

# Migration and Management of Process in JOS

## CSE506 Final Project

Dongli Zhang

Department of Computer Science, Stony Brook University

### Introduction

Virtualization is a popular research topic in computer science. Virtual machine can facilitate our work. We can save a running operating system as an image file on the disk and later reload the image. A necessary function of virtual machine is virtual machine migration over network. It involves the manipulation of page table and pages. Thus, to help me understand the migration of virtual machine, I implement a set of commands in this project to manage processes.

In this project, I primarily add two new features to JOS. First, I add the commands to save the process as an image on the file system and then later to load the image back to memory. Second, I implement two methods to move a process from one machine to another machine over TCP.

The following is a list of my works:

1. **ps** command to list status for all processes.
2. **pause** command to pause a running process.
3. **resume** command to resume a sleeping process.
4. **kill** command to stop a running or sleeping process.
5. **copy** command to clone a process on the same machine.
6. **save** command to save a sleeping process as an image file on the disk of JOS.
7. **load** command to load an image of process on the disk back to the kernel.
8. **migrate\_send\_normal** and **migrate\_recv\_normal** commands to move a process from one machine to another machine without the help from page fault handler. That is, all memory pages will be dumped to the second machine together.
9. **migrate\_send\_pgfault** and **migrate\_recv\_pgfault** commands to move a process from one machine to another machine with the help from page fault handler. That is, only page table entries will be dumped to the second machine, and the page fault handler will fetch required pages from the first machine.

### Command Manual

1. **ps**
2. **pause** [env\_id]
3. **resume** [env\_id]

4. **kill** [env\_id]
5. **copy** [env\_id]
6. **save** [env\_id] [image\_file\_name]
7. **load** [image\_file\_name]
8. **migrate\_send\_normal** [env\_id] [remote\_ip] [remote\_port50001] [local\_ip]
9. **migrate\_recv\_normal**
10. **migrate\_send\_pgfault** [env\_id] [remote\_ip] [remote\_port50002]  
[local\_ip] [local\_port50003]
11. **migrate\_recv\_pgfault**

## Basic Commands

### *ps, pause, resume, kill, copy*

These commands are not difficult to implement. I just manage all environments with the help of *envs* in user space and system calls.

## Migration of Processes

### 1. **migrate\_send\_normal** and **migrate\_recv\_normal**

This is the first method I use to implement the process migration. I will move all page tables and memory pages to another machine over TCP connection. Then I will resume the process on the remote machine. The steps are as follows:

- (1) Send the Trapframe structure to the remote machine.
- (2) Traverse page table entries from UTEXT to User Normal Stack. For each page entry, if there is a corresponding page, divide the page into four 1024-bytes packages and send each of them to the remote machine.
- (3) Reassemble all pages on the remote machine and copy them and the Trapframe to the user level memory space.

### 2. **migrate\_send\_pgfault** and **migrate\_recv\_pgfault**

This is the second method I use to implement the process migration. I will only move all page table entries to the remote machine. Then I resume the process on the remote machine. There is a super process (ENV 3) on the remote machine to handle the user level page fault. If there is a page fault, the super process will contact the original machine and fetch the required page from the original machine over TCP. Finally, it inserts the page to the memory space of process. The steps are as follows:

- (1) Send the Trapframe structure to the remote machine.
- (2) Traverse page table entries from UTEXT to User Normal Stack. For each entry, if there is a corresponding page, send this entry to the remote machine. Now this machine will listen and handle page fault request from remote machine.
- (3) Insert the received page table entries on the remote machine and then

resume the process. If there is a user level page fault, send a request to the super page fault handler on this machine. The super process will fetch the required page from the original machine.

## Process Image Management

### 1. **save**

This command is quite like the **migrate\_send\_normal** command. Instead of receiving the pages from the network service with read function and network descriptor, I receive the pages from the file system service with read function and file descriptor.

### 2. **load**

This command is quite like the **migrate\_recv\_normal** command. Instead of receiving the pages from the network service with read function and network descriptor, I receive the pages from the file system service with read function and file descriptor.

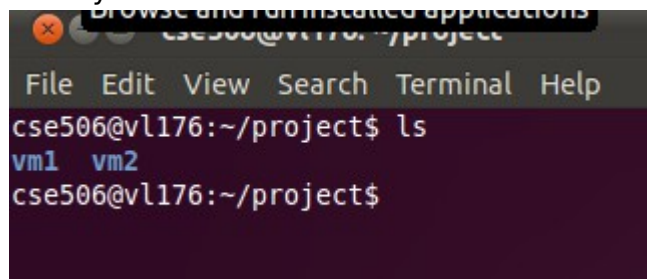
The following is the tutor of each command.

## Manual: An example for test

The following is an example to test all my programs. You can do the same work to test my program.

The source code is in my submitted projects. There is a package called project.tar.gz.

After you decompress the project package, you will have a project folder with two vm subfolder. They are VM-1 and VM-2.

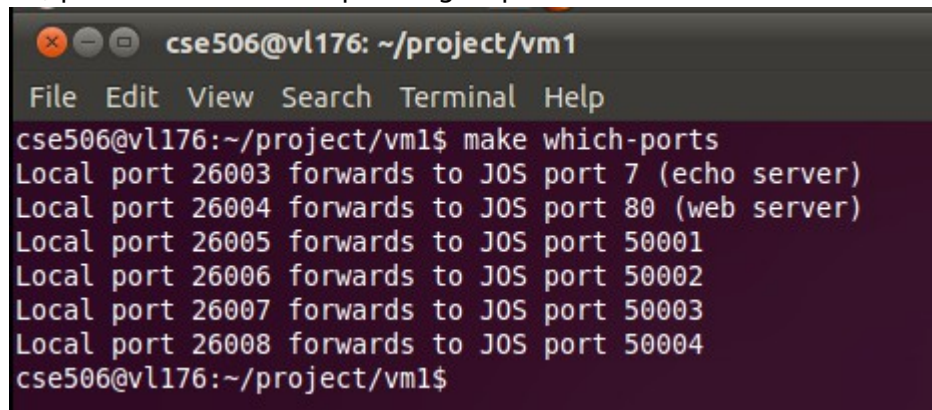
A terminal window with a dark background and light-colored text. The window title is "Browse and Run Installed Applications". The prompt is "cse506@vl176: ~/project". The command "ls" has been entered, and the output shows "vm1" and "vm2" in blue text. The prompt is now "cse506@vl176: ~/project\$".

```
File Edit View Search Terminal Help
cse506@vl176:~/project$ ls
vm1  vm2
cse506@vl176:~/project$
```

On both VMs

In this project, we will move a process from VM-1 to VM-2. The listening port for Normal Migration is 50001 and the listening port for PGfault Migration 50002. The listening port to handle page fault on VM-1 is 50003. We need to get the corresponding ports.

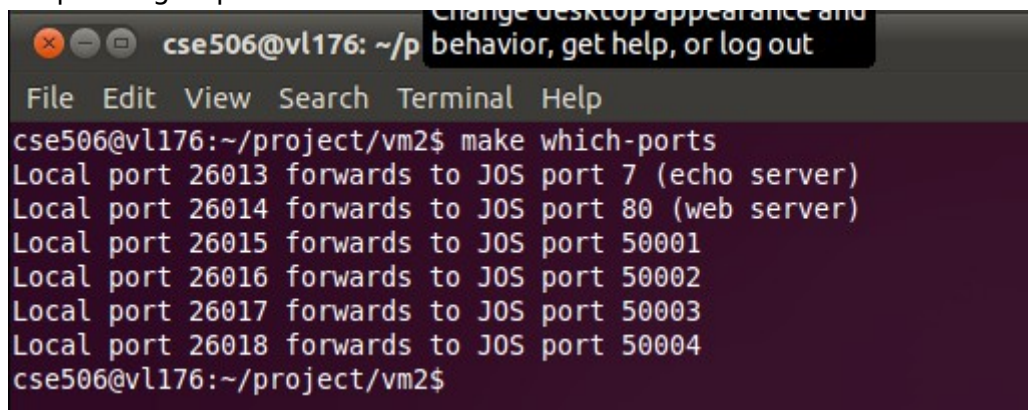
On VM-1, port 26007 is corresponding to port 50003.

A terminal window titled 'cse506@vl176: ~/project/vm1' with a menu bar (File, Edit, View, Search, Terminal, Help). The output of the 'make which-ports' command is displayed, showing a list of local ports and their corresponding JOS ports.

```
cse506@vl176:~/project/vm1$ make which-ports
Local port 26003 forwards to JOS port 7 (echo server)
Local port 26004 forwards to JOS port 80 (web server)
Local port 26005 forwards to JOS port 50001
Local port 26006 forwards to JOS port 50002
Local port 26007 forwards to JOS port 50003
Local port 26008 forwards to JOS port 50004
cse506@vl176:~/project/vm1$
```

On VM-1

On VM-2, port 26015 is corresponding to port 50001 and port 26016 is corresponding to port 50002.

A terminal window titled 'cse506@vl176: ~/p' with a menu bar (File, Edit, View, Search, Terminal, Help). The output of the 'make which-ports' command is displayed, showing a list of local ports and their corresponding JOS ports.

```
cse506@vl176:~/project/vm2$ make which-ports
Local port 26013 forwards to JOS port 7 (echo server)
Local port 26014 forwards to JOS port 80 (web server)
Local port 26015 forwards to JOS port 50001
Local port 26016 forwards to JOS port 50002
Local port 26017 forwards to JOS port 50003
Local port 26018 forwards to JOS port 50004
cse506@vl176:~/project/vm2$
```

On VM-2

By running **make qemu**, we boot up both two machines.

```
cse506@vl176: ~/project/vm1
File Edit View Search Terminal Help
superblock is good
bitmap is good
alloc_block is good
file_open is good
file_get_block is good
file_flush is good
file_truncate is good
file_rewrite is good
icode: read /motd
This is /motd, the message of the day.

Welcome to the JOS kernel, now with a file system!

icode: close /motd
icode: spawn /init
ns: 52:54:00:12:34:56 bound to static IP 10.0.2.15
NS: TCP/IP initialized.
icode: exiting
init: running
init: data seems okay
init: bss seems okay
init: args: 'init' 'initarg1' 'initarg2'
init: running sh
init: starting sh
$ Test Program, Env ID: 4100, Counter: 1
$ Test Program, Env ID: 4100, Counter: 2
$ Test Program, Env ID: 4100, Counter: 3
$ Test Program, Env ID: 4100, Counter: 4
$
```

On VM-1

By default, there is a test process on VM-1 with Environment ID 4100. It is from the **loop.c** file under user folder. It will keep running in a loop and display the counter and Environment ID. We will move this environment to VM-2 finally.

```
init: running sh
init: starting sh
$ Test Program, Env ID: 4100, Counter: 1
$ Test Program, Env ID: 4100, Counter: 2
$ Test Program, Env ID: 4100, Counter: 3
$ Test Program, Env ID: 4100, Counter: 4
$ Test Program, Env ID: 4100, Counter: 5
$ Test Program, Env ID: 4100, Counter: 6
$ pause 4100
Process 4100 is paused.
$
```

On VM-1

We can pause the process by running command **pause** on VM-1. Now the counter is 6.

```

$ ps
Process ID: 4096   Parent ID: 0   Status: Running
Process ID: 4097   Parent ID: 0   Status: Sleeping
Process ID: 4098   Parent ID: 0   Status: Sleeping
Process ID: 8195   Parent ID: 4103  Status: Running
Process ID: 4100   Parent ID: 0   Status: Sleeping
Process ID: 4101   Parent ID: 4098  Status: Running
Process ID: 4102   Parent ID: 4098  Status: Running
Process ID: 4103   Parent ID: 4099  Status: Running
Process ID: 4104   Parent ID: 4098  Status: Sleeping
Process ID: 12297  Parent ID: 8195  Status: Running
Process ID: 8202   Parent ID: 12297 Status: Running
$

```

On VM-1

We can list all processes and their status on VM-1 with command **ps** on VM-1.

```

$ resume 4100
Process 4100 is resumed.
Test Program, Env ID: 4100, Counter: 7
$ $ Test Program, Env ID: 4100, Counter: 8
$ Test Program, Env ID: 4100, Counter: 9
$ Test Program, Env ID: 4100, Counter: 10
$ Test Program, Env ID: 4100, Counter: 11
$ Test Program, Env ID: 4100, Counter: 12

```

On VM-1

We can **resume** the process on VM-1 regarding the Environment ID as the argument.

```

$ Test Program, Env ID: 4100, Counter: 13
$ Test Program, Env ID: 4100, Counter: 14
$ Test Program, Env ID: 4100, Counter: 15
$ pause 4100
Process 4100 is paused.
$

```

On VM-1

We finally **pause** the process 4100 on VM-1 and the current counter is 15.

```
$ save 4100 my_image
Save Page: 0x00800000, Sequence: 1
Save Page: 0x00800000, Sequence: 2
Save Page: 0x00800000, Sequence: 3
Save Page: 0x00800000, Sequence: 4
Save Page: 0x00801000, Sequence: 1
Save Page: 0x00801000, Sequence: 2
Save Page: 0x00801000, Sequence: 3
Save Page: 0x00801000, Sequence: 4
Save Page: 0x00802000, Sequence: 1
Save Page: 0x00802000, Sequence: 2
Save Page: 0x00802000, Sequence: 3
Save Page: 0x00802000, Sequence: 4
Save Page: 0x00803000, Sequence: 1
```

On VM-1

By running the command **save** on VM-1, we can save the running process as an image on the disk.

```
ps
pause
resume
loop
copy
migrate_recv_normal
migrate_send_normal
migrate_recv_pgfault
migrate_send_pgfault
kill
save
load
my_image
$
```

On VM-1

By running command **ls** you will find that the image file 'my\_image' is on the disk on VM-1.



```

$ load my_image
New Env Established: 4107
Load Page: 0x00800000, sequence: 1
Load Page: 0x00800000, sequence: 2
Load Page: 0x00800000, sequence: 3
Load Page: 0x00800000, sequence: 4
Load Page: 0x00801000, sequence: 1
Load Page: 0x00801000, sequence: 2
Load Page: 0x00801000, sequence: 3
Load Page: 0x00801000, sequence: 4
Load Page: 0x00802000, sequence: 1
Load Page: 0x00802000, sequence: 2
Load Page: 0x00802000, sequence: 3
Load Page: 0x00802000, sequence: 4
Load Page: 0x00803000, sequence: 1

```

On VM-1

By running the command **load** we will load the image back to the kernel as a process with Environment ID 4107 on VM-1.

```

Load Page: 0xeebfd000, sequence: 3
Load Page: 0xeebfd000, sequence: 4
Load Image Successful, You can resume New Env: 4107
$ resume 4107
Process 4107 is resumed.
$ Test Program, Env ID: 4107, Counter: 16
$ Test Program, Env ID: 4107, Counter: 17
$ Test Program, Env ID: 4107, Counter: 18
$

```

On VM-1

After we **resume** the process, the counter is 16 now on VM-1. We remember the previous counter is 15.

```

$ Test Program, Env ID: 4107, Counter: 23
$ Test Program, Env ID: 4107, Counter: 24
$ Test Program, Env ID: 4107, Counter: 25
$ Test Program, Env ID: 4107, Counter: 26
$ Test Program, Env ID: 4107, Counter: 27
$ pause 4107
Process 4107 is paused.
$

```

On VM-1

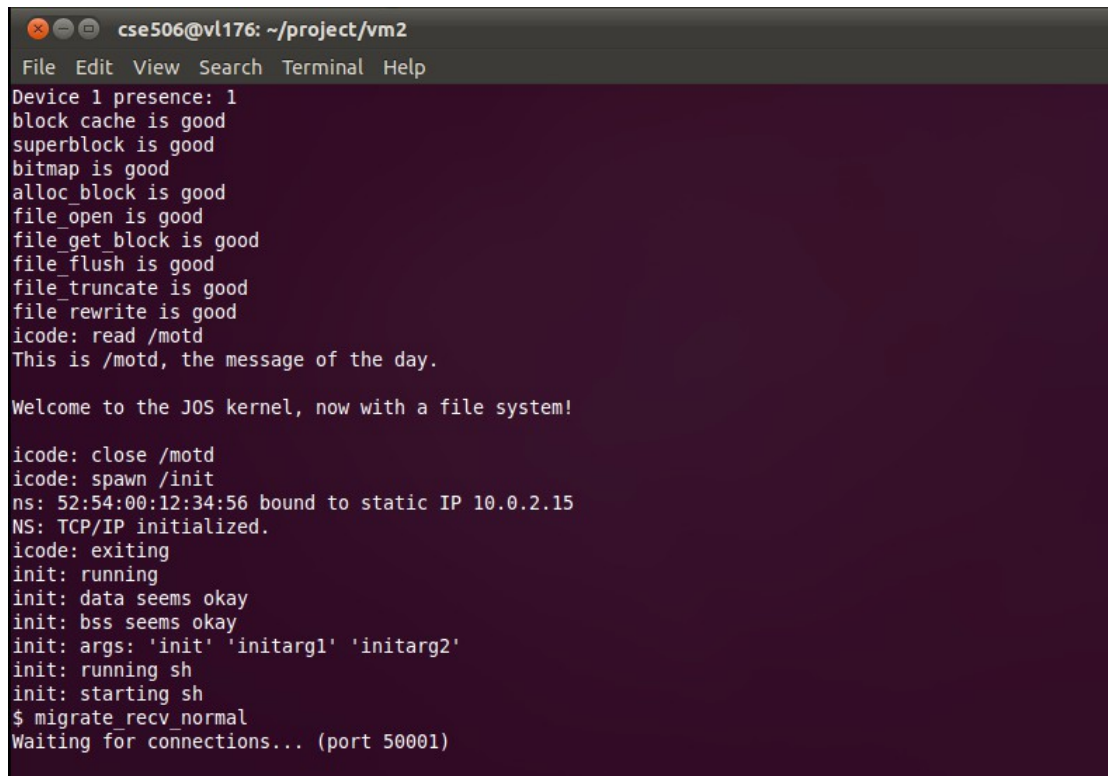
We **pause** the process on VM-1 and the last counter is 27.

We will begin the migration now. The IP address of my virtual machine is 130.245.30.176. If you want to test my program, you should use your own IP



address. You should also get the corresponding ports with ***make which-port***.

We first do the migration with the **normal method**. We will dump all pages together to the other machine.

A terminal window titled 'cse506@vl176: ~/project/vm2' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal output shows the JOS kernel boot process: 'Device 1 presence: 1', various 'is good' messages for block cache, superblock, bitmap, alloc block, file\_open, file\_get block, file\_flush, file\_truncate, file\_rewrite, and 'icode: read /motd'. It then says 'This is /motd, the message of the day.' and 'Welcome to the JOS kernel, now with a file system!'. The 'icode' process continues with 'close /motd', 'spawn /init', 'ns: 52:54:00:12:34:56 bound to static IP 10.0.2.15', 'NS: TCP/IP initialized.', 'icode: exiting', 'init: running', 'init: data seems okay', 'init: bss seems okay', 'init: args: 'init' 'initarg1' 'initarg2'', 'init: running sh', and 'init: starting sh'. Finally, the user enters '\$ migrate\_rcv\_normal' and the terminal shows 'Waiting for connections... (port 50001)'.

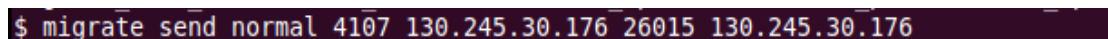
```
cse506@vl176: ~/project/vm2
File Edit View Search Terminal Help
Device 1 presence: 1
block cache is good
superblock is good
bitmap is good
alloc block is good
file_open is good
file_get block is good
file_flush is good
file_truncate is good
file_rewrite is good
icode: read /motd
This is /motd, the message of the day.

Welcome to the JOS kernel, now with a file system!

icode: close /motd
icode: spawn /init
ns: 52:54:00:12:34:56 bound to static IP 10.0.2.15
NS: TCP/IP initialized.
icode: exiting
init: running
init: data seems okay
init: bss seems okay
init: args: 'init' 'initarg1' 'initarg2'
init: running sh
init: starting sh
$ migrate_rcv_normal
Waiting for connections... (port 50001)
```

On VM-2

Run command ***migrate\_rcv\_normal*** on VM-2. The listening port is 50001. The corresponding port to port 50001 on VM-2 is 26015 on my VMs.

A terminal window showing the command '\$ migrate\_send\_normal 4107 130.245.30.176 26015 130.245.30.176' being entered.

```
$ migrate_send_normal 4107 130.245.30.176 26015 130.245.30.176
```

On VM-1

We run ***migrate\_send\_normal*** on VM-1. 4107 is the Environment ID. 26015 is the listening port on remote machine (VM-2) corresponding to 50001. The first '130.245.30.176' is the remote machine IP address. The last '130.245.30.176' is the local machine IP address.

```
$ migrate_send_normal 4107 130.245.30.176 26015 130.245.30.176
Send Page: 0x00800000, Sequence: 1
Send Page: 0x00800000, Sequence: 2
Send Page: 0x00800000, Sequence: 3
Send Page: 0x00800000, Sequence: 4
Send Page: 0x00801000, Sequence: 1
Send Page: 0x00801000, Sequence: 2
Send Page: 0x00801000, Sequence: 3
Send Page: 0x00801000, Sequence: 4
Send Page: 0x00802000, Sequence: 1
Send Page: 0x00802000, Sequence: 2
Send Page: 0x00802000, Sequence: 3
Send Page: 0x00802000, Sequence: 4
Send Page: 0x00803000, Sequence: 1
Send Page: 0x00803000, Sequence: 2
Send Page: 0x00803000, Sequence: 3
Send Page: 0x00803000, Sequence: 4
Send Page: 0x00804000, Sequence: 1
```

On VM-1

On VM-1, pages have been sent to VM-2.

```
Recv Page: 0x00805000, sequence: 1
Recv Page: 0x00805000, sequence: 2
Recv Page: 0x00805000, sequence: 3
Recv Page: 0x00805000, sequence: 4
Recv Page: 0x00806000, sequence: 1
Recv Page: 0x00806000, sequence: 2
Recv Page: 0x00806000, sequence: 3
Recv Page: 0x00806000, sequence: 4
Recv Page: 0x00807000, sequence: 1
Recv Page: 0x00807000, sequence: 2
Recv Page: 0x00807000, sequence: 3
Recv Page: 0x00807000, sequence: 4
Recv Page: 0xeebfd000, sequence: 1
Recv Page: 0xeebfd000, sequence: 2
Recv Page: 0xeebfd000, sequence: 3
Recv Page: 0xeebfd000, sequence: 4
Process Migration Finished, You can resume New Env: 4107
$
```

On VM-2

On VM-2, pages have been received and a new Environment with ID 4107 has been established.

```
$ resume 4107
Process 4107 is resumed.
$ Test Program, Env ID: 4107, Counter: 28
$ Test Program, Env ID: 4107, Counter: 29
$ Test Program, Env ID: 4107, Counter: 30
$ Test Program, Env ID: 4107, Counter: 31
$
```

On VM-2

By running **resume**, we resume the process 4107 on VM-2. The current counter is 28. We remember that the previous counter on VM-1 is 27.

```
$ kill 4107
Process 4107 is killed.
$
```

On VM-2

We kill the process 4107 by running the command **kill** on VM-2.

We have successfully moved a process from VM-1 to VM-2 with the **normal method** (without help from page fault handler).

Now we will move the process with the help of page fault handler. We call it the **page fault method**.

Process 4107 is still sleeping on VM-1.

```
You have successfully moved Env 4107 to 130.245.30.176
$ copy 4107
Copying from Env: 4107
New Env Created: 4108
$ kill 4107
Process 4107 is killed.
$
```

On VM-1

By running command **copy** we clone a new process on VM-1 with Environment ID 4108 from process 4107 and we then **kill** process 4107.

```
$ migrate_rcv_pgfault
Waiting for connections... (port 50002)
```

#### On VM-2

On VM-2, we run command 'migrate\_rcv\_pgfault'. VM-2 will listen on port 50002, which is corresponding to port 26016.

```
$ migrate_send_pgfault 4108 130.245.30.176 26016 130.245.30.176 26007
Send Page Table Entry: 0x00800000
Send Page Table Entry: 0x00801000
Send Page Table Entry: 0x00802000
Send Page Table Entry: 0x00803000
Send Page Table Entry: 0x00804000
Send Page Table Entry: 0x00805000
Send Page Table Entry: 0x00806000
Send Page Table Entry: 0x00807000
Send Page Table Entry: 0xeebfd000

Waiting for connections... (port 50003)
Waiting to handle page fault
```

#### On VM-1

On VM-1, we run command `migrate_send_pgfault 4108 130.245.30.176 26016 130.245.30.176 26007`. 4108 is the process to be moved. The first '130.245.30.176' is the IP address of the remote machine. The second '130.245.30.176' is the local IP address. 26016 is the listening port of the remote machine. The port 26007, which is corresponding to 50003 on VM-1, is the port that VM-1 will listen on to handle page fault on the remote host. VM-1 will send all presented page table entries to VM-2.

```
$ migrate_rcv_pgfault
Waiting for connections... (port 50002)
Migration:
  From: 130.245.30.176
  To: 130.245.30.176
  Env ID: 4108
New Env Established: 8203
Recv Page Table Entry: 0x00800000
Recv Page Table Entry: 0x00801000
Recv Page Table Entry: 0x00802000
Recv Page Table Entry: 0x00803000
Recv Page Table Entry: 0x00804000
Recv Page Table Entry: 0x00805000
Recv Page Table Entry: 0x00806000
Recv Page Table Entry: 0x00807000
Recv Page Table Entry: 0xeebfd000
Process Migration Finished, You can resume New Env: 8203
$
```

#### On VM-2

By running **migrate\_recv\_pgfault**, VM-2 will receive all page table entries from VM-1.

```
$ resume 8203
Process 8203 is resumed.
$
Handle User Page Fault, Env: 8203, Addr: 0x00801553
Remote Host: 130.245.30.176:26007, Env: 8203
Recv Page: 0x00801553, Sequence: 1
Recv Page: 0x00801553, Sequence: 2
Recv Page: 0x00801553, Sequence: 3
Recv Page: 0x00801553, Sequence: 4

Handle User Page Fault, Env: 8203, Addr: 0xeebdfd84
Remote Host: 130.245.30.176:26007, Env: 8203
Recv Page: 0xeebdfd84, Sequence: 1
Recv Page: 0xeebdfd84, Sequence: 2
Recv Page: 0xeebdfd84, Sequence: 3
Recv Page: 0xeebdfd84, Sequence: 4

Handle User Page Fault, Env: 8203, Addr: 0x008000a1
Remote Host: 130.245.30.176:26007, Env: 8203
Recv Page: 0x008000a1, Sequence: 1
Recv Page: 0x008000a1, Sequence: 2
Recv Page: 0x008000a1, Sequence: 3
```

On VM-2

On VM-2, we **resume** the new process 8203. Since process 8203 has no pages mapping to the page table entries. If there is a page fault, the kernel will notify the super page fault handler process. This super process will fetch the missed pages from the original machine, VM-1.



```
Waiting for connections... (port 50003)
Waiting to handle page fault

Recv Request For Page: 0x00801553
Send Page: 0x00801553, Sequence: 1
Send Page: 0x00801553, Sequence: 2
Send Page: 0x00801553, Sequence: 3
Send Page: 0x00801553, Sequence: 4

Recv Request For Page: 0xeebdfd84
Send Page: 0xeebdfd84, Sequence: 1
Send Page: 0xeebdfd84, Sequence: 2
Send Page: 0xeebdfd84, Sequence: 3
Send Page: 0xeebdfd84, Sequence: 4

Recv Request For Page: 0x008000a1
Send Page: 0x008000a1, Sequence: 1
Send Page: 0x008000a1, Sequence: 2
Send Page: 0x008000a1, Sequence: 3
Send Page: 0x008000a1, Sequence: 4

Recv Request For Page: 0x00802e44
Send Page: 0x00802e44, Sequence: 1
Send Page: 0x00802e44, Sequence: 2
Send Page: 0x00802e44, Sequence: 3
Send Page: 0x00802e44, Sequence: 4
```

On VM-1

VM-1 will block and handle the page fault on VM-2. In my program, VM-1 will keep blocking, because there is no non-blocking network service in JOS.

```
Handle User Page Fault, Env: 8203, Addr: 0x00802e44
Remote Host: 130.245.30.176:26007, Env: 8203
Recv Page: 0x00802e44, Sequence: 1
Recv Page: 0x00802e44, Sequence: 2
Recv Page: 0x00802e44, Sequence: 3
Recv Page: 0x00802e44, Sequence: 4

Handle User Page Fault, Env: 8203, Addr: 0x008030e4
Remote Host: 130.245.30.176:26007, Env: 8203
Recv Page: 0x008030e4, Sequence: 1
Recv Page: 0x008030e4, Sequence: 2
Recv Page: 0x008030e4, Sequence: 3
Recv Page: 0x008030e4, Sequence: 4
Test Program, Env ID: 8203, Counter: 28
$ Test Program, Env ID: 8203, Counter: 29
$ Test Program, Env ID: 8203, Counter: 30
$ Test Program, Env ID: 8203, Counter: 31
$ Test Program, Env ID: 8203, Counter: 32
$ Test Program, Env ID: 8203, Counter: 33
$ Test Program, Env ID: 8203, Counter: 34
$ Test Program, Env ID: 8203, Counter: 35
$ Test Program, Env ID: 8203, Counter: 36
$ Test Program, Env ID: 8203, Counter: 37
$ Test Program, Env ID: 8203, Counter: 38
$ Test Program, Env ID: 8203, Counter: 39
```

On VM-2

Now, the new process works well on VM-2.

The process migration with **page fault method** is finished.