Enter` : untuk menyimpan

(b) `Ctrl + X` : untuk keluar setelah menyimpan

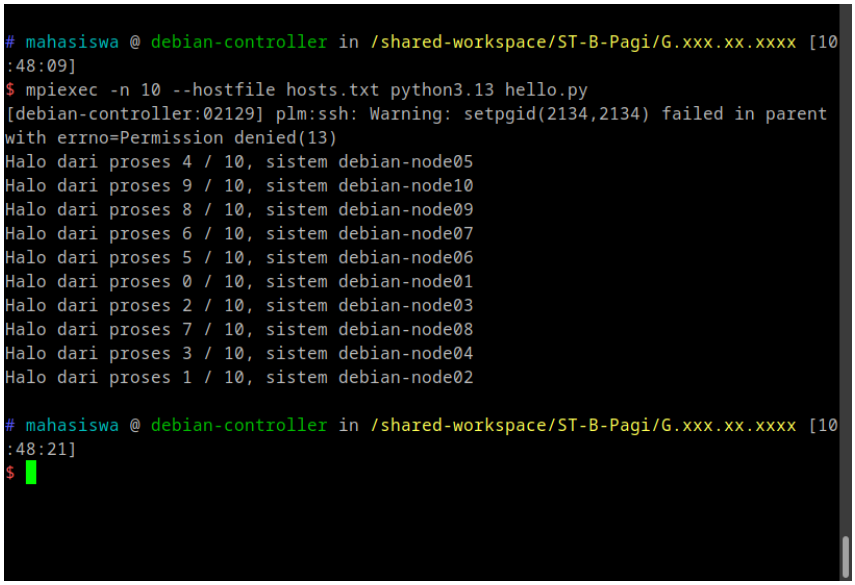
20. Jalankan kode tersebut menggunakan perintah berikut:

Perintah Terminal

```
mpiexec -n 10 --hostfile hosts.txt python3.13 hello.py
```

Penjelasan

Penjelasan : Jalankan 10 proses Python 3.13 untuk menjalankan skrip hello.py



Gambar 1.18: Menjalankan 10 Proses hello.py

21. Ciri khas utama dari sistem terdistribusi adalah: kita tidak bisa mengatur siapa yang duluan selesai secara eksplisit.

22. Screenshot hasil ini dan kirimkan ke e-Learning

Bab 2

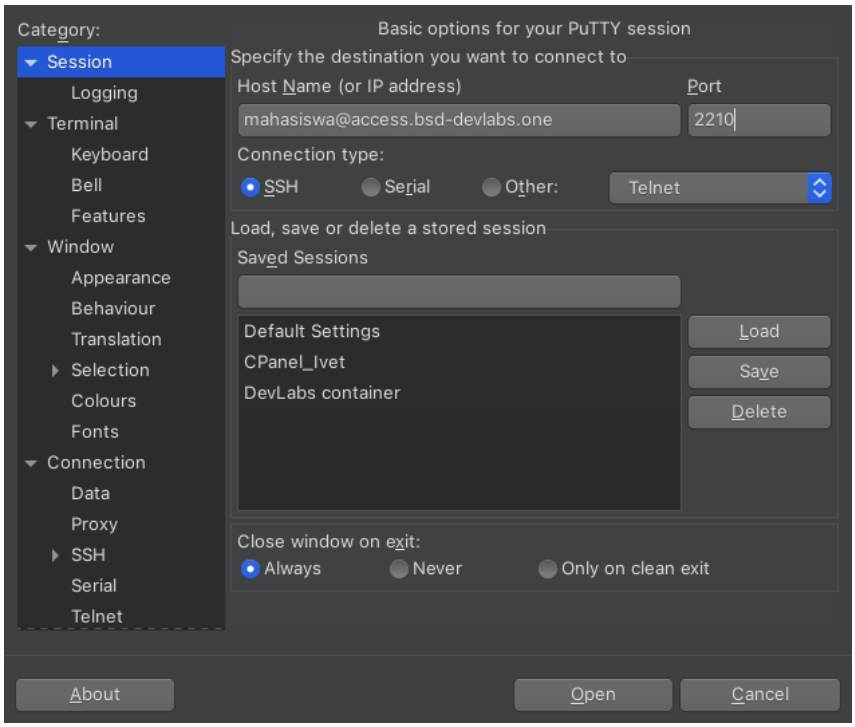
Komunikasi Dasar MPI

2.1 Pendahuluan

Komunikasi dasar dalam MPI (Message Passing Interface) merupakan komponen inti dalam komputasi paralel berbasis sistem terdistribusi, yang memungkinkan pertukaran data antar proses yang berjalan pada node berbeda dalam sebuah cluster. Melalui implementasi menggunakan OpenMPI dan binding Python mpi4py, konsep komunikasi dapat dipelajari secara praktis melalui dua pendekatan utama, yaitu komunikasi point-to-point seperti send dan recv, serta komunikasi kolektif seperti broadcast, scatter, dan gather. Pemahaman terhadap mekanisme ini sangat penting untuk mengoptimalkan distribusi beban kerja, sinkronisasi proses, serta efisiensi eksekusi program paralel. Oleh karena itu, praktikum ini dirancang untuk memberikan pengalaman langsung dalam mengimplementasikan berbagai pola komunikasi dasar MPI pada lingkungan cluster yang telah dikonfigurasi, sehingga mendukung pengembangan aplikasi komputasi paralel yang lebih kompleks dan efisien.

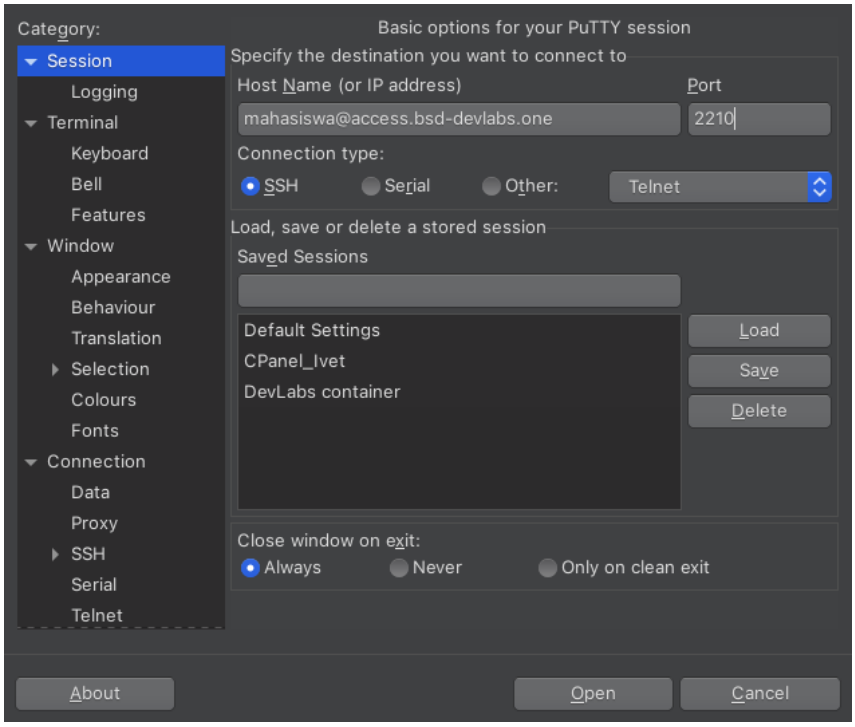
2.2 Tutorial

1. Jalankan aplikasi PuTTY



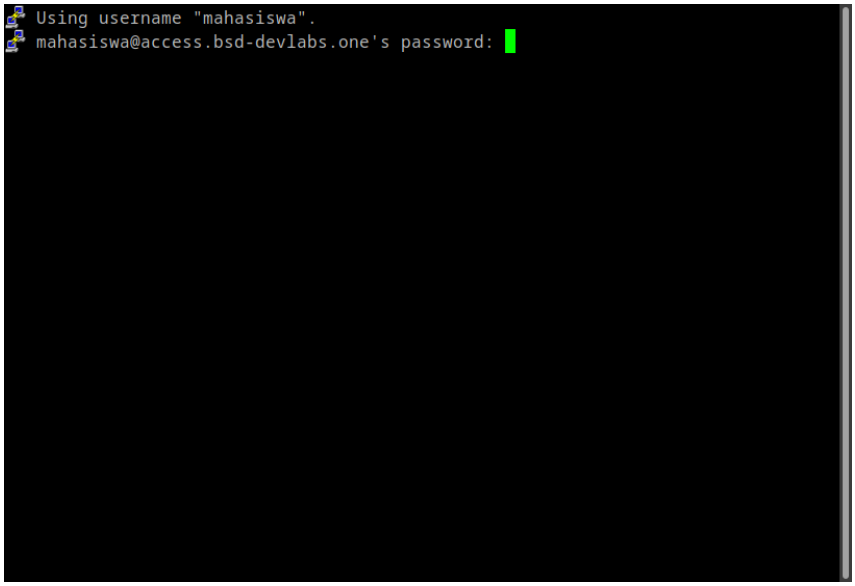
Gambar 2.1: Jalankan PuTTY

2. Masukkan *hostname*, *username* dan *port*
   - hostname : access.bsd-devlabs.one
   - username : mahasiswa
   - password : mahasiswa
   - port : 2210



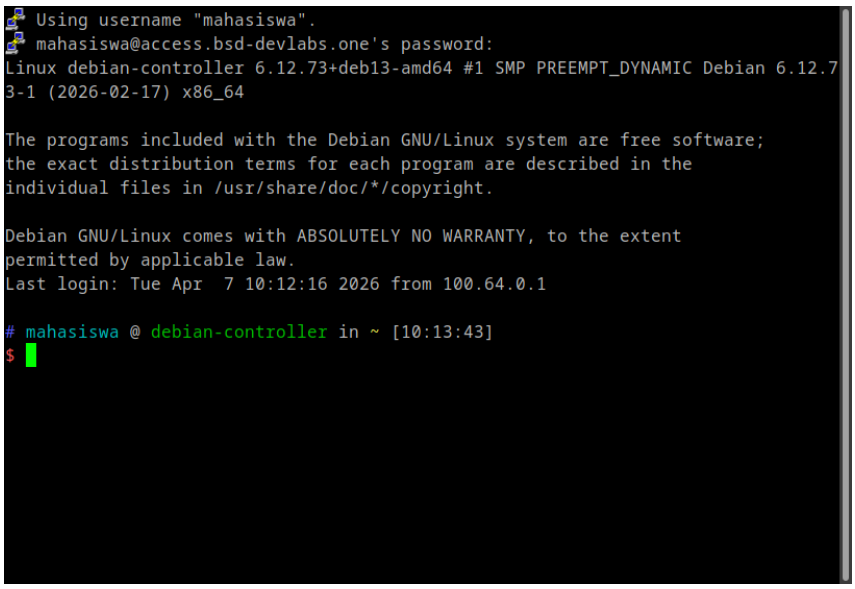
Gambar 2.2: Masukkan Akses

3. Masukkan *Password* ketika diminta



Gambar 2.3: Masukkan Password

4. Jika berhasil, maka PuTTY akan menampilkan prompt



Gambar 2.4: Prompt Node Kontroler

5. Masuk ke folder mahasiswa masing-masing dengan perintah:

   - Kelas Pagi

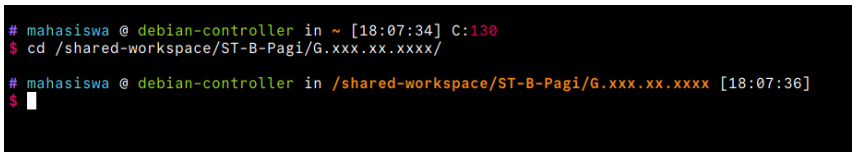
     Perintah Terminal

     ```
     cd /shared-workspace/ST-B-Pagi/G.xxx.xx.xxxx/
     ```

   - Kelas Sore

     Perintah Terminal

     ```
     cd /shared-workspace/ST-B-Sore/G.xxx.xx.xxxx/
     ```



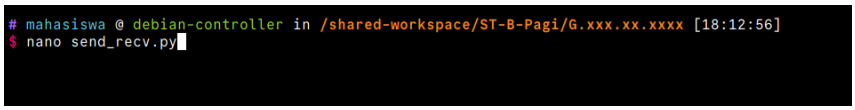
Gambar 2.5: Berpindah ke folder mahasiswa

2.2.1 Komunikasi Point-to-Point

1. Setelah masuk ke folder masing-masing mahasiswa, buat sebuah file dengan nama **send_recv.py** dengan perintah:

Perintah Terminal

```
nano send_recv.py
```



Gambar 2.6: Membuat file send_recv.py

2. Di dalam **editor nano**, masukkan kode berikut:

Perintah Terminal

```
from mpi4py import MPI
from socket import gethostname

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
hostname = gethostname()

if rank == 0:
    data = "FTIK-USM"
    comm.send(data, dest=1)
    print(f"{ hostname }-Proses { rank } mengirimkan : { data }")
elif rank == 1:
    data = comm.recv(source=0)
    print(f"{ hostname }-Proses { rank } menerima : { data }")
else:
    pass
```

Penjelasan

```
Penjelasan : Rank / Proses 0 Kirim Data ke Rank / Proses 1.
Proses lain mengabaikan
```

```
 GNU nano 8.4
from mpi4py import MPI
from socket import gethostname

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
hostname = gethostname()

if rank == 0:
    data = "FTIK-USM"
    comm.send(data, dest=1)
    print(f"{hostname}-Proses {rank} mengirimkan Data")
elif rank == 1:
    data = comm.recv(source=0)
    print(f"{hostname}-Proses {rank} menerima : {data}")
else:
    pass
```

Gambar 2.7: Isi file send_recv.py

3. Jalankan script dengan perintah

Perintah Terminal

```
mpiexec -n 10 --hostfile hosts.txt python3.13 send_recv.py
```

```
# mahasiswa @ debian-controller in /shared-workspace/ST-B-Pagi/G.xxx.xx.xxxx [18:24:18] C:130
$ mpiexec -n 10 --hostfile hosts.txt python3.13 send_recv.py
```

Gambar 2.8: Jalankan Script send_recv.py

4. Tekan **Enter** untuk melihat **Rank / Proses 0** mengirimkan data ke **Rank / Proses 1**

```
# mahasiswa @ debian-controller in /shared-workspace/ST-B-Pagi/G.xxx.xx.xxxx [18:27:08]
$ mpiexec -n 10 --hostfile hosts.txt python3.13 send_recv.py
debian-node02-Proses 1 menerima : FTIK-USM
debian-node01-Proses 0 mengirimkan : FTIK-USM

# mahasiswa @ debian-controller in /shared-workspace/ST-B-Pagi/G.xxx.xx.xxxx [18:27:11]
$
```

Gambar 2.9: Hasil Script send_recv.py

2.2.2 Komunikasi Broadcast

1. Berikutnya adalah membuat komunikasi broadcast dengan mengirimkan data ke banyak node sekaligus. Node dengan **Rank 0** yang akan melakukan broadcast
2. Pastikan PuTTY sedang menampilkan **Prompt Shell** dan masuk ke dalam folder masing-masing

```
# mahasiswa @ debian-controller in /shared-workspace/ST-B-Pagi/G.xxx.xx.xxxx [3:24:48]
$
```

Gambar 2.10: Kondisi Prompt Shell

3. Buat file dengan nama **broadcast.py** dengan perintah

Perintah Terminal

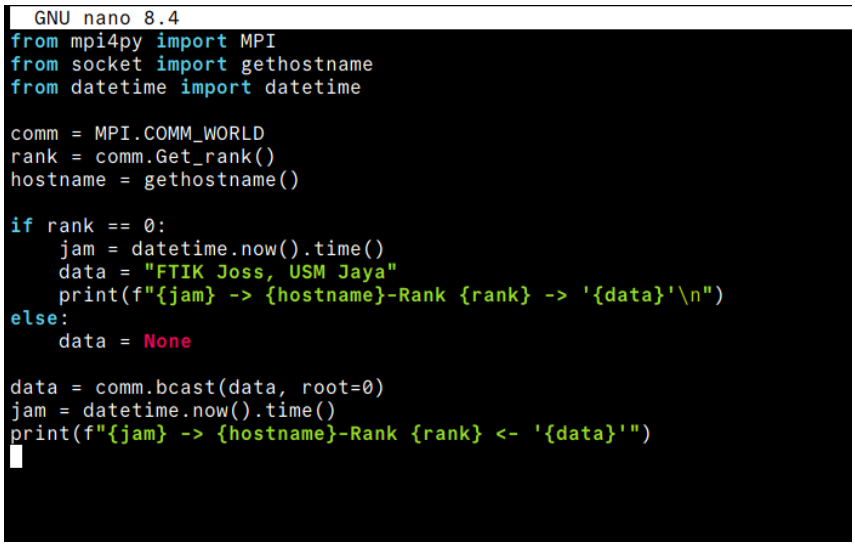
```
nano broadcast.py
```

Gambar 2.11: Membuat broadcast.py

4. Di dalam file tersebut, masukkan kode berikut ini:

Perintah Terminal

```
from mpi4py import MPI
from socket import gethostname
from datetime import datetime

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
hostname = gethostname()

if rank == 0:
    jam = datetime.now().time()
    data = "FTIK Joss, USM Jaya"
    print(f"{ jam } -> { hostname }-Rank { rank } -> '{ data }' \n")
else:
    data = None

data = comm.bcast(data, root=0)
jam = datetime.now().time()
print(f"{ jam } -> { hostname }-Rank { rank } <- '{ data }'")
```

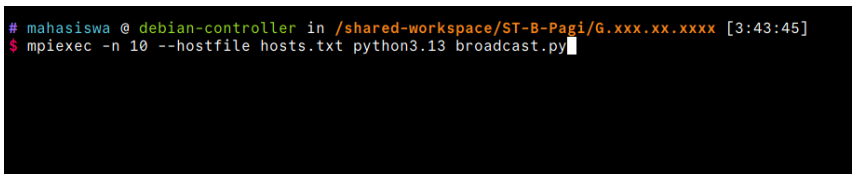


Gambar 2.12: Isi broadcast.py

5. Jalankan script dengan perintah

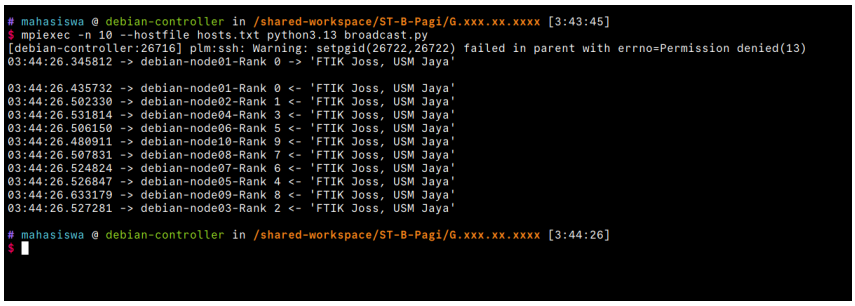
Perintah Terminal

```
mpiexec -n 10 --hostfile hosts.txt python3.13 broadcast.py
```



Gambar 2.13: Jalankan Script broadcast.py

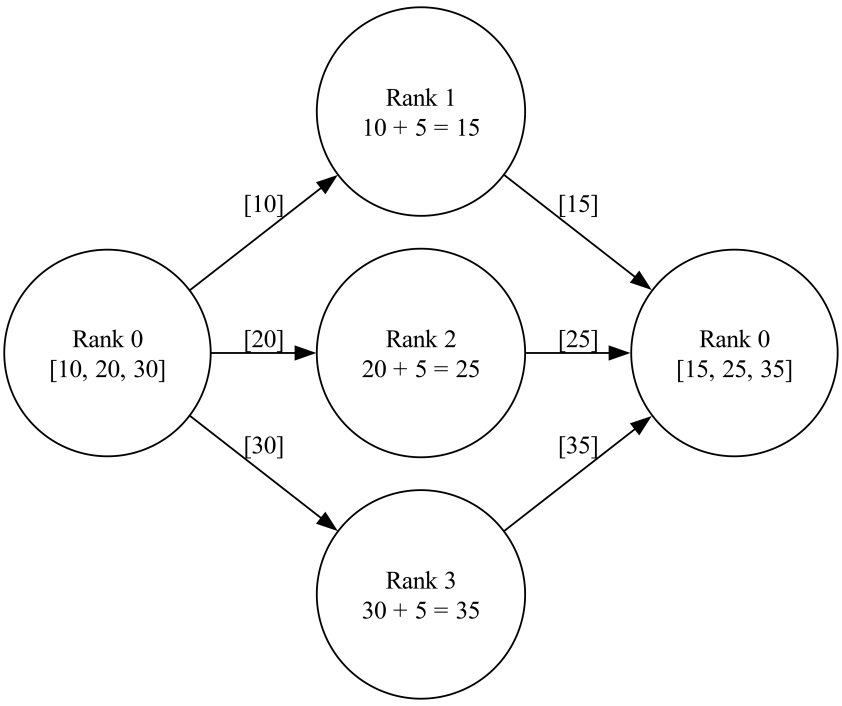
6. Tekan **Enter** untuk melihat **Rank** / **Proses 0** mengirimkan broadcast data ke semua **Rank** / **Proses**



Gambar 2.14: Hasil Script broadcast.py

2.2.3 Scatter dan Gather

1. Setelah **broadcast** adalah **Scatter** dan **Gather**. Digunakan untuk menyebarkan data, lakukan kalkulasi, kemudian dikumpulkan kembali ke node asal. Perhatikan ilustrasi berikut:



Gambar 2.15: Prosedur Scatter dan Gather

2. Pastikan PuTTY sedang menampilkan **Prompt Shell** dan masuk ke dalam folder masing-masing

Gambar 2.16: Kondisi Prompt Shell

3. Buat file dengan nama scatter_gatter.py dengan perintah

Perintah Terminal

```
nano scatter_gather.py
```

Gambar 2.17: Buat File scatter_gather.py

4. Masukkan kode berikut ke dalam file scatter_gather.py

Perintah Terminal

```
from mpi4py import MPI
from socket import gethostname
from datetime import datetime
from random import random

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
hostname = gethostname()

if rank == 0:
    data = [int(random()*10) for _ in range(10)]
    jam = datetime.now().time()
    print(f"\n{ jam } -> { hostname }-{ rank } -> '{ data }'\n")
else:
    data = None

data = comm.scatter(data, root=0)
jam = datetime.now().time()
print(f"{ jam } => { hostname }-{ rank } <- '{ data }'\n")

result = data * 2
gathered = comm.gather(result, root=0)

if rank == 0:
    jam = datetime.now().time()
    print(f"\n{ jam } -> { hostname }-{ rank } -> '{ data }'")
```

```
 GNU nano 8.4
from mpi4py import MPI
from socket import gethostname
from datetime import datetime
from random import random


comm = MPI.COMM_WORLD
rank = comm.Get_rank()
hostname = gethostname()


if rank == 0:
    data = [int(random()*10) for _ in range(10)]
    jam = datetime.now().time()
    print(f"\n{jam} -> {hostname}-{rank} -> '{data}'\n")
else:
    data = None


data = comm.scatter(data, root=0)
jam = datetime.now().time()
print(f"{jam} => {hostname}-{rank} <- '{data}'")

result = data * 2
gathered = comm.gather(result, root=0)

if rank == 0:
    jam = datetime.now().time()
    print(f"\n{jam} -> {hostname}-{rank} <- '{gathered}'")
```

Gambar 2.18: Isi scatter_gather.py

5. Jalankan script dengan perintah

Perintah Terminal

```
mpiexec -n 10 --hostfile hosts.txt python3.13 scatter_gather.py
```

```
# mahasiswa @ debian-controller in /shared-workspace/ST-B-Pagi/G.xxx.xx.xxxx [4:12:17]
$ mpiexec -n 10 --hostfile hosts.txt python3.13 scatter_gather.py
```

Gambar 2.19: Jalankan Script scatter_gather.py

6. Tekan **Enter** untuk melihat **Rank** / **Proses 0** mengirimkan broadcast data ke semua **Rank** / **Proses** dan dikumpulkan secara berurutan kembali

```
# mahasiswa @ debian-controller in /shared-workspace/ST-B-Pagi/G.xxx.xx.xxxx [4:12:17]
$ mpiexec -n 10 --hostfile hosts.txt python3.13 scatter_gather.py
[debian-controller:27340] plm:ssh: Warning: setpgid(27345,27345) failed in parent with errno=Permission denied(13)

04:12:33.078718 -> debian-node01-0 -> '[7, 9, 0, 1, 9, 2, 3, 5, 8, 4]'

04:12:33.662404 => debian-node03-2 <- '0'
04:12:33.701850 => debian-node02-1 <- '9'
04:12:33.851452 => debian-node04-3 <- '1'
04:12:33.827606 => debian-node05-4 <- '9'
04:12:33.841744 => debian-node06-5 <- '2'
04:12:33.844932 => debian-node07-6 <- '3'
04:12:33.848948 => debian-node08-7 <- '5'
04:12:33.852080 => debian-node01-0 <- '7'
04:12:33.792019 => debian-node10-9 <- '4'
04:12:33.918595 => debian-node09-8 <- '8'

04:12:33.853492 -> debian-node01-0 <- '[14, 18, 0, 2, 18, 4, 6, 10, 16, 8]'
```

Gambar 2.20: Hasil Script scatter_gather.py

7. Screenshot hasil dan kirimkan ke e-Learning

Bab 3

Sinkronisasi dan Deadlock

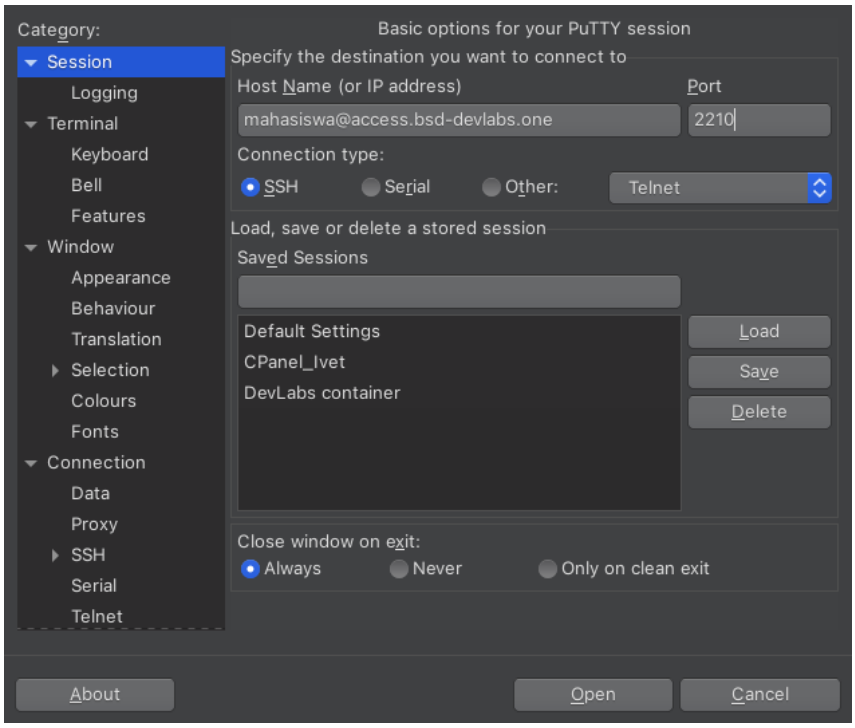
3.1 Pendahuluan

Praktikum ini membahas konsep fundamental dalam sistem terdistribusi, yaitu sinkronisasi proses dan kondisi deadlock pada komunikasi antar node menggunakan mpi4py di atas platform Open MPI. Dalam lingkungan komputasi paralel, koordinasi antar proses menjadi aspek krusial untuk memastikan eksekusi berjalan konsisten dan efisien. Namun, kesalahan dalam pengaturan komunikasi, khususnya pada penggunaan mekanisme blocking, dapat menyebabkan deadlock yang menghambat seluruh sistem. Melalui serangkaian eksperimen, praktikum ini mengarahkan analisis terhadap penyebab deadlock serta implementasi solusi melalui teknik sinkronisasi dan komunikasi non-blocking, sehingga memberikan pemahaman praktis mengenai pengelolaan komunikasi yang aman dan optimal dalam sistem terdistribusi.

3.2 Tutorial

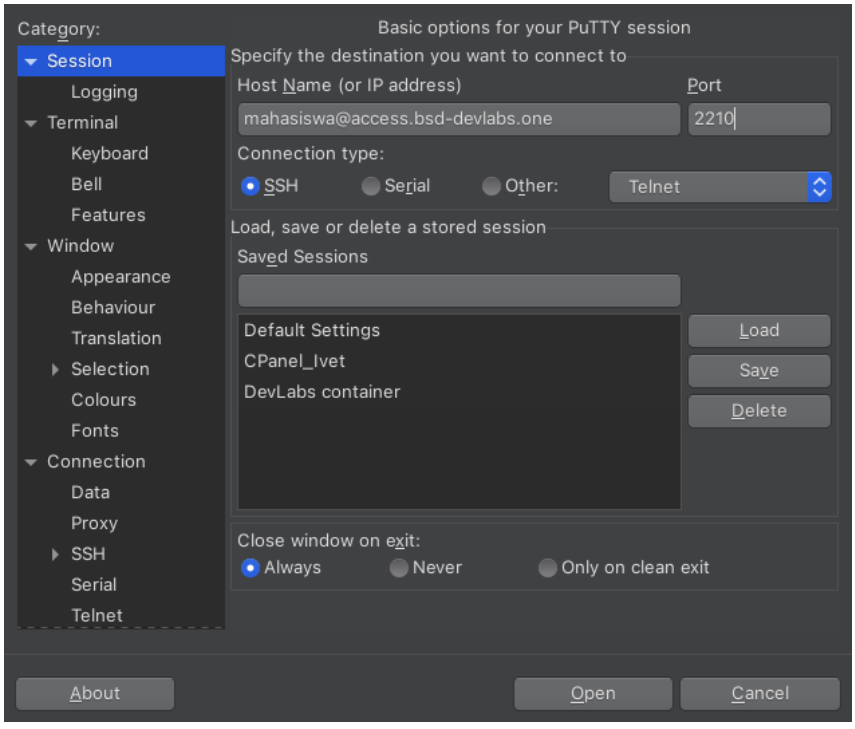
3.2.1 Akses SSH

1. Jalankan aplikasi PuTTY



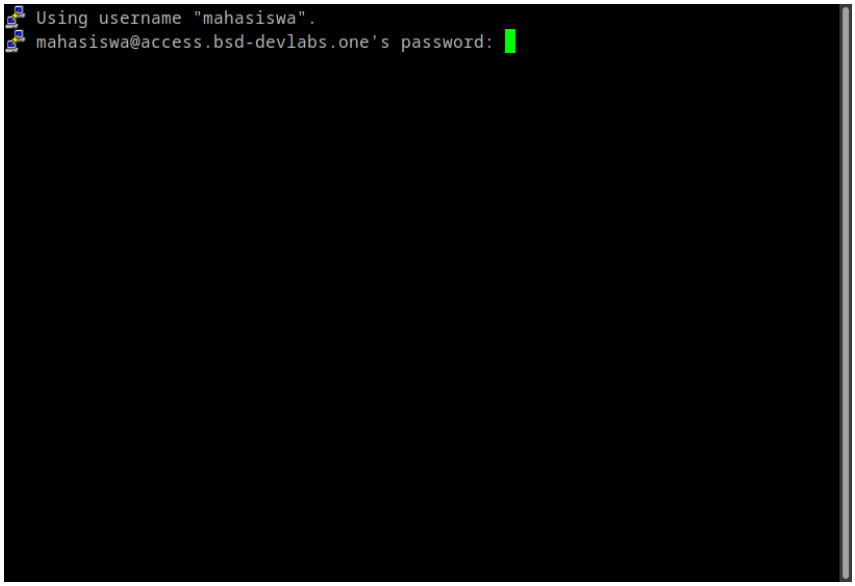
Gambar 3.1: Jalankan PuTTY

2. Masukkan *hostname*, *username* dan *port*
   - hostname : access.bsd-devlabs.one
   - username : mahasiswa
   - password : mahasiswa
   - port : 2210



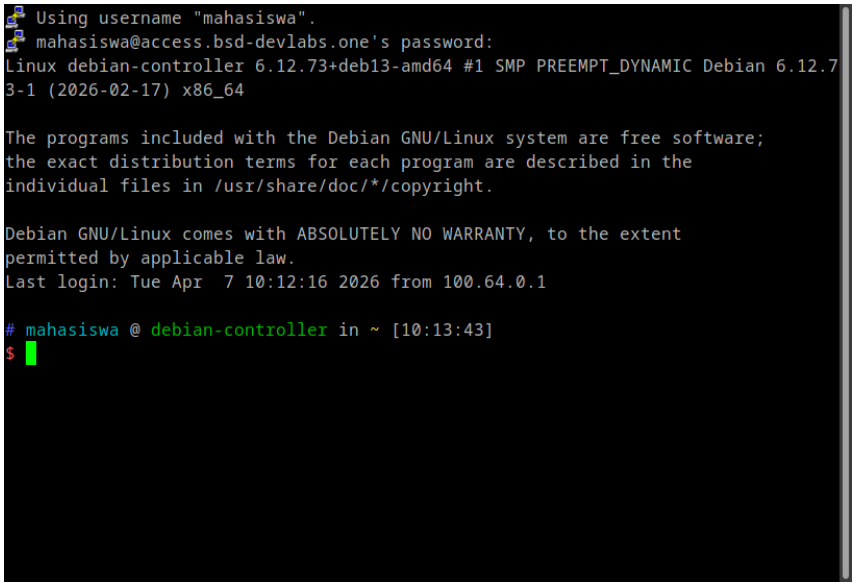
Gambar 3.2: Masukkan Akses

3. Masukkan *Password* ketika diminta



Gambar 3.3: Masukkan Password

4. Jika berhasil, maka PuTTY akan menampilkan prompt



Gambar 3.4: Prompt Node Kontroler

5. Masuk ke folder mahasiswa masing-masing dengan perintah:

   - Kelas Pagi

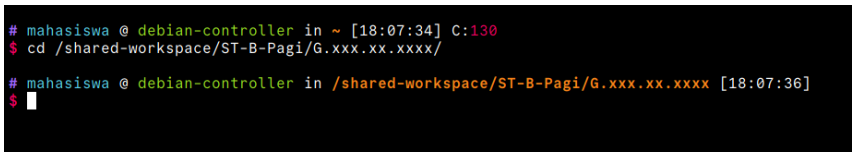
     Perintah Terminal

     ```
     cd /shared-workspace/ST-B-Pagi/G.xxx.xx.xxxx/
     ```

   - Kelas Sore

     Perintah Terminal

     ```
     cd /shared-workspace/ST-B-Sore/G.xxx.xx.xxxx/
     ```



Gambar 3.5: Berpindah ke folder mahasiswa

3.2.2 Deadlock dengan Blocking Communication

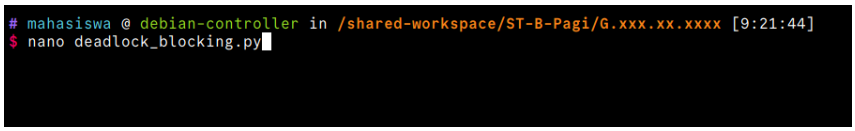
1. Pastikan mahasiswa sudah berada di folder masing-masing

Gambar 3.6: Folder Aktif Mahasiswa

2. Buat sebuah file dengan nama **deadlock_blocking.py** dengan menggunakan perintah

Perintah Terminal

```
nano deadlock_blocking.py
```

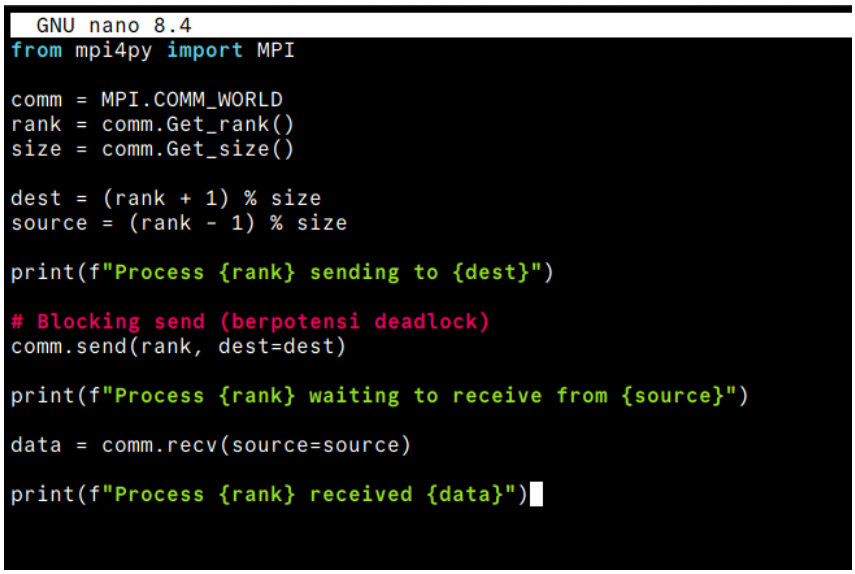


Gambar 3.7: Membuat file deadlock_blocking.py

3. Masukkan kode berikut yang berfungsi untuk mengirimkan data ke **node** lain namun bersifat **blocking**.

Perintah Terminal

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

dest = (rank + 1) % size
source = (rank - 1) % size

print(f"Process { rank } sending to { dest }")

# Blocking send (berpotensi deadlock)
comm.send(rank, dest=dest)

print(f"Process { rank } waiting to receive from { source }")

data = comm.recv(source=source)

print(f"Process { rank } received { data }")
```

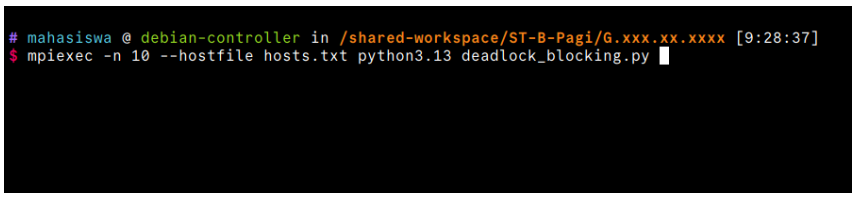


Gambar 3.8: Kode Deadlock Blocking

4. Simpan dengan kombinasi:
   - **CTRL+o** lalu **Enter**
   - Keluar dengan **CTRL+x**
5. Jalankan dengan perintah

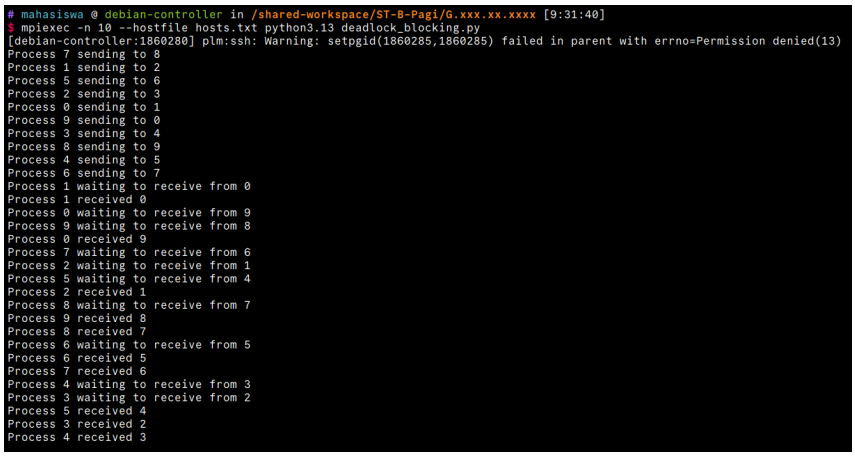
Perintah Terminal

```
mpiexec -n 10 --hostfile hosts.txt python3.13 deadlock_blocking.py
```



Gambar 3.9: Menjalankan Kode Deadlock Blocking

6. Ketika dijalankan, kode tersebut memiliki **Kemungkinan** akan berhenti di tengah jalan. Namun karena **Sistem Terdistribusi** bekerja dengan ketelitian hingga milidetik, sehingga kemungkinan deadlock akan tipis.



Gambar 3.10: Hasil Deadlock Blocking

3.2.3 Sinkronisasi dengan Barrier

1. Berikutnya adalah memahami sinkronisasi. Normalnya proses yang berjalan ketika sudah selesai akan segera menjalankan proses berikutnya.
2. Namun dengan menggunakan **barrier**, proses-proses yang seharusnya sudah selesai ditahan hingga semua proses selesai.
3. Secara sederhana, **Sinkronisasi** membuat proses mulai bersama dan berakhir bersama
4. Buat file dengan nama **sync.py** dengan menggunakan perintah

Perintah Terminal

```
nano sync.py
```

Gambar 3.11: Membuat file sync.py

5. Masukkan kode berikut:

Perintah Terminal

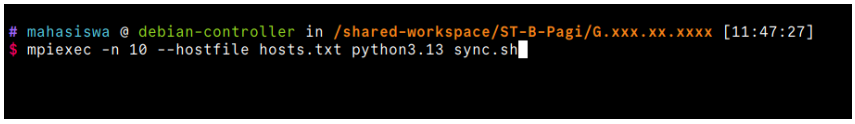
```
from mpi4py import MPI
import time

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

print(f"Process { rank } before barrier - { time.time() }")
time.sleep(rank)  # simulasi proses berbeda

comm.Barrier()

print(f"Process { rank } after barrier - { time.time() }")
```

Gambar 3.12: Membuat file sync.py

6. Simpan dengan kombinasi:
   - **CTRL+o** lalu **Enter**
   - Keluar dengan **CTRL+x**
7. Jalankan kode dengan perintah

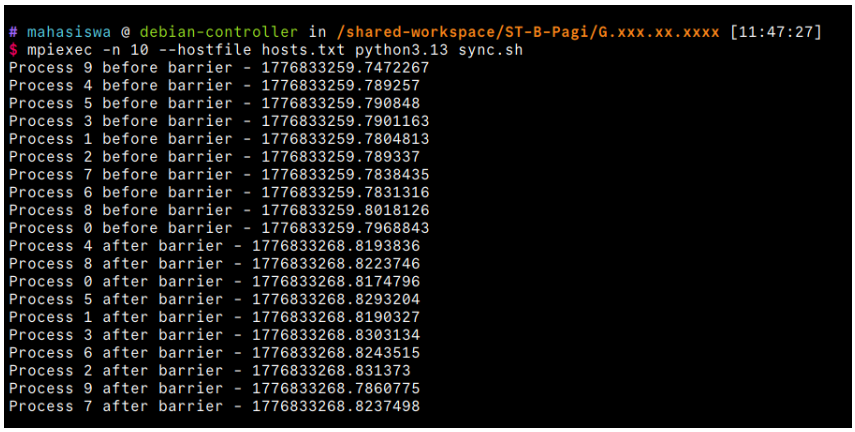
Perintah Terminal

```
mpiexec -n 10 --hostfile hosts.txt python3.13 sync.py
```



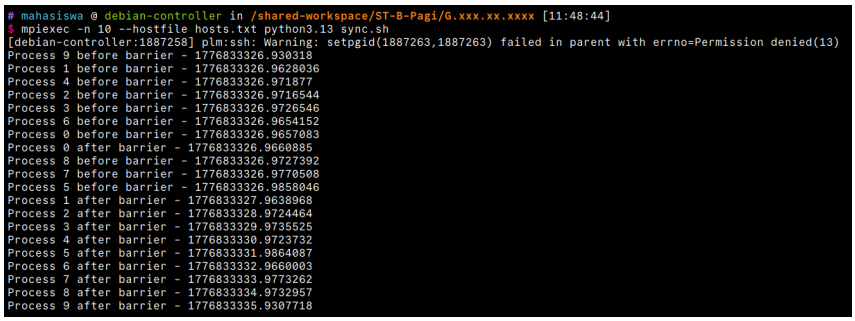
Gambar 3.13: Menjalankan Kode Sync

8. Hasil kode, proses akan dijalankan bersama-sama, dan berakhir bersama-sama



Gambar 3.14: Menjalankan Kode Sync

9. Bandingkan dengan tanpa perintah **barrier** yang di mana proses ada yang sudah selesai sebelum proses lain selesai, seperti berikut:



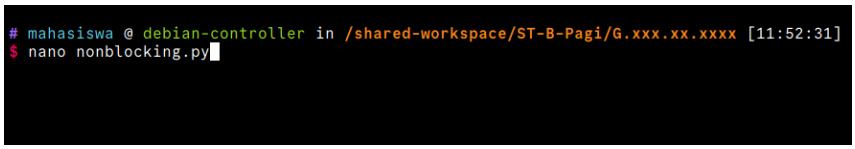
Gambar 3.15: Tanpa Barrier

3.2.4 Solusi Deadlock dengan Non-Blocking

1. Berikutnya adalah solusi untuk deadlock, metode ini sangat direkomendasikan jika **blocking** terjadi.
2. Buat file dengan nama **nonblocking.py** dengan menggunakan perintah

Perintah Terminal

```
nano nonblocking.py
```

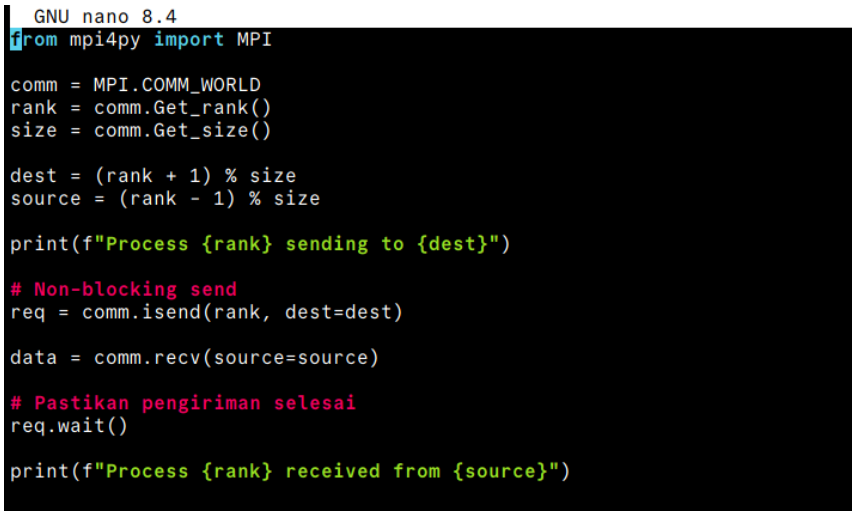


Gambar 3.16: Membuat file nonblocking.py

3. Isi file tersebut dengan kode berikut

Perintah Terminal

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

dest = (rank + 1) % size
source = (rank - 1) % size

print(f"Process {rank} sending to {dest}")

# Non-blocking send
req = comm.isend(rank, dest=dest)

data = comm.recv(source=source)

# Pastikan pengiriman selesai
req.wait()

print(f"Process {rank} received from {source}")
```

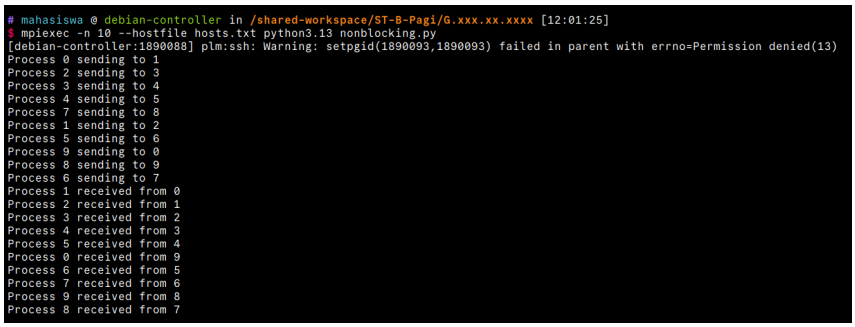


Gambar 3.17: Isi file nonblocking.py

4. Simpan dengan kombinasi:
   - **CTRL+o** lalu **Enter**
   - Keluar dengan **CTRL+x**
5. Jalankan kode dengan perintah dan lihat pola nya.

Perintah Terminal

```
mpiexec -n 10 --hostfile hosts.txt python3.13 nonblocking.py
```



Gambar 3.18: Menjalankan Kode Nonblocking

Bab 4

Logical Clock (Lamport)

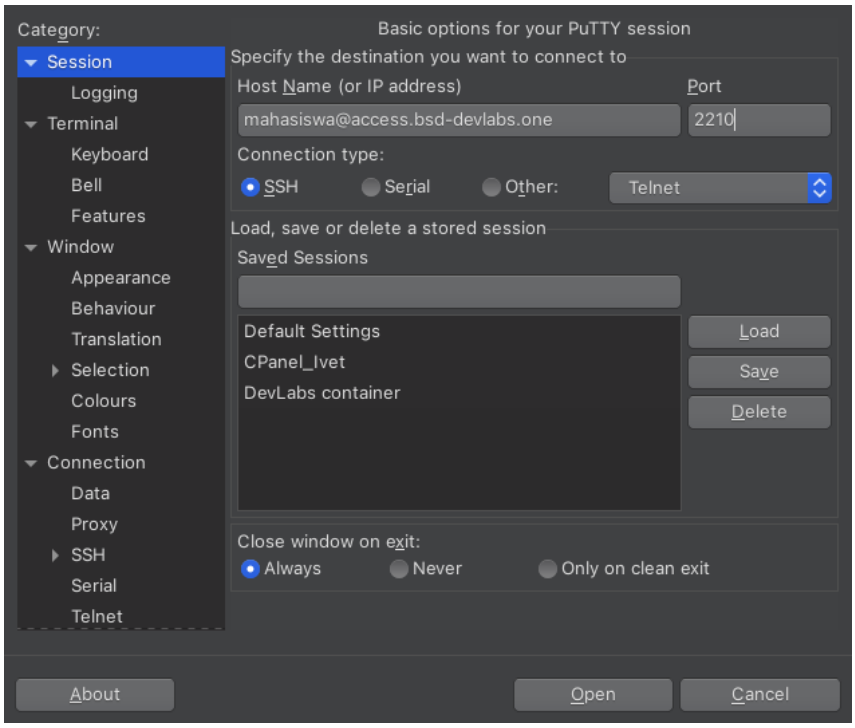
4.1 Pendahuluan

Praktikum ini bertujuan untuk memahami mekanisme pengurutan kejadian dalam sistem terdistribusi menggunakan konsep logical clock yang diperkenalkan oleh Leslie Lamport. Dalam lingkungan terdistribusi, tidak tersedia acuan waktu global yang konsisten, sehingga diperlukan pendekatan logis untuk menentukan hubungan sebab-akibat (happened-before) antar event. Melalui implementasi menggunakan OpenMPI dan mpi4py, praktikum ini memfokuskan pada pembuatan event lokal, pengiriman timestamp antar proses, serta sinkronisasi clock logis berdasarkan aturan Lamport. Dengan demikian, mahasiswa diharapkan mampu memahami bagaimana sistem terdistribusi menjaga konsistensi urutan kejadian tanpa bergantung pada sinkronisasi waktu fisik.

4.2 Tutorial

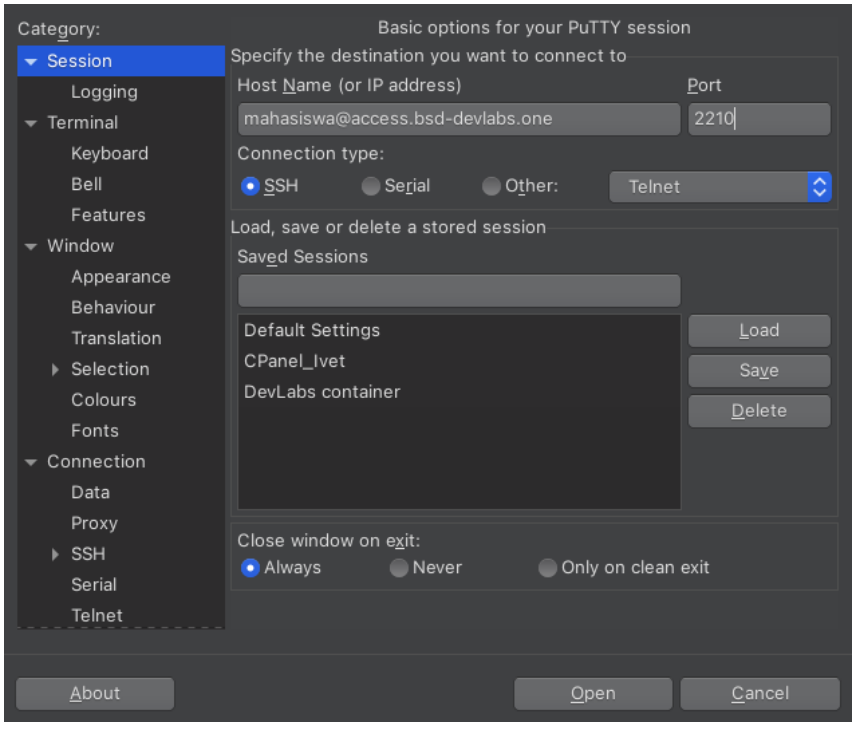
4.2.1 Akses SSH

1. Jalankan aplikasi PuTTY



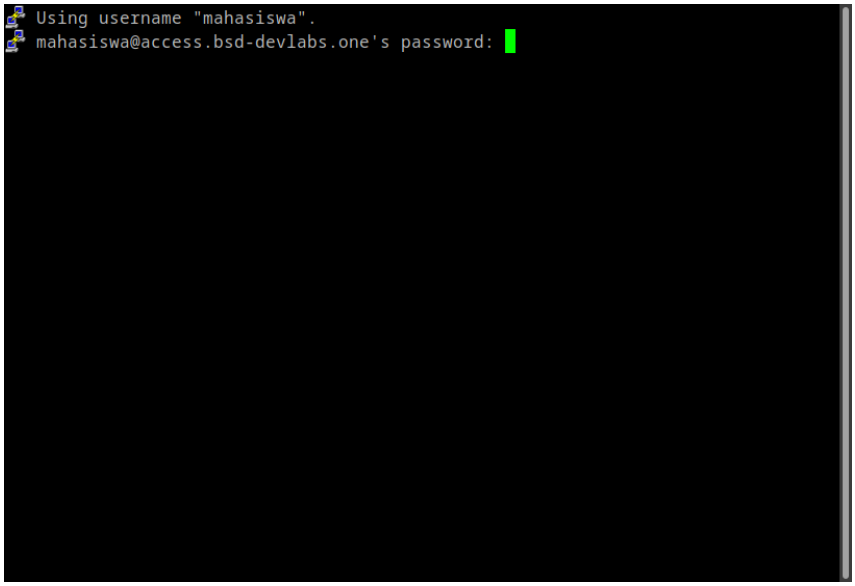
Gambar 4.1: Jalankan PuTTY

2. Masukkan *hostname*, *username* dan *port*
   - hostname : access.bsd-devlabs.one
   - username : mahasiswa
   - password : mahasiswa
   - port : 2210



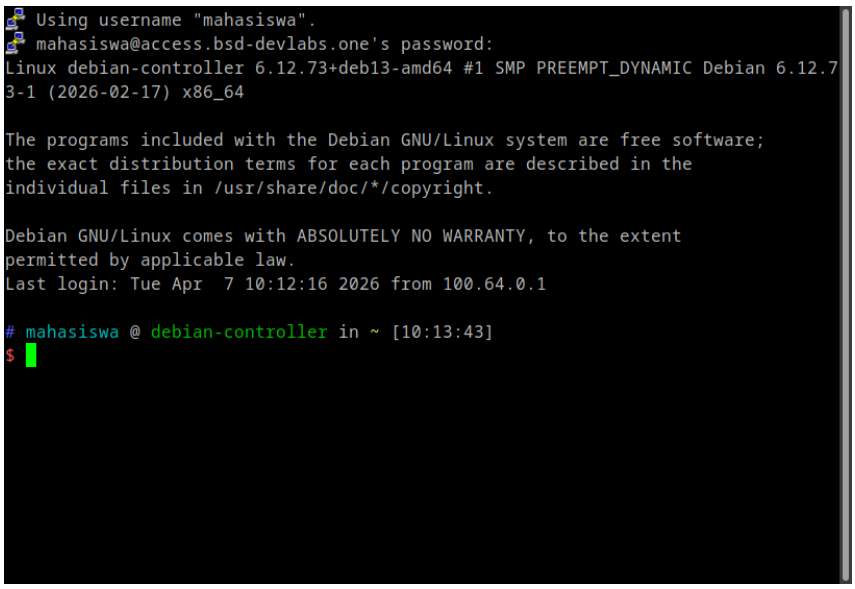
Gambar 4.2: Masukkan Akses

3. Masukkan *Password* ketika diminta



Gambar 4.3: Masukkan Password

4. Jika berhasil, maka PuTTY akan menampilkan prompt

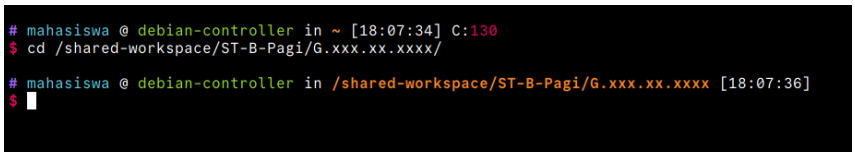


Gambar 4.4: Prompt Node Kontroler

5. Masuk ke folder mahasiswa masing-masing dengan perintah:
   - Kelas Pagi

     Perintah Terminal
     ```
     cd /shared-workspace/ST-B-Pagi/G.xxx.xx.xxxx/
     ```
   - Kelas Sore

     Perintah Terminal
     ```
     cd /shared-workspace/ST-B-Sore/G.xxx.xx.xxxx/
     ```



Gambar 4.5: Berpindah ke folder mahasiswa

4.2.2 Program Lamport

1. Di folder mahasiswa masing-masing, buat sebuah file dengan nama **lamport_clock.py** dengan perintah berikut

Perintah Terminal

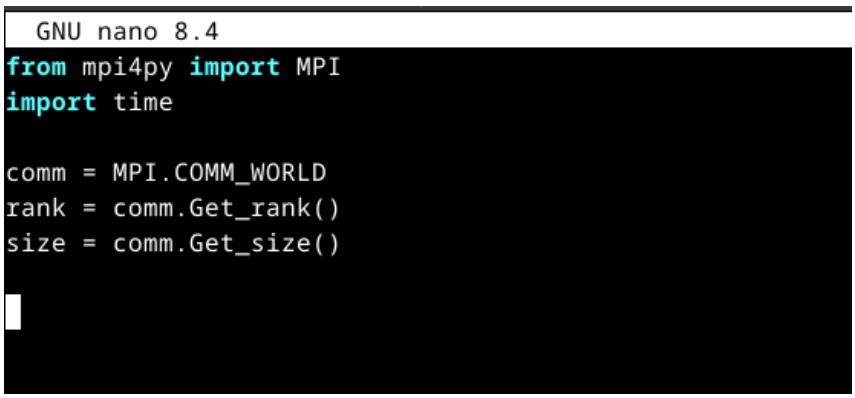
```
nano lamport_clock.py
```

Gambar 4.6: Membuat file lamport_clock.py

2. Isikan dengan kode berikut, bagian pertama adalah impor library dan inisialisasi awal

Perintah Terminal

```
from mpi4py import MPI
import time

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()
```

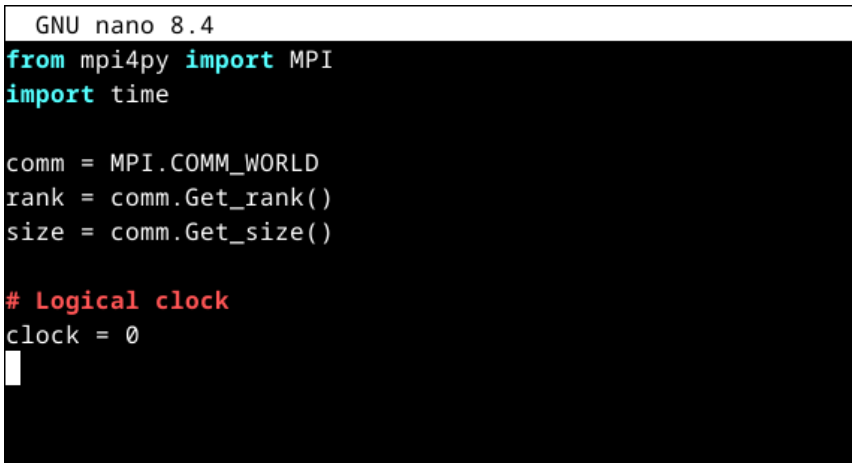


Gambar 4.7: Inisialisasi Kode

3. Berikutnya adalah membuat lokal `clock` yang nanti dimiliki oleh node yang aktif

Perintah Terminal

```
# Logical clock
clock = 0
```

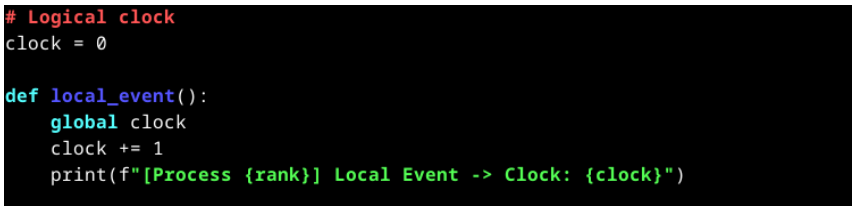


Gambar 4.8: Menambahkan Logical Clock

4. Setelah mmengkonfigurasikan *logical clock*, berikutnya adalah membuat fungsi pembantu untuk membuat *local event*. Letakkan setelah *clock*

Perintah Terminal

```
def local_event():
    global clock
    clock += 1
    print(f"[Process {rank}] Local Event -> Clock: {clock}")
```

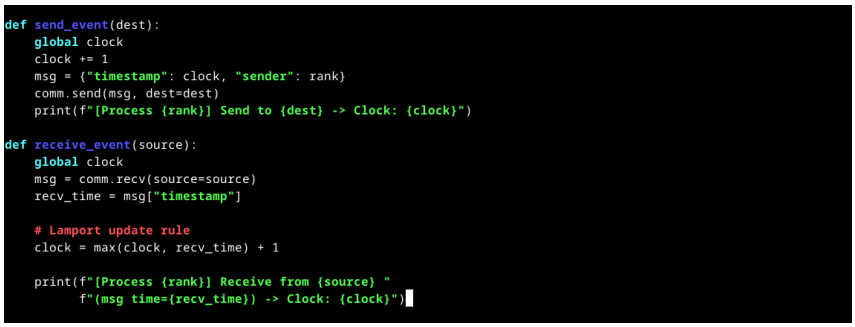


Gambar 4.9: Menambahkan Fungsi Local Event

5. Kode untuk event setelah ditambahkan, maka langkah berikutnya adalah membuat fungsi untuk *messaging event* ke node lainnya

Perintah Terminal

```
def send_event(dest):
    global clock
    clock += 1
    msg = {"timestamp": clock, "sender": rank}
    comm.send(msg, dest=dest)
    print(f"[Process {rank}] Send to {dest} -> Clock: {clock}")

def receive_event(source):
    global clock
    msg = comm.recv(source=source)
    recv_time = msg["timestamp"]

    # Update Lamport Clock
    clock = max(clock, recv_time) + 1

    print(f"[Process {rank}] Receive from {source} "
          f"(msg time={recv_time}) -> Clock: {clock}")
```



Gambar 4.10: Menambahkan Fungsi Komunikasi

6. Setelah membuat fungsi-fungsi yang diperlukan untuk *Lamport Clock*, berikutnya adalah membuat simulasi untuk lamport clock. Masukkan kode simulasi berikut menggunakan 3 (tiga) node

Perintah Terminal

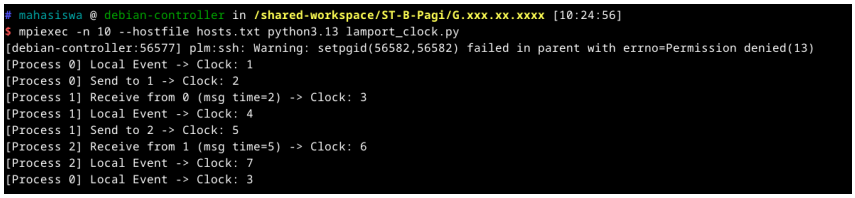
```
# Simulasi
if rank == 0:
    local_event()
    time.sleep(1)
    send_event(1)
    time.sleep(1)
    local_event()

elif rank == 1:
    receive_event(0)
    local_event()
    send_event(2)

elif rank == 2:
    receive_event(1)
    local_event()
```

Gambar 4.11: Menambahkan Kode Simulasi

7. Setelah kode lengkap, jalankan aplikasi dengan perintah

Perintah Terminal

```
mpiexec -n 10 --hostfile hosts.txt python3.13 lamport_clock.py
```



Gambar 4.12: Menjalankan Kode

8. Kode untuk *Logical Clock Lamport* ini memiliki keunikan yang dimana *event* yang dilakukan akan selalu berurutan. Coba jalankan hingga tiga kali

```
# mahasiswa @ debian-controller in /shared-workspace/ST-B-Pagi/G.xxx.xx.xxxx [10:24:56]
$ mpiexec -n 10 --hostfile hosts.txt python3.13 lamport_clock.py
[debian-controller:56577] plm:ssh: Warning: setpgid(56582,56582) failed in parent with errno=Permission denied(13)
[Process 0] Local Event -> Clock: 1
[Process 0] Send to 1 -> Clock: 2
[Process 1] Receive from 0 (msg time=2) -> Clock: 3
[Process 1] Local Event -> Clock: 4
[Process 1] Send to 2 -> Clock: 5
[Process 2] Receive from 1 (msg time=5) -> Clock: 6
[Process 2] Local Event -> Clock: 7
[Process 0] Local Event -> Clock: 3

# mahasiswa @ debian-controller in /shared-workspace/ST-B-Pagi/G.xxx.xx.xxxx [10:25:34]
$ mpiexec -n 10 --hostfile hosts.txt python3.13 lamport_clock.py
[debian-controller:56608] plm:ssh: Warning: setpgid(56613,56613) failed in parent with errno=Permission denied(13)
[Process 0] Local Event -> Clock: 1
[Process 0] Send to 1 -> Clock: 2
[Process 1] Receive from 0 (msg time=2) -> Clock: 3
[Process 1] Local Event -> Clock: 4
[Process 1] Send to 2 -> Clock: 5
[Process 2] Receive from 1 (msg time=5) -> Clock: 6
[Process 2] Local Event -> Clock: 7
[Process 0] Local Event -> Clock: 3

# mahasiswa @ debian-controller in /shared-workspace/ST-B-Pagi/G.xxx.xx.xxxx [10:26:20]
$ mpiexec -n 10 --hostfile hosts.txt python3.13 lamport_clock.py
[debian-controller:56627] plm:ssh: Warning: setpgid(56636,56636) failed in parent with errno=Permission denied(13)
[Process 0] Local Event -> Clock: 1
[Process 0] Send to 1 -> Clock: 2
[Process 1] Receive from 0 (msg time=2) -> Clock: 3
[Process 1] Local Event -> Clock: 4
[Process 1] Send to 2 -> Clock: 5
[Process 2] Receive from 1 (msg time=5) -> Clock: 6
[Process 2] Local Event -> Clock: 7
[Process 0] Local Event -> Clock: 3
```

Gambar 4.13: Konsistensi Logical Clock

9. Ubah file **lamport_clock.py**, tambahkan kode baru di node 2 dan tambahkan node baru di bagian simulasi seperti berikut

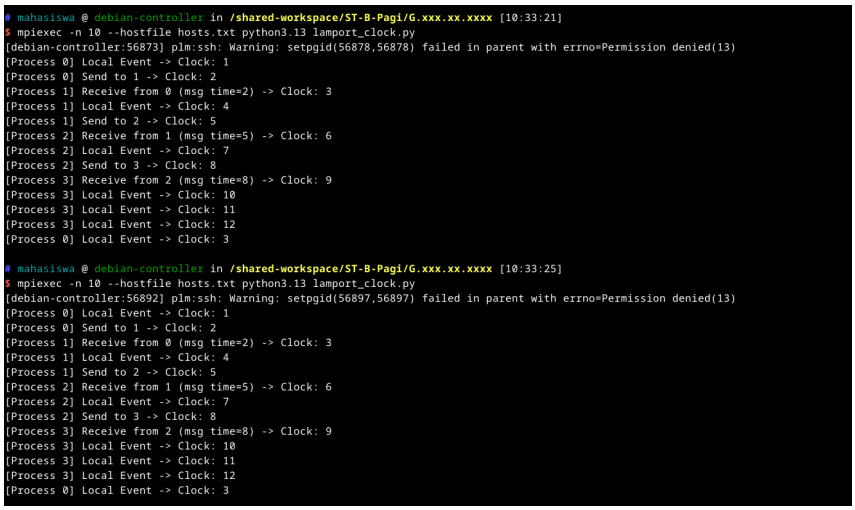
Perintah Terminal

```
elif rank == 2:
    receive_event(1)
    local_event()
    send_event(3)

elif rank == 3:
    receive_event(2)
    for _ in range(3):
        local_event()
```

```
elif rank == 2:
    receive_event(1)
    local_event()
    send_event(3)

elif rank == 3:
    receive_event(2)
    for _ in range(3):
        local_event()
```

Gambar 4.14: Memodifikasi Kode

10. Jalankan kembali dan lihat hasilnya



Gambar 4.15: Memodifikasi Kode

Bab 5

Mutual Exclusion

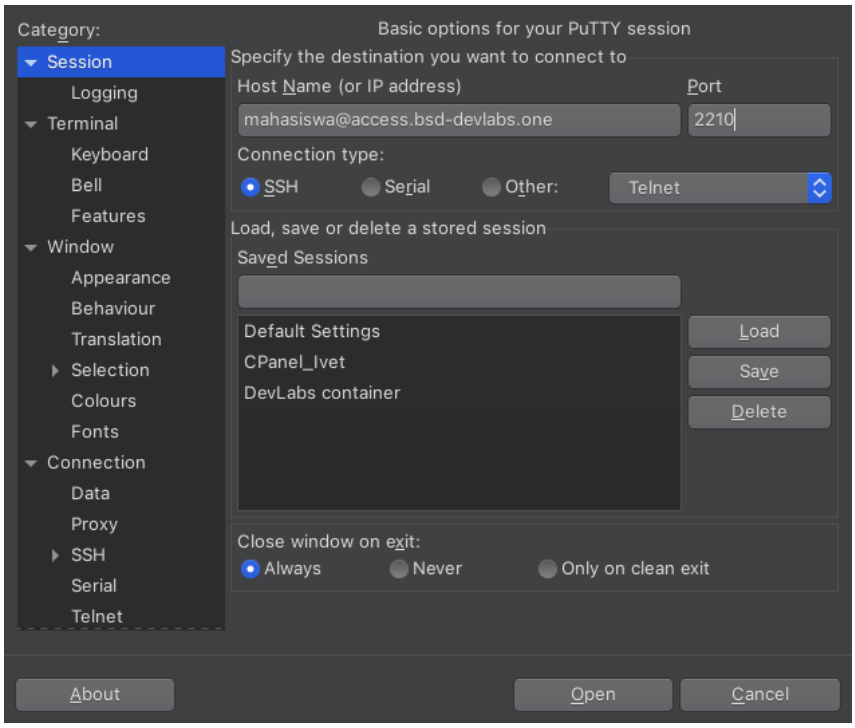
5.1 Pendahuluan

Mutual exclusion merupakan konsep fundamental dalam sistem terdistribusi yang bertujuan untuk menjamin bahwa hanya satu proses yang dapat mengakses critical section pada satu waktu, sehingga konsistensi data dan integritas sistem tetap terjaga. Dalam lingkungan terdistribusi, ketiadaan memori bersama dan sinkronisasi terpusat menyebabkan mekanisme pengendalian akses menjadi lebih kompleks dibandingkan sistem terpusat. Oleh karena itu, berbagai algoritma seperti token-based dan permission-based dikembangkan untuk mengatasi permasalahan tersebut secara efisien. Praktikum ini dirancang untuk memberikan pemahaman konseptual sekaligus pengalaman implementatif terhadap mekanisme mutual exclusion menggunakan Message Passing Interface (MPI) melalui pustaka mpi4py pada lingkungan OpenMPI, sehingga dapat diamati secara langsung bagaimana koordinasi antar proses dilakukan dalam menjaga eksklusivitas akses terhadap sumber daya bersama.

5.2 Tutorial

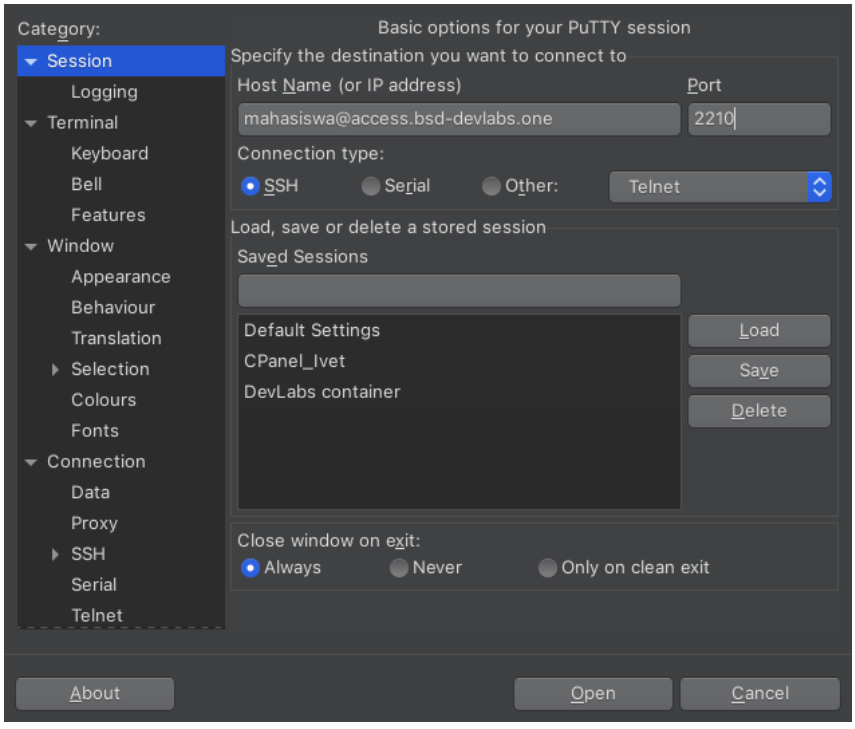
5.2.1 Akses SSH

1. Jalankan aplikasi PuTTY



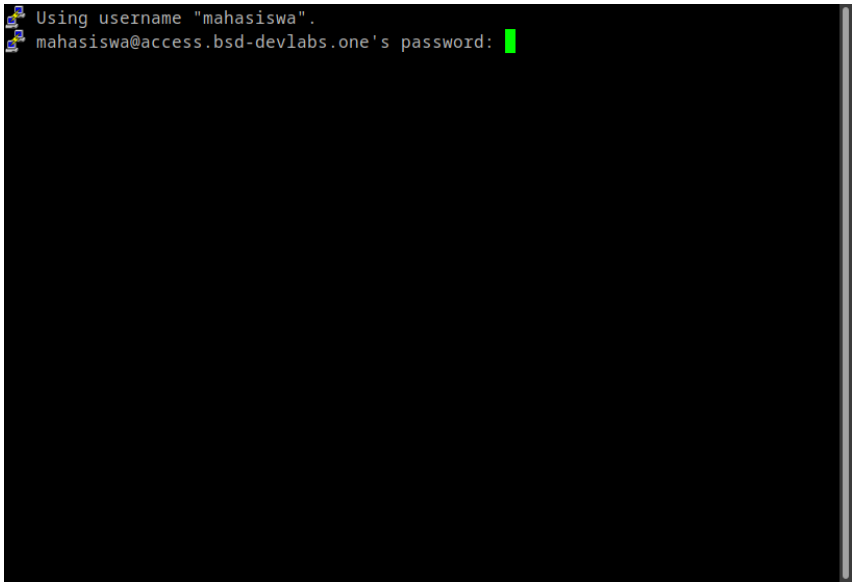
Gambar 5.1: Jalankan PuTTY

2. Masukkan *hostname*, *username* dan *port*
   - hostname : access.bsd-devlabs.one
   - username : mahasiswa
   - password : mahasiswa
   - port : 2210



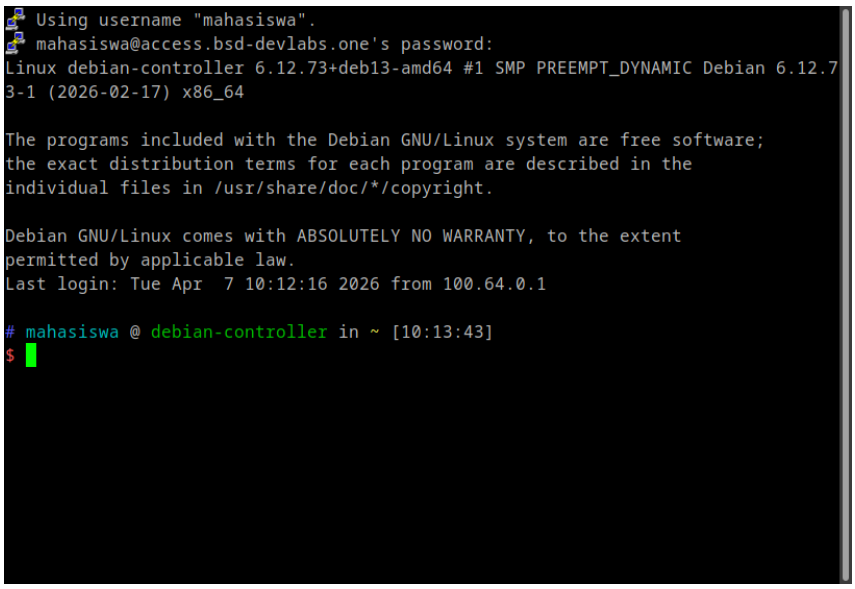
Gambar 5.2: Masukkan Akses

3. Masukkan *Password* ketika diminta



Gambar 5.3: Masukkan Password

4. Jika berhasil, maka PuTTY akan menampilkan prompt

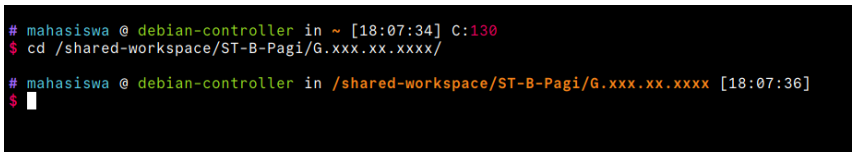


Gambar 5.4: Prompt Node Kontroler

5. Masuk ke folder mahasiswa masing-masing dengan perintah:
   - Kelas Pagi

     Perintah Terminal
     ```
     cd /shared-workspace/ST-B-Pagi/G.xxx.xx.xxxx/
     ```
   - Kelas Sore

     Perintah Terminal
     ```
     cd /shared-workspace/ST-B-Sore/G.xxx.xx.xxxx/
     ```



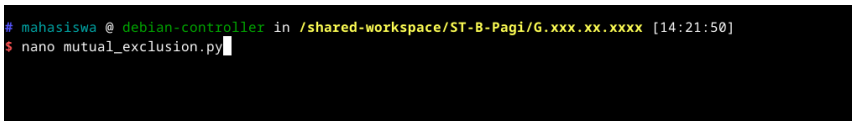
Gambar 5.5: Berpindah ke folder mahasiswa

5.2.2 Mutual Exclusion

1. Buat sebuah file dengan nama **mutual_exclusion.py**

Perintah Terminal

```
nano mutual_exclusion.py
```

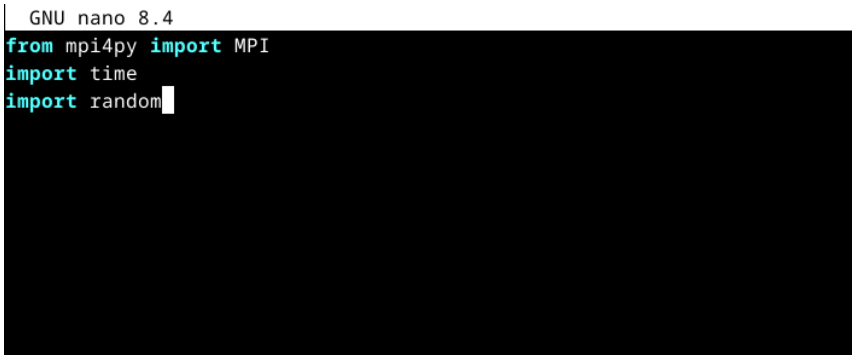


Gambar 5.6: Membuat file mutual_exclusion.py

2. Impor library yang diperlukan

Perintah Terminal

```
from mpi4py import MPI
import time
import random
```



Gambar 5.7: Menambahkan Library

3. Inisialisasi MPI

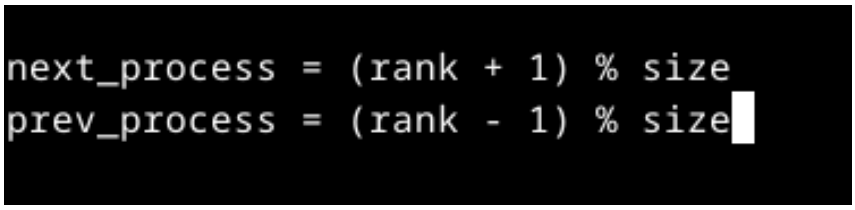
Perintah Terminal

```
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()
```

Gambar 5.8: Menambahkan Inisialisasi MPI

4. Berikutnya menggunakan mekanisme **Ring** untuk **mutual exclusion**. Masukkan kode berikut

Perintah Terminal
```
next_process = (rank + 1) % size
prev_process = (rank - 1) % size
```



Gambar 5.9: Menambahkan Variabel untuk Ring

5. Setelah itu inisialisasi Token dan dimulai dari Node 0

Perintah Terminal
```
token = None

if rank == 0:
    token = "TOKEN"
```

Gambar 5.10: Menambahkan Variabel untuk Ring

6. Kode terakhir adalah simulasi mutual exclusion. Masukkan kode berikut dibagian akhir file

Perintah Terminal
```
for i in range(1):  # setiap proses mencoba 3 kali
    if token is None:
        # Tunggu token dari proses sebelumnya
        token = comm.recv(source=prev_process)

    print(f"Process {rank} menerima token")

    # Masuk critical section
    print(f">>> Process {rank} MASUK critical section")
    time.sleep(random.uniform(0.5, 2))
    print(f"<<< Process {rank} KELUAR critical section")

    # Kirim token ke proses berikutnya
    comm.send(token, dest=next_process)
    token = None

    # Delay simulasi
    time.sleep(random.uniform(0.5, 2))
```

```python
for i in range(3):  # setiap proses mencoba 3 kali
    if token is None:
        # Tunggu token dari proses sebelumnya
        token = comm.recv(source=prev_process)

    print(f"Process {rank} menerima token")

    # Masuk critical section
    print(f">>> Process {rank} MASUK critical section")
    time.sleep(random.uniform(0.5, 2))
    print(f"<<< Process {rank} KELUAR critical section")

    # Kirim token ke proses berikutnya
    comm.send(token, dest=next_process)
    token = None

    # Delay simulasi
    time.sleep(random.uniform(0.5, 2))
```

Gambar 5.11: Menambahkan Variabel untuk Ring

7. Jalankan kode dengan menggunakan perintah

Perintah Terminal

```
mpiexec -n 10 --hostfile hosts.txt python3.13 mutual_exclusion.py
```

```
# mahasiswa @ debian-controller in /shared-workspace/ST-B-Pagi/G.xxx.xx.xxxx [14:40:45]
$ mpiexec -n 10 --hostfile hosts.txt python3.13 mutual_exclusion.py
```

Gambar 5.12: Menjalankan Kode

8. Lihat apa yang dilakukan oleh kode tersebut. Token dimiliki oleh Proses 0, sehingga proses ini diizinkan untuk akses ekslusif. Secara bergantian melingkar mengakses *critical section*

```
# mahasiswa @ debian-controller in /shared-workspace/ST-B-Pagi/G.xxx.xx.xxxx [14:40:45]
$ mpiexec -n 10 --hostfile hosts.txt python3.13 mutual_exclusion.py
[debian-controller:01918] plm:ssh: Warning: setpgid(1923,1923) failed in parent with errno=Permission denied(13)
Process 0 menerima token
>>> Process 0 MASUK critical section
<<< Process 0 KELUAR critical section

Process 1 menerima token
>>> Process 1 MASUK critical section
<<< Process 1 KELUAR critical section

Process 2 menerima token
>>> Process 2 MASUK critical section
<<< Process 2 KELUAR critical section

Process 3 menerima token
>>> Process 3 MASUK critical section
<<< Process 3 KELUAR critical section

Process 4 menerima token
>>> Process 4 MASUK critical section
<<< Process 4 KELUAR critical section

Process 5 menerima token
>>> Process 5 MASUK critical section
<<< Process 5 KELUAR critical section

Process 6 menerima token
>>> Process 6 MASUK critical section
<<< Process 6 KELUAR critical section

Process 7 menerima token
>>> Process 7 MASUK critical section
<<< Process 7 KELUAR critical section

Process 8 menerima token
>>> Process 8 MASUK critical section
<<< Process 8 KELUAR critical section
```

Gambar 5.13: Pola Akses Mutual Exclusion

Bab 6

Leader Election

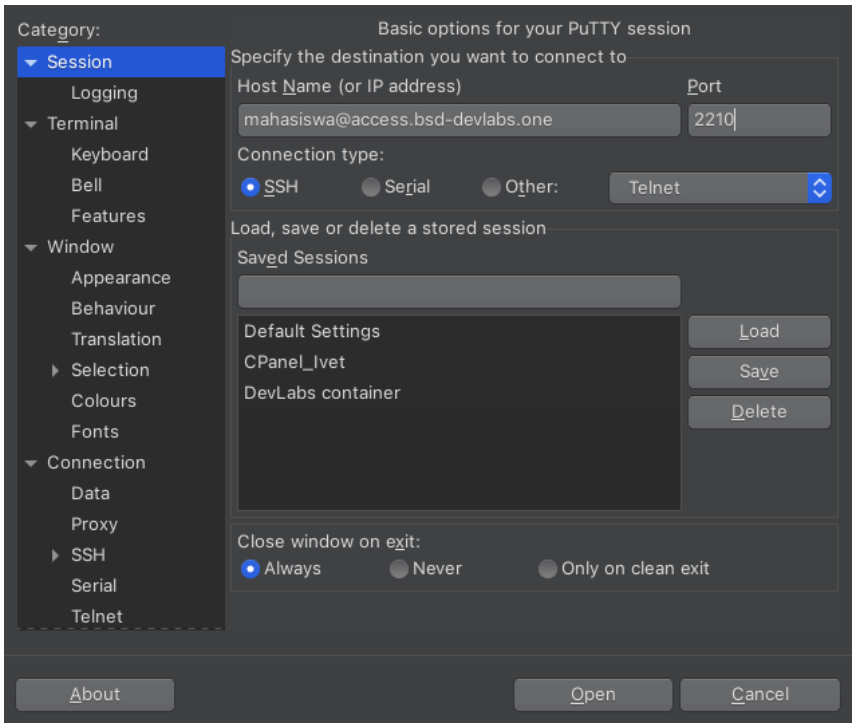
6.1 Pendahuluan

Praktikum ini membahas implementasi mekanisme leader election dalam sistem terdistribusi menggunakan Message Passing Interface (MPI) melalui pustaka mpi4py. Dalam lingkungan terdistribusi, pemilihan satu node sebagai koordinator menjadi aspek penting untuk menjaga konsistensi, koordinasi tugas, serta toleransi terhadap kegagalan. Dua algoritma klasik yang digunakan dalam praktikum ini adalah Bully Algorithm dan Ring Algorithm, yang masing-masing memiliki karakteristik berbeda dalam hal kompleksitas komunikasi, struktur jaringan, dan respons terhadap kegagalan node. Melalui simulasi menggunakan beberapa proses MPI, praktikum ini bertujuan memberikan pemahaman konseptual sekaligus pengalaman praktis mengenai bagaimana node berkomunikasi, mendeteksi kegagalan, dan menentukan pemimpin secara terdistribusi.

6.2 Tutorial

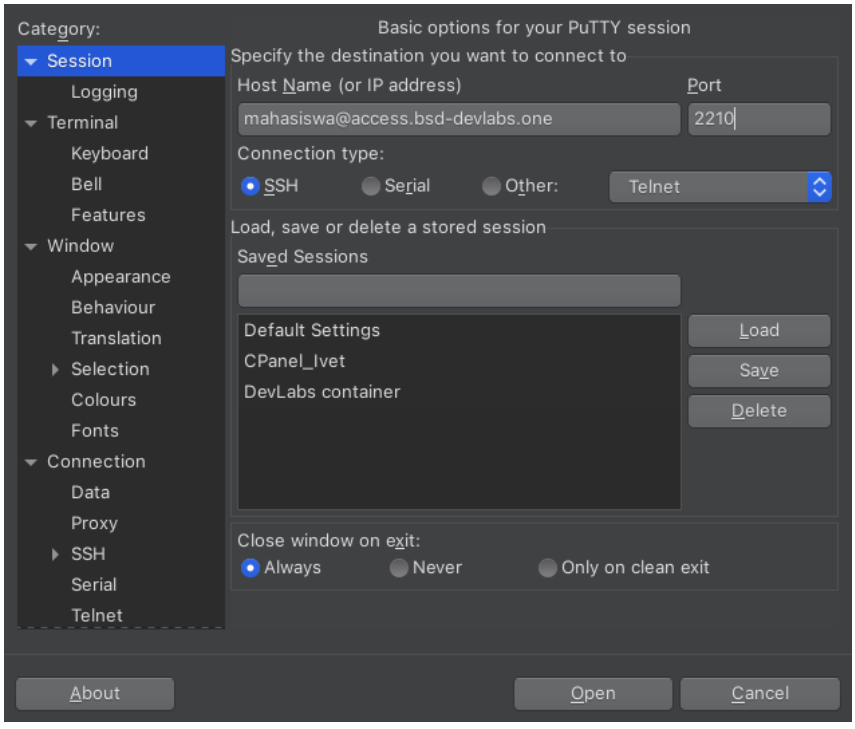
6.2.1 Akses SSH

1. Jalankan aplikasi PuTTY



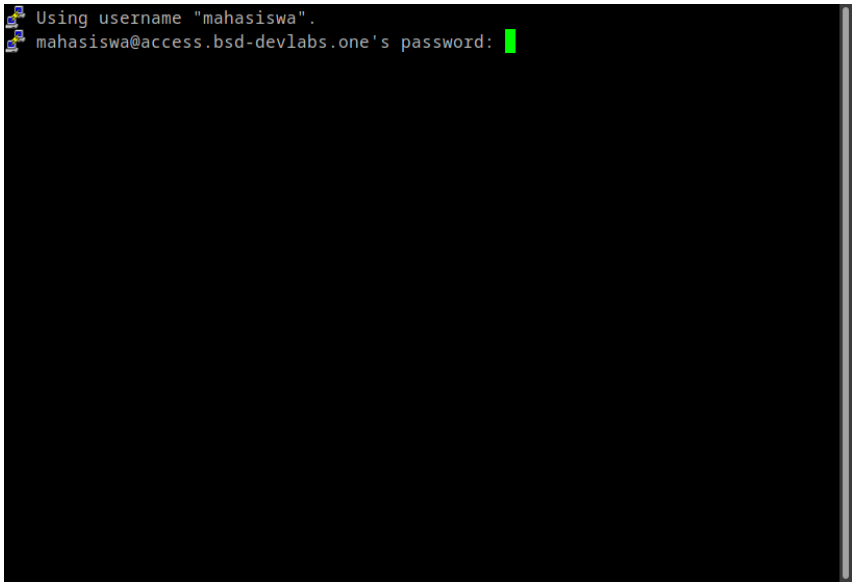
Gambar 6.1: Jalankan PuTTY

2. Masukkan *hostname*, *username* dan *port*
   - hostname : access.bsd-devlabs.one
   - username : mahasiswa
   - password : mahasiswa
   - port : 2210



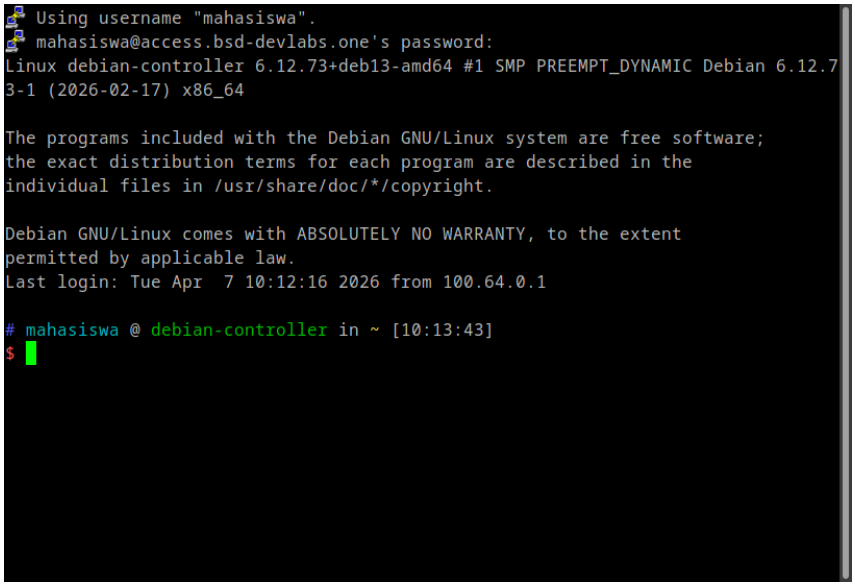
Gambar 6.2: Masukkan Akses

3. Masukkan *Password* ketika diminta



Gambar 6.3: Masukkan Password

4. Jika berhasil, maka PuTTY akan menampilkan prompt

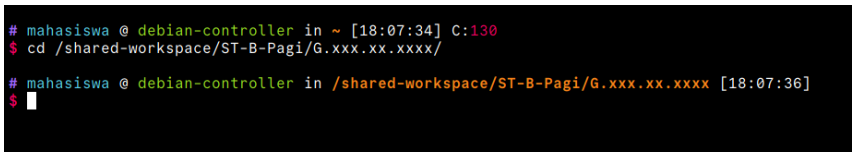


Gambar 6.4: Prompt Node Kontroler

5. Masuk ke folder mahasiswa masing-masing dengan perintah:
   - Kelas Pagi

     Perintah Terminal
     ```
     cd /shared-workspace/ST-B-Pagi/G.xxx.xx.xxxx/
     ```
   - Kelas Sore

     Perintah Terminal
     ```
     cd /shared-workspace/ST-B-Sore/G.xxx.xx.xxxx/
     ```



Gambar 6.5: Berpindah ke folder mahasiswa

6.2.2 Bully Algorithm

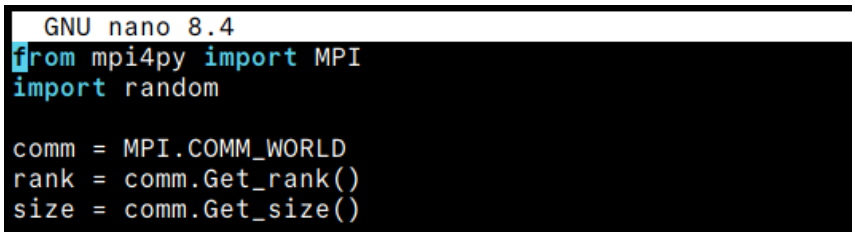
1. Buat sebuah file dengan nama **bully.py**

Perintah Terminal
```
nano bully.py
```

Gambar 6.6: Membuat file bully.py

2. Masukkan kode inisialisasi berikut ini

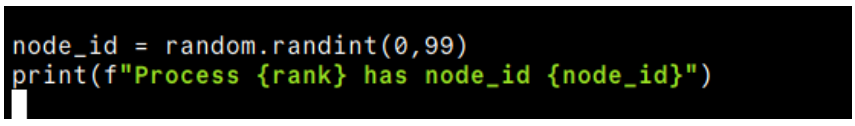
Perintah Terminal
```
from mpi4py import MPI
import random

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()
```



Gambar 6.7: Memasukkan Kode Inisialisasi

3. Lanjutkan kode dengan menambah random ID kepada masing-masing Node

Perintah Terminal
```
node_id = random.randint(0,99)
print(f"Process rank has node_id node_id")
```



Gambar 6.8: Memasukkan Kode Penambah ID

4. Langkah berikutnya adalah mengumpulkan semua Node ID ke Node 0 sebagai Root

Perintah Terminal
```
all_ids = comm.gather(node_id, root=0)
```

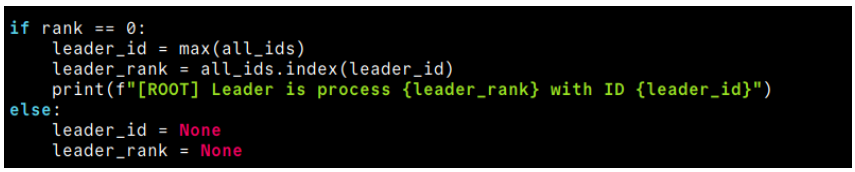
```
all_ids = comm.gather(node_id, root=0)
```

Gambar 6.9: Memasukkan Kode Agregasi node ID

5. Blok kode berikutnya memiliki fungsi untuk memulai proses Election dimulai dari Node 0

Perintah Terminal

```
if rank == 0:
    leader_id = max(all_ids)
    leader_rank = all_ids.index(leader_id)
    print(f"[ROOT] Leader is process {leader_rank} with ID {leader_id}")
else:
    leader_id = None
    leader_rank = None
```

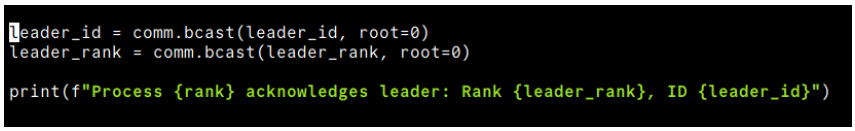


Gambar 6.10: Memasukkan Kode Memulai Election

6. Node dengan ID terbesar sebagai leader akan membroadcast ke semua Node

Perintah Terminal

```
leader_id = comm.bcast(leader_id, root=0)
leader_rank = comm.bcast(leader_rank, root=0)

print(f"Process {rank} acknowledges leader: Rank {leader_rank}, ID {leader_id}")
```

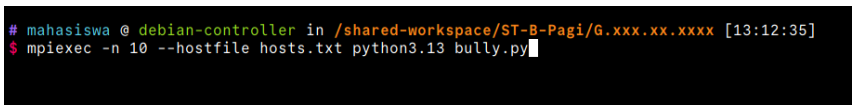


Gambar 6.11: Memasukkan Kode Broadcast Leader

7. Jalankan kode dengan perintah

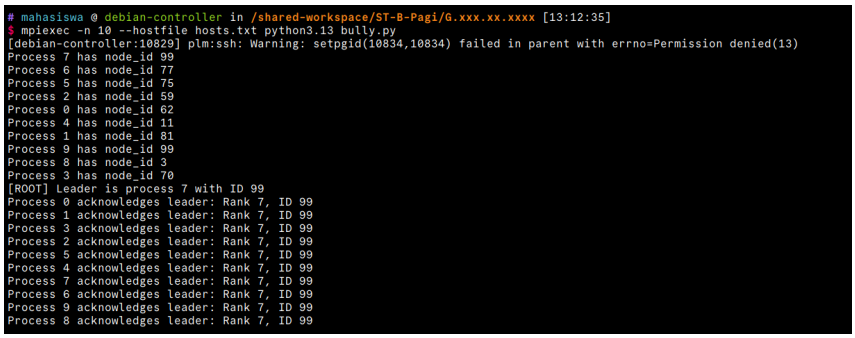
Perintah Terminal

```
mpiexec -n 10 --hostfile hosts.txt python3.13 bully.py
```



Gambar 6.12: Menjalankan Kode

8. Setiap proses eksekusi akan menghasilkan ID yang random, sehingga Leader bisa berbeda



Gambar 6.13: Hasil Menjalankan Kode

Bab 7

Konsistensi Data

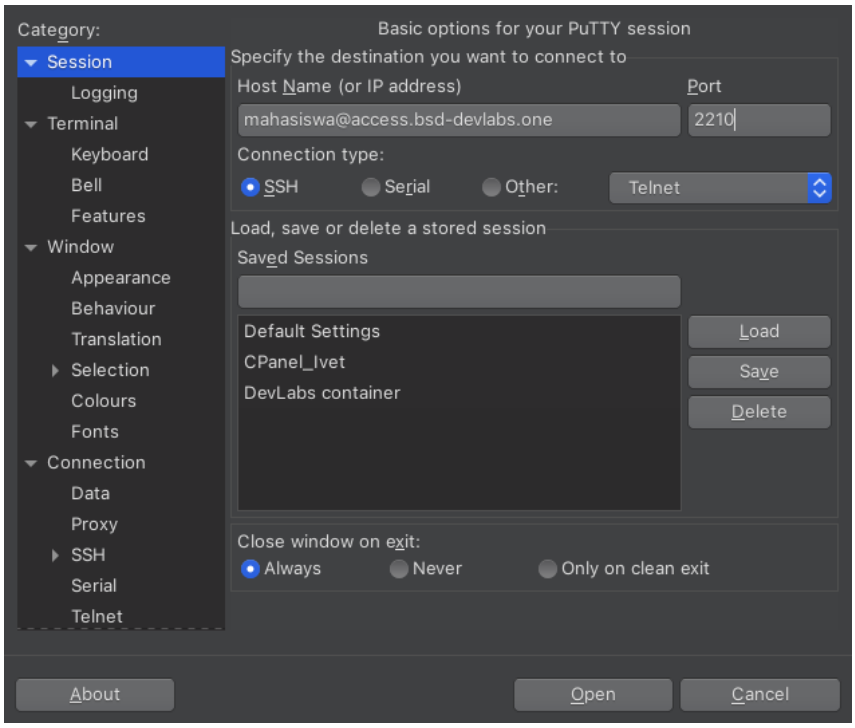
7.1 Pendahuluan

Praktikum ini membahas konsep konsistensi data dalam sistem terdistribusi dengan menitikberatkan pada perbedaan antara model strong consistency dan eventual consistency melalui simulasi menggunakan Open MPI dan mpi4py. Dalam lingkungan terdistribusi, setiap node memiliki salinan data yang dapat berubah akibat proses pembaruan dan keterlambatan komunikasi, sehingga berpotensi menimbulkan inkonsistensi sementara. Melalui eksperimen yang melibatkan pembaruan variabel global serta simulasi delay komunikasi, praktikum ini bertujuan untuk mengamati bagaimana sinkronisasi ketat menghasilkan konsistensi yang kuat, serta bagaimana pendekatan asinkron memungkinkan sistem tetap berjalan meskipun terjadi perbedaan nilai antar node dalam periode tertentu. Pemahaman terhadap kedua model ini menjadi penting dalam perancangan sistem modern seperti komputasi awan dan Internet of Things yang menuntut keseimbangan antara kinerja, skalabilitas, dan keakuratan data.

7.2 Tutorial

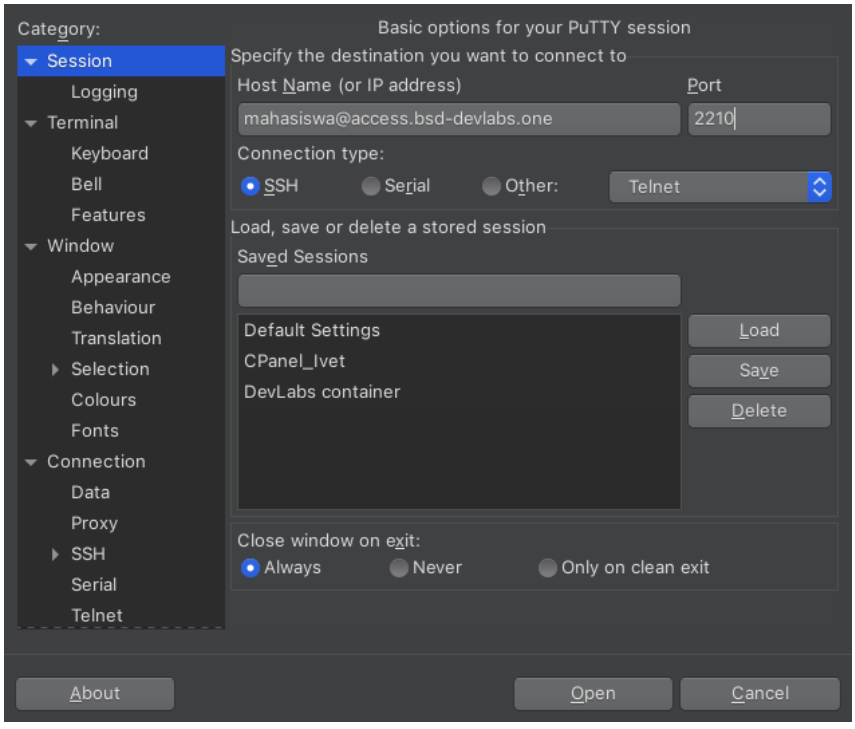
7.2.1 Akses SSH

1. Jalankan aplikasi PuTTY



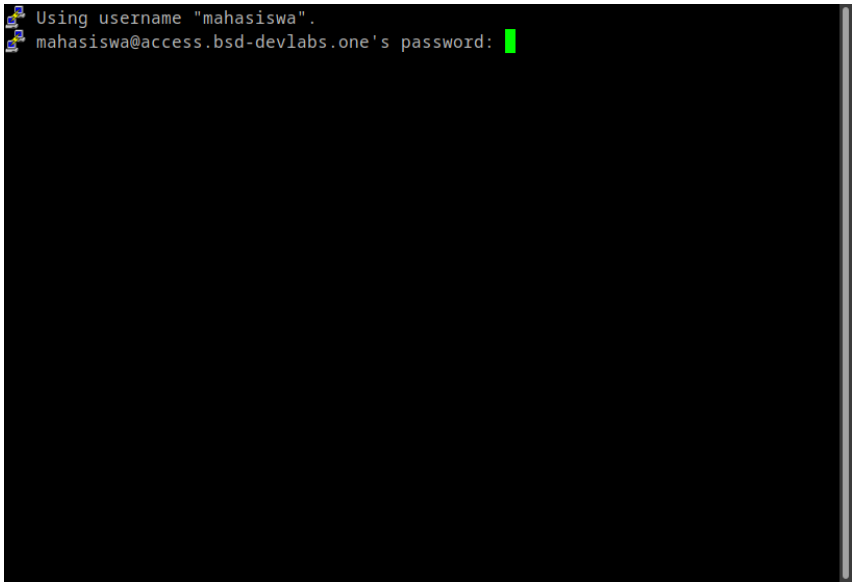
Gambar 7.1: Jalankan PuTTY

2. Masukkan *hostname*, *username* dan *port*
    - hostname : access.bsd-devlabs.one
    - username : mahasiswa
    - password : mahasiswa
    - port : 2210



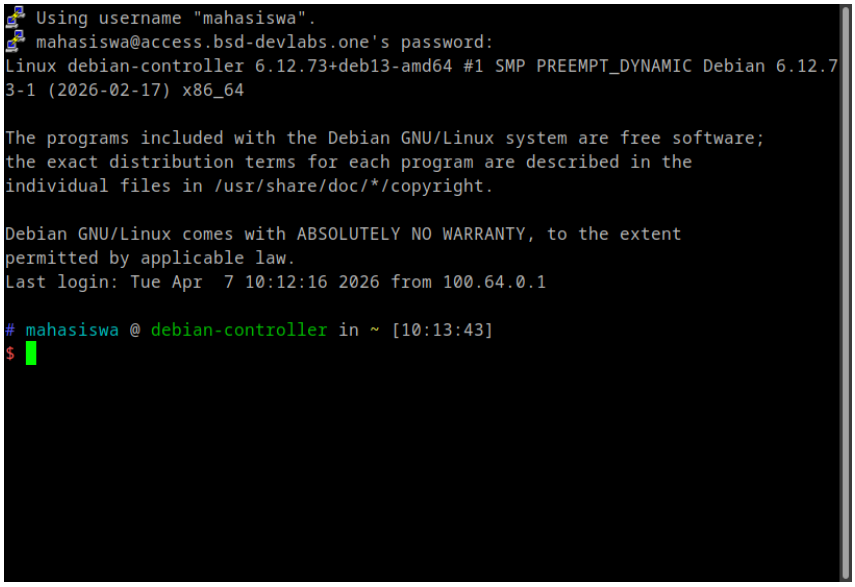
Gambar 7.2: Masukkan Akses

3. Masukkan *Password* ketika diminta



Gambar 7.3: Masukkan Password

4. Jika berhasil, maka PuTTY akan menampilkan prompt



Gambar 7.4: Prompt Node Kontroler

5. Masuk ke folder mahasiswa masing-masing dengan perintah:

   - Kelas Pagi

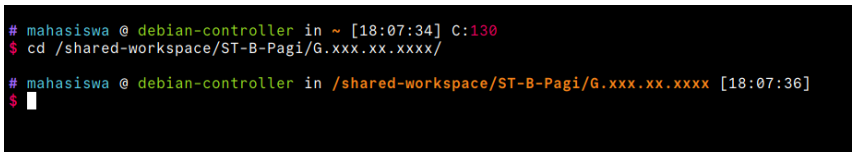
     Perintah Terminal

     ```
     cd /shared-workspace/ST-B-Pagi/G.xxx.xx.xxxx/
     ```

   - Kelas Sore

     Perintah Terminal

     ```
     cd /shared-workspace/ST-B-Sore/G.xxx.xx.xxxx/
     ```



Gambar 7.5: Berpindah ke folder mahasiswa

7.2.2 Konsistensi Kuat

1. Buat sebuah file dengan nama **strong.py**

Perintah Terminal

```
nano strong.py
```

Gambar 7.6: Membuat File strong.py

2. Masukkan potongan kode berikut

- **Kode Library dan Param**

Perintah Terminal

```
from mpi4py import MPI
import time

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

# Variabel global
global_value = 0
```

Gambar 7.7: Kode Library

- **Kode Simulasi**

Perintah Terminal

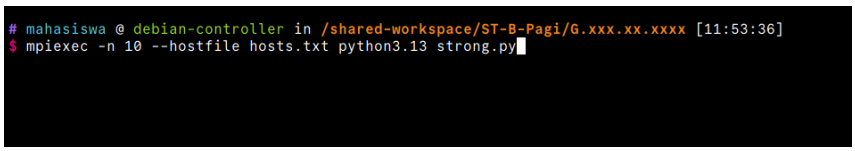
```
if rank == 0:
    print(f"[Node rank] Updating global value...")
    global_value = 100
    time.sleep(2)  # simulasi delay update

# Broadcast ke semua node (blocking)
global_value = comm.bcast(global_value, root=0)

# Sinkronisasi
comm.barrier()

print(f"[Node rank] Global Value = global_value")
```

Gambar 7.8: Kode Simulasi

3. Jalankan kode dengan perintah

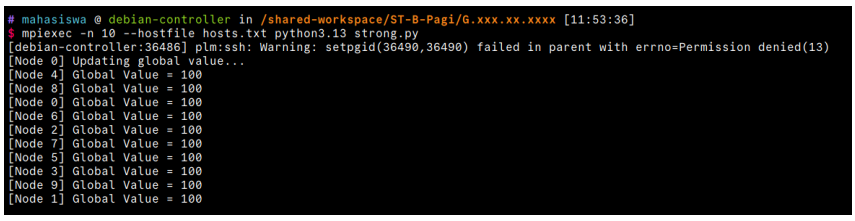
Perintah Terminal

```
mpiexec -n 10 --hostfile hosts.txt python3.13 strong.py
```



Gambar 7.9: Menjalankan Kode

4. Hasil yang muncul, semua node menampilkan data 100 dan bukan 0



Gambar 7.10: Hasil Simulasi

7.2.3 Konsistensi Eventual

1. Buat file dengan nama **eventual.py**

Perintah Terminal

```
nano eventual.py
```

Gambar 7.11: Membuat File Eventual

2. Di dalam nya, masukkan kode berikut

- **Kode Library dan Parameter**

Perintah Terminal

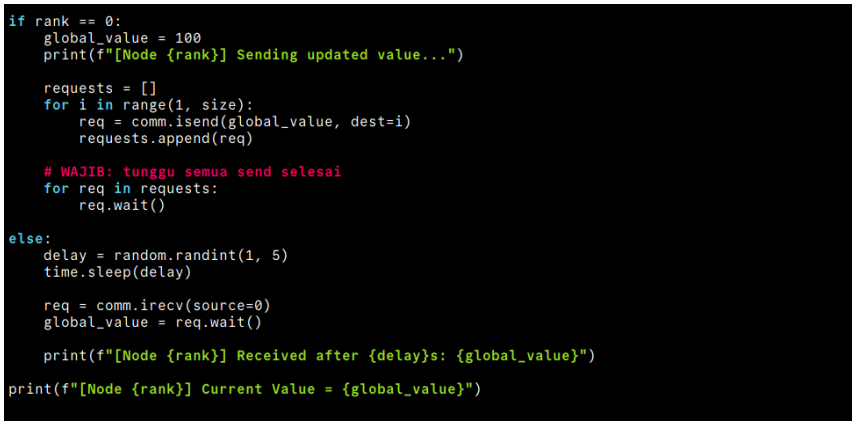
```
from mpi4py import MPI
import time
import random

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

global_value = 0
```

Gambar 7.12: Kode Library dan Parameter

- **Kode Simulasi**

Perintah Terminal

```
if rank == 0:
    global_value = 100
    print(f"[Node {rank}] Sending updated value...")

    requests = []
    for i in range(1, size):
        req = comm.isend(global_value, dest=i)
        requests.append(req)

    for req in requests:
        req.wait()

else:
    delay = random.randint(1, 5)
    time.sleep(delay)

    req = comm.irecv(source=0)
    global_value = req.wait()

    print(f"[Node {rank}] Received after {delay}s: {global_value}")

print(f"[Node {rank}] Current Value = {global_value}")
```

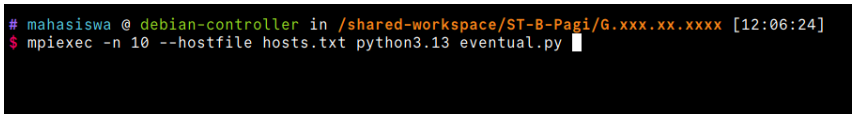


Gambar 7.13: Kode Simulasi

3. Jalankan kode dengan perintah

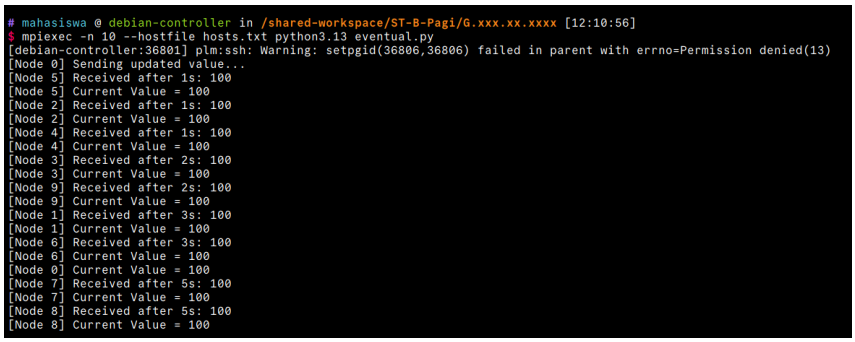
Perintah Terminal

```
mpiexec -n 10 --hostfile hosts.txt python3.13 eventual.py
```



Gambar 7.14: Menjalankan Kode

4. Hasil yang dimunculkan adalah semua node akan menerima global value yang sudah diupdate meskipun telah melewati beberapa detik



Gambar 7.15: Menjalankan Kode